Sudoku

- Lucrare pentru atestarea Competențelor Profesionale -

Cuprins:

1. Introducere
2. Regulile jocului
3. Motivația alegerii temei
4. Structura aplicației
5. Detalii tehnice si de implementare
6. Modalități de utilizare
7. Posibilități de dezvoltare
8. Funcții foarte relevante pentru program
9. Bibliografie
10. Introducere

**Sudoku** este un joc de puzzle numeric, în care scopul este să umpli o grilă de 9x9 cu cifre, astfel încât fiecare coloană, fiecare rând și fiecare sub-grilă de 3x3 să conțină toate cifrele de la 1 la 9, fără a exista cifre duplicate în nicio coloană, niciun rând și nicio sub-grilă. Jocul este considerat un exercițiu excelent pentru îmbunătățirea abilităților de gândire logică și de rezolvare a problemelor.

1. Regulile jocului
2. Grila de joc este formată din 81 de pătrățele împărțite în 9 sub-grile de 3x3.
3. Scopul jocului este să umpleți fiecare pătrățel cu o cifră de la 1 la 9.
4. Fiecare coloană, fiecare rând și fiecare sub-grilă de 3x3 trebuie să conțină toate cifrele de la 1 la 9.
5. Nu există cifre duplicate în nicio coloană, niciun rând și nicio sub-grilă.
6. Motivația

Am ales pentru proiectul de atestat jocul Sudoku deoarece consider că este o provocare destul de mare în a-l implementa, iar dificultățile întâlnite pe parcurs au fost fel de fel:

1. Realizarea unei interfețe grafice pentru joc, care să cuprindă tabla de joc de dimensiune 9 x 9, împărțită la rândul ei în 9 submatrici de 3 x 3, reprezentând fiecare câte un cadran al jocului.

2. Funcționalitățile acestuia, precum implementarea unui algoritm care să genereze atât o matrice inițială pentru jucător, cât și o soluție validă, validarea regulilor în timp real (fiecare linie, coloană și submatrice de 3 x 3 trebuie să conțină doar valori distincte), dar și crearea altor utilități extra precum: timer, numărul de greșeli permise, oferirea de indicii în cazul în care jucătorul se blochează, dar și recompensarea acestuia cu indicii bonus, ce depind de performanța sa din timpul jocului. O altă funcție extra adăugată este abilitatea jucătorului de a-și configura tabla de Sudoku cu ajutorul unor comenzi simple.

De asemenea, a fost o oportunitate foarte bună de a mai exersa și învăța principii noi de OOP, care stau la baza atât limbajului de programare C++, cât și a altor limbaje de programare precum: C#, Java, Kotlin și Python.

*4.* Detalii tehnice și de implementare

*a. Rolul claselor*

Pentru implementarea jocului Sudoku am folosit o structură bazată pe clase si headere, fiecare clasă având un scop diferit.

1. Game – Clasa Game se ocupă cu inițializarea tuturor variabilelor (Instanța clasei Table, Instanța clasei Menu, tipul programului, etc.), a window-ului din librăria Winbgim folosit pentru reprezentarea grafică a matricei de joc, cât și a diferitelor stiluri grafice. Tot din clasa Game este controlat jocul și sunt interogate acțiuni precum (plasarea unei cifre, eliminarea unei cifre, oferirea unui indiciu, scăderea vieților, etc.) folosind clase incluse în clasa Game. Clasa Game este clasa principală, rădăcina arborelui.
2. Table – se ocupă cu tot ce ține de aria de proximitate a matricei, respectiv inițializarea de pătrate, verificarea condițiilor necesare funcționării jocului, desenarea pătratelor folosind culori prestabilite, plasarea cifrelor corespunzătoare în casete, gestionarea hint-urilor și vieților.
3. Square – se ocupă cu tot ce ține de aria lui de lucru: valoarea care se află înăuntrul lui, desenarea marginilor, cât și o funcție apelată din cadrul clasei Table de autodesenare.
4. Solver – se ocupă de generarea matricei pe care jucătorul trebuie să o rezolve, dar și de generarea soluției unice specifice rundei. Sunt folosite două metode diferite de generare, prima fiind ceva mai neeficientă, dar care garantează un procent de randomizare a matricei foarte ridicat, pe când cea de-a doua este foarte rapidă, dar se bazează pe utilizarea unui pattern ce ar putea fi dedus de către jucător dacă este atent la poziția cifrelor.
5. Menu – se ocupă de interacțiunea cu jucătorul, acesta putând configura din cadrul meniului/consolei tipul de scenariu (joc normal sau simulare), design-ul mapei de joc inițiale la care este adăugat și un preview pentru o ușoară remediere în cazul unui design nesatisfăcător.
6. Config – se ocupă doar de păstrarea unei constante precum (latura pătratului, culoarea pătratului, poziția hint-ului), includerea de librării sau a unor funcții ce sunt folosite în cadrul întregului program.

Fiecare clasă are asociat un header (.h) , unde sunt declarate prototipurile funcțiilor, variabile membre sau alte librării ajutătoare. Implementările funcțiilor sunt după regăsite in file-ul .cpp, fiecare din .cpp fiind incluse în compilarea programului. De menționat că in fiecare header sunt folosite Guard-uri, concept ce are rolul de a preveni includirea multiplă de clase într-un file, astfel evitând erori de compilare greu de detectat după.

*b. Metode de generare pentru Sudoku*

Prima metodă, cea mai eficientă, dar care se folosește de un pattern bazat pe permutarea permutărilor, funcționează în felul următor:

Să considerăm permutarea: 1 5 3 6 9 2 7 4 8. Dacă prima linie a tablei de joc ar fi mereu o permutare, putem observa că regula caracteristică liniei va fi mereu respectată. Acum, pentru a rezolva matricile de 3 x 3, vom permuta la dreapta permutarea actuală de 3 ori, obținând: 7 4 8 1 5 3 6 9 2. Aceasta va fi a doua linie. În felul acesta, vom menține pe întreaga perioadă a algoritmului matricile de 3 x 3 valide. Mai rămâne doar problema coloanelor. După ce ajungem la permutarea 6 9 2 7 4 8 1 5 3, observăm că putem pentru următorul set de matrice să schimbăm puțin ordinea, permutând o singură dată la dreapta, obținând 3 6 9 2 7 4 8 1 5. Astfel, condiția va fi respectată și pe coloane, întrucât e ca și cum am schimba setul de 3 elemente inițiale. Pentru prima coloană ar fi 1, 7, 6, trecând după la 3, 8, 2.

Pentru implementarea acestui algoritm avem nevoie să:

Generăm toate permutările de 9 elemente, din care după alegem una pentru a o permuta.

Alegem o permutare random din cele 9! generate și o permutăm după regulile de mai sus.

Pentru a obține matricea jucătorului, eliminăm x elemente random și obținem și matricea de joc, dar și soluția. Algoritmul este foarte rapid.

A doua metodă, mai puțin optimizată decât prima, se folosește de următoarea strategie:

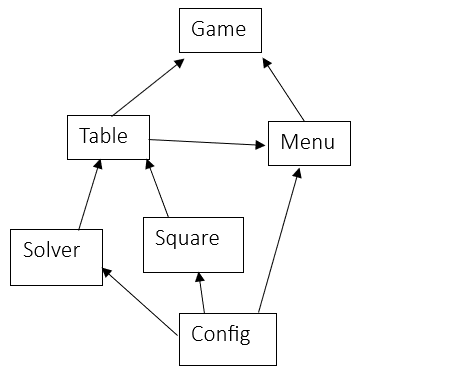
Să considerăm matricele de 3x3 de pe diagonala principală. Putem observa că aceste 3 matrici de 3x3 nu se afectează reciproc, deci pot fi umplute individual. Astfel, selectăm o permutare random din cele 9! generate și umplem matricea de 3x3 cu ea. Am eliminat din 81 de poziții ce trebuiesc umplute 27. Acum, pentru a umple cele 54 de poziții rămase, folosim un algoritm de tip backtracking, care va încerca să plaseze numere până când rămânem fără. Această variantă este mult mai înceată față de prima, dar compensează prin corectitudinea din punct de vedere al randomizării de care dă dovadă.

*c. Realizarea graficii*

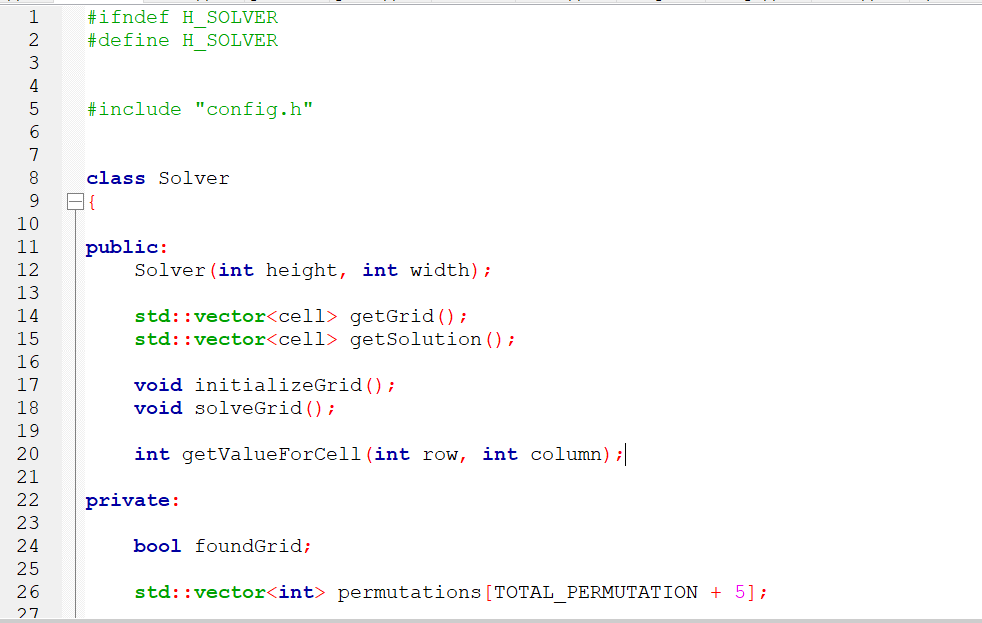
Pentru grafică, am folosit librăria **WinbGim**, cu diferite funcții pentru realizarea tutoror task-urilor. Pentru desenarea pătratelor, am folosit funcția line împreună cu o formulă matematică pentru a automatize procesul de desenare a celor 81 de pătrate. Tot folosind funcția line am desenat și marginile, ce pot fi schimbate de către jucător.

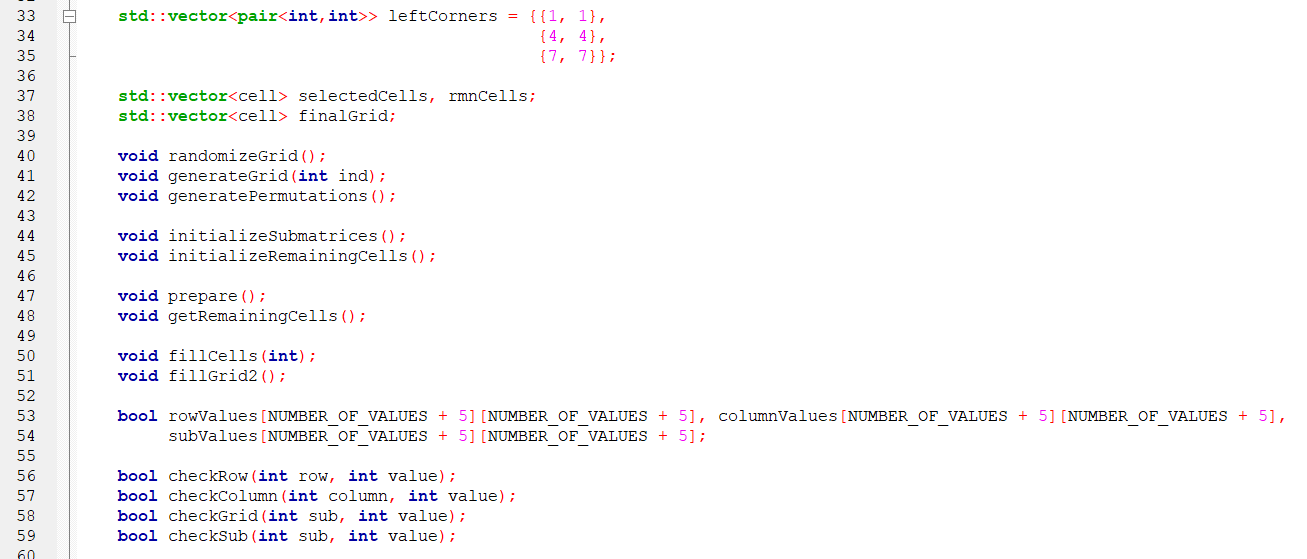
Pentru afișarea diferitelor cifre, am folosit funcția outtext. Cât pentru colorarea pătratelor, am folosit funcția floodfill, care primește ca un punct de referință din care începe un algoritm de tip fill, oprindu-se în momentul în care întâlnește culoarea specificată în parametru. Alte funcții precum setcolor, setfillstyle sau settextstyle au fost folosite strict pentru marimea, look-ul si înclinarea font-ului.

Pentru lucrul cu imagini, funcția readimagefile rezolvă tot, fiind foarte ușor de utilizat. Imagini am folosit pentru punctele de viață, numărul de hint-uri si end screen-urile de la joc.

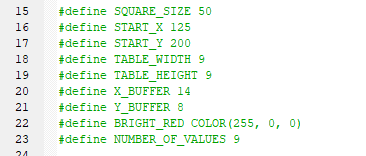
5. Structura aplicației

O reprezentare grafică a arborelui de clase.

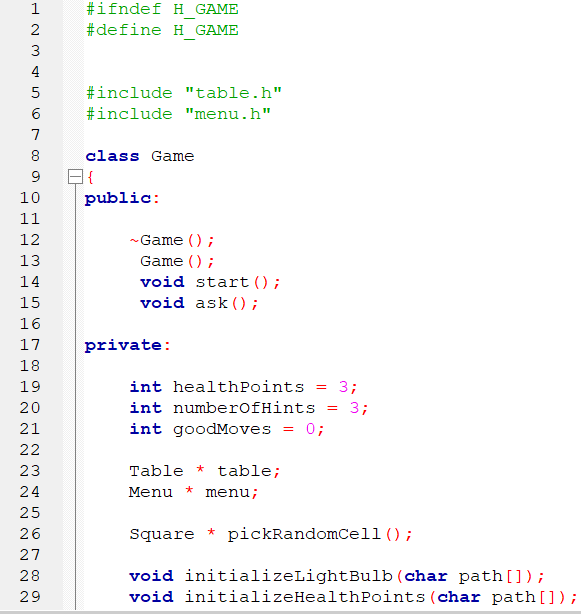
1. Clasa Solver

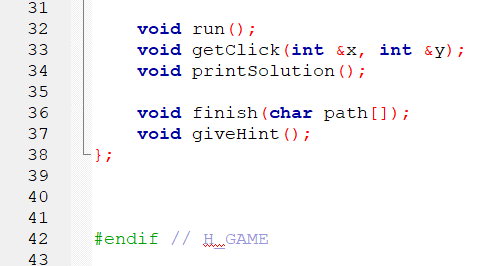


1. Clasa Config(conține doar define-uri sau funcții utilitare)

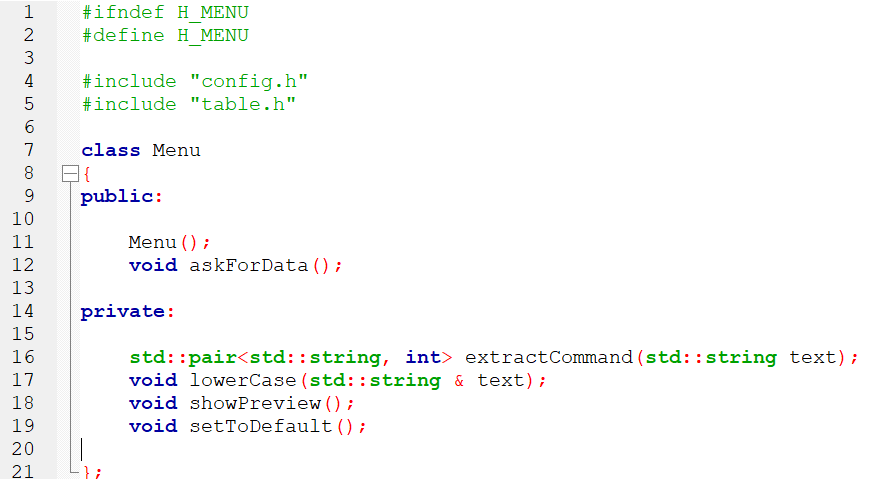


1. Clasa Game (conține doar define-uri sau funcții utilitare)

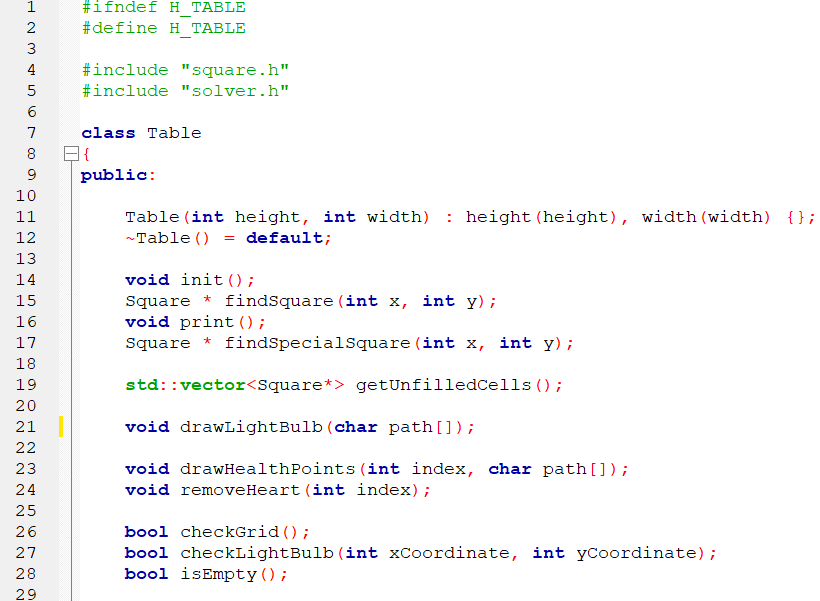
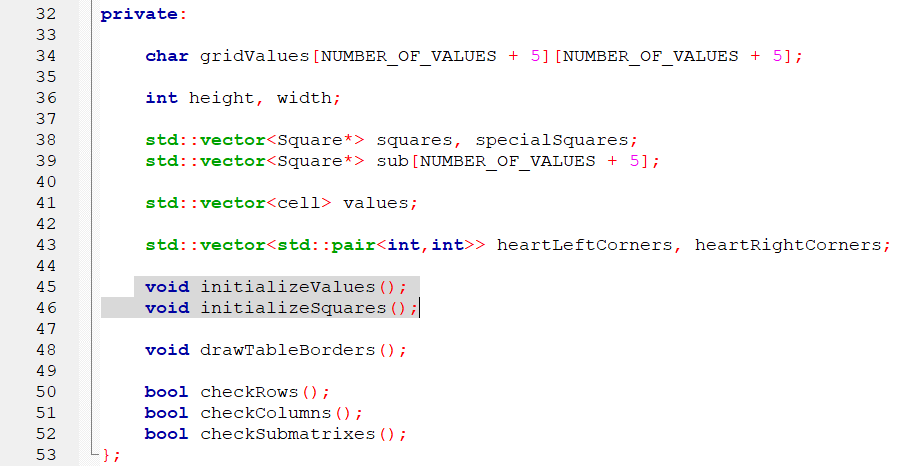




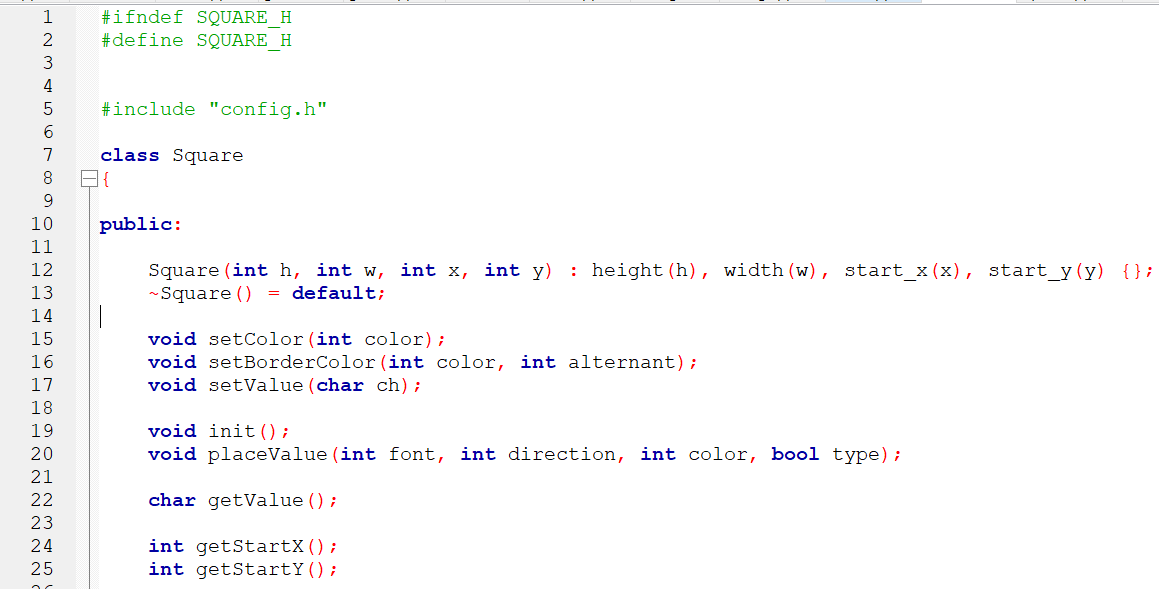
1. Clasa Menu

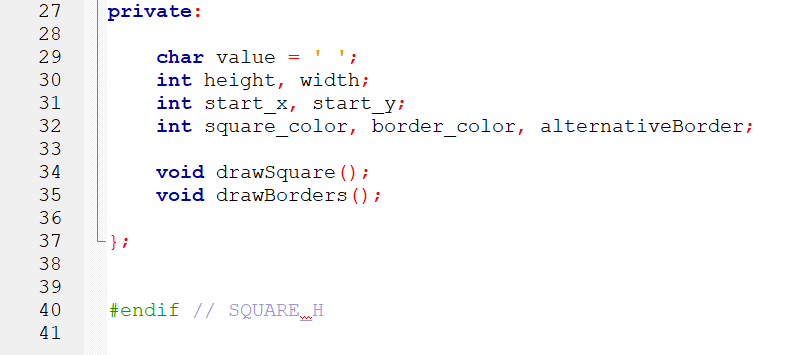


1. Clasa Table



1. Clasa Square





6. *Modalități de utilizare*

1. Antrenarea creierului: Sudoku este un joc de logică și îi poate ajuta pe jucători să își îmbunătățească abilitățile de rezolvare a problemelor și să-și antreneze creierul. Jucând regulat Sudoku, jucătorii pot dezvolta strategii și tehnici de rezolvare a problemelor, precum și să își îmbunătățească capacitatea de concentrare și de luare a deciziilor.
2. Socializarea: Sudoku poate fi, de asemenea, o modalitate excelentă de a socializa. Jucând cu prietenii sau cu membrii familiei, poți transforma jocul într-o activitate de grup distractivă și interactivă. De asemenea, poți împărtăși puzzle-urile cu prietenii tăi prin intermediul rețelelor sociale sau a aplicațiilor de mesagerie și puteți discuta strategiile și soluțiile găsite.
3. Competiția: Sudoku poate fi un joc competitiv, în special atunci când joci împotriva altor jucători sau încerci să rezolvi puzzle-ul într-un timp cât mai scurt. Această abordare poate fi o modalitate excelentă de a-ți testa abilitățile de rezolvare a problemelor și de a-ți îmbunătăți performanțele.
4. *Posibilități de dezvoltare*
5. O modalitate de a îmbunătăți jocul Sudoku este de a adăuga un mod de joc împotriva computerului. În acest mod de joc, jucătorii se vor confrunta cu un adversar controlat de computer. Computerul va fi programat să rezolve cât mai rapid posibil jocul, astfel încât jucătorul să fie provocat să rezolve jocul într-un timp cât mai scurt. Aceasta poate fi o modalitate excelentă de a îmbunătăți abilitățile de rezolvare a puzzle-ului Sudoku ale jucătorilor și poate adăuga o doză de competitivitate.
6. O altă idee de dezvoltare a jocului Sudoku este de a adăuga funcții sociale. Acest lucru poate fi realizat prin conectarea jucătorilor prin intermediul rețelelor sociale, astfel încât să poată să se confrunte cu prietenii lor în jocul Sudoku. De asemenea, poți adăuga o funcție de împărtășire a puzzle-urilor, astfel încât jucătorii să poată crea propriile lor puzzle-uri și să le ofere prietenilor lor să le rezolve.
7. O altă idee interesantă este de a adăuga puzzle-uri personalizate. În acest mod, jucătorii pot crea propriile lor puzzle-uri, pe care le pot rezolva apoi ei sau le pot oferi prietenilor lor să le rezolve. Aceasta poate adăuga o doză de creativitate în joc și poate face ca jucătorii să se simtă implicați și interesați.

*8. Funcții foarte relevante pentru program*

*1. Clasa Square*

a. drawSquare()

@params: fără parametrii;

@return: desenează un pătrat între coordonatele, având coordonatele colțului stânga sus x1, y1 și a colțului dreapta jos x2, y2 folosind funcția line. Coordonatele pătratului sunt transmise prin intermediul constructorului, la fei si dimensiunea pătratului.

b. placeValue(int, int, int, bool)

@params: @font – fontul folosit pentru valoarea pătratului

@direction – orientarea textului

@color – culoarea textului

@squareType – această variabilă este folosită pentru a diferenția pătratele pentru primirea input-ului față de cele pentru primirea output-ului.

@return: plasează în interiorul unei casete valorea pe care jucătorul o solicită și este corectă

1. *Clasa Menu*

a. askForData()

@params: fără parametrii

@return: asteaptă input de la jucător, ce este folosit pentru a determina : configurația tablei de joc, dificultatea jocului, afișarea unui preview pentru tabla de joc, cât și tipul jocului, simulare sau normal.

1. *Clasa Game*

a. run()

@params: fără parametrii

@return: gestionează toate acțiunile jucatorului legate de : plasarea unei valori, identificarea unei acțiuni greșite si aplicarea penalizărilor, identificarea unui sfârșit de joc(câștig sau înfrângere), cât și oferirea de ajutor jucătorului prin hint-uri.

b. getClick(int, int)

@params: @&x – prin intermediul acestei variabile este returnată poziția x a mouse-ului în momentul unui click

@&y – prin intermediul acestei variabile este returnată poziția y a mouse-ului în momentul unui click

@return: folosind alte funcții din librăria WinbGim, ismouseclick și getmouseclick pentru a inregistra interacțiunile cu jucătorul.

1. *Clasa Solver*

a. generatePermutations()

@params: fără parametrii

@return: generează toate permutările de 9 elemente

b. fillGrid2()

@params: fără parametrii

@return: umple tabla de joc cu valori ce sunt permutate succesiv, folosind o permutare de 9 elemente aleasă random

c. fillCells(int)

@params: @ind – a câta celulă este umplută

@return: umple table de joc cu valori ce sunt încercate pe rând până la găsirea unei soluții, asemănător unui algoritm de tip backtracking. Valorile sunt alese într-o ordine aleatorie

1. *Clasa Table*

a. drawHealthPoints(int, char[])

@params: @index – indexul inimii

@path – locația în care se află imaginea inimă

@return: desenează o inimă în funcție de poziția ei în intefață și index

b. drawLightBulb(char [])

@params: @path – locația în care se află imaginea cu becul

@return: desenează un buc în funcție de poziția lui în intefață

c. findSquare(int, int)

@params: @x – poziția pe axa Ox

@y - poziția pe axa Oy

@return: returneză pointerul către pătratul specific poziției în plan cartezian, sau NULL dacă nu există un pătrat in zona respectivă

d. initializeSquares()

@params: fără parametrii

@return: creează în memorie obiectele de tip pătrat și le configurează după mai multe criterii(valoare inițială, culoare, font, culoarea marginii). În această funcție sunt inițializate atât casetele destinate cifrelor, cât și cele folosite ca butoane pentru plasarea unei cifre pe tablă.

*9. Bibliografie*

Website-uri care m-au ajutat foarte mult în realizarea produsului soft sunt:

1. Documentația WinbGim: <https://home.cs.colorado.edu/~main/bgi/doc/>
2. StackOverflow: https://stackoverflow.com/questions/30821356/multiple-definition-first-defined-here-errors
3. Refactoring Guru: https://refactoring.guru/design-patterns/cpp