Algoritam za logičku zagonetku minolovac u programskom jeziku C#

Horvat, Oliver

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

ALGORITAM ZA LOGIČKU ZAGONETKU MINOLOVAC U PROGRAMSKOM JEZIKU C#

Završni rad

Oliver Horvat

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 10.07.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Oliver Horvat
Studij, smjer:	Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4496, 27.07.2020.
OIB Pristupnika:	91483479547
Mentor:	doc. dr. sc. Tomislav Rudec
Sumentor:	izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Naslov završnog rada:	Algoritam za logičku zagonetku minolovac u programskom jeziku C#
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Student će izraditi aplikaciju za rješavanje logičke zagonetke minolovac (minesweeper) u programskom jeziku C#. Sumentor s FERIT-a: Alfonzo Baumgartner Tema je zauzeta za Olivera Horvata.
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	10.07.2023.
Datum potvrde ocjene od strane Odbora:	17.07.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 17.07.2023.

Ime i prezime studenta:	Oliver Horvat
Studij:	Programsko inženjerstvo
Mat. br. studenta, godina upisa:	R4496, 27.07.2020.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Algoritam za logičku zagonetku minolovac u programskom jeziku C#**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Rudec

i sumentora izv. prof. dr. sc. Alfonzo Baumgartner

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Minolovac (engl. *minesweeper*) je poznata računalna igrica. Veliki broj ljudi je upoznat s njom jer se od 1992. godine, točnije od izlaska *Windows* 3.1 operacijskog sustava, do 2012. godine, kada je izašao Windows 8, nalazila kao unaprijed instalirana igra u *Windows* operacijskim sustavima. U sadašnjem vremenu se može besplatno instalirati sa službene *Microsoft* stranice. Osim *Windows* minolovca, postoje razne verzije igrice, a najpopularnija je *minesweeper.online* koja omogućuje igranje bez preuzimanje igre te sadrži razne značajke kao bilježenje statistike ili pružanje mogućnost dopisivanja s ostalim igračima.

Minolovac je logička igra koja se sastoji od ploče koja sadrži određeni broj redaka i stupaca polja koje mogu sadržavati mine. Broj redaka, stupaca i mina ovise o odabranoj težini igre. Težina „početnik“ najčešće sadrži 9 redaka i 9 stupaca polja s 10 mina, težina „srednji“ najčešće sadrži 16 redaka i 16 stupaca polja s 40 mina te težina „stručnjak“ najčešće sadrži 30 redaka i 16 stupaca polja s 99 mina. Cilj minolovca je otkriti sva polja koja ne sadrže minu uz pomoć brojeva koji otkrivaju koliko susjednih polja otkrivenog polja sadrže minu. Susjedna polja nekog polja su 8 polja koja okružuju to polje. U kvalitetnim verzijama igre se pritiskom na otkriveno polje otkrivaju njegova susjedna polja. Kada sazna gdje se nalazi mina, igrač ima mogućnost označiti polje zastavicom kako se kasnije ne bi zabunio ili ga ne bi otkrio pritiskom na otkriveno polje.

Dva glavna pristupa igranju minolovca su rješavanje u najkraćem mogućem vremenu i rješavanje s obzirom na stopu pobjede. Kada se igra pokušava riješiti u najkraćem mogućem vremenu igrač više riskira kod otvaranja polja i pokušava smanjiti broj pritisaka tipki miša kako bi postigao optimalno vrijeme. Pri pristupu rješavanja s obzirom na stopu pobjede igrač igra polako i prilikom pogađanja pokušava otkriti polja koja će mu omogućiti najbolje šanse u postizanju danjeg napretka u ploči.

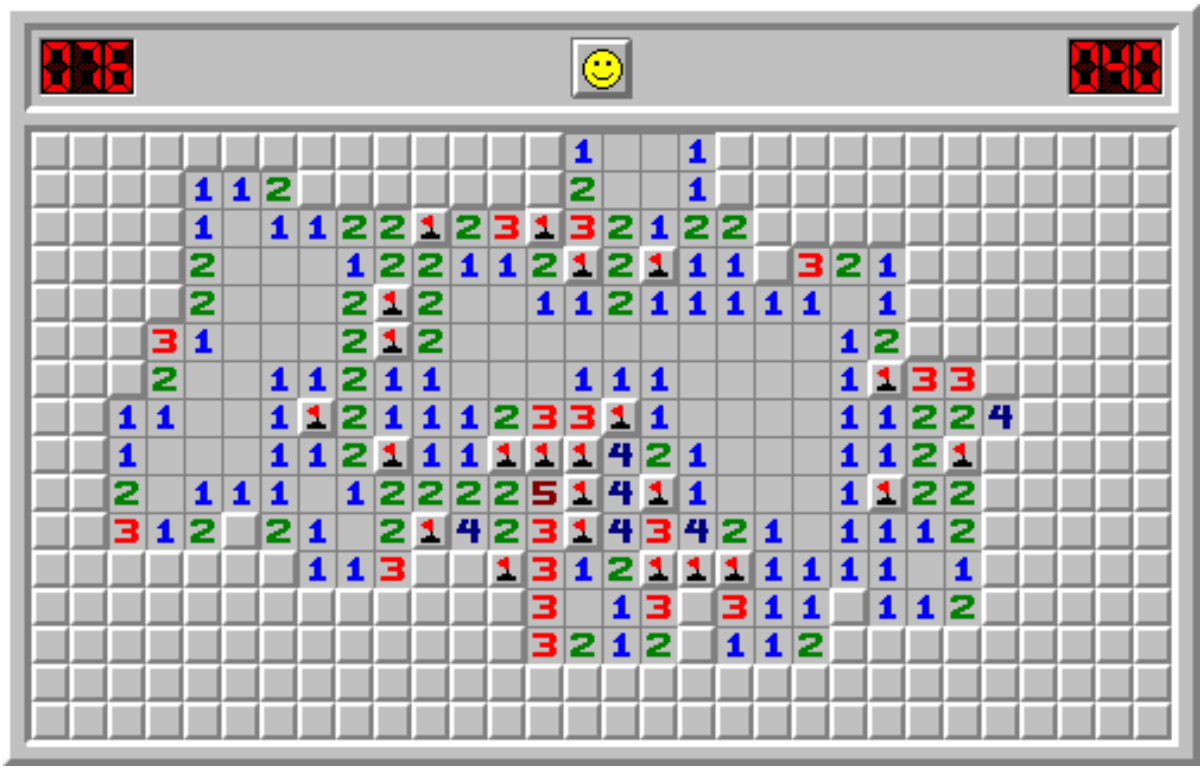
Minolovac se razlikuje od velikog broja logičkih igara zbog toga što je veliki broj generiranih ploča nemoguće riješiti bez pogađanja, a to je uzrokovano nasumičnim odabirom polja koje sadrže minu. U nekim verzijama igre postoje postavke koje namještaju da svaka ploča bude rješiva samo primjenom logike, bez situacija u kojima je potrebno pogađati.

Podaci s *minesweeper.online* iz 15. lipnja 2023. godine otkrivaju da je:

- stopa pobjede na težini „početnik“ 46.6% s najbržim postignutim vremenom 0.606 s,

- stopa pobjede na težini „srednji“ 22.3% s najbržim postignutim vremenom 7.198 s,
- stopa pobjede na težini „stručnjak“ 2.6% s najbržim postignutim vremenom 29.461 s,
- odigrano je 2 094 544 028 polja sa ukupnom stopom pobjede 11.4%.

Tema ovog završnog rada je izrada algoritma koji će igrati minolovac sa što većom stopom pobjede.



Slika 1.1 Djelomično riješena ploča igre minolovac

1.1. Zadatak završnog rada

Zadatak završnog rada je napraviti algoritam u programu *C#* koji će generirati ploče igre minolovca za težine: „početnik“ , „srednji“ i „stručnjak“, rješavati ih korak po korak te izmjeriti stopu pobjede algoritma.

2. Radovi vezani uz područje minolovac

Ubrzo nakon ulaska minolovca među unaprijed instalirane igre u *Windows* operacijskim sustavima 1992. godine, počeli su nastajati radovi o igri minolovac.

Godine 1994. profesor Mark Holliday sa sveučilišta Western Carolina University je u svom radu [6] pisao o svom istraživanju o korištenju entuzijazma studenata za igranje računalnih igrica kao poticaj za vježbanje razvijanja programerske logike. Dva od pet zadataka su bila vezana uz implementiranje igre minolovac.

Godine 1997. Philip Crow je u svom radu [7] uveo notacijski sustav za igru i definirao nekoliko teorema o uzorcima koji opisuju određene situacije u ploči igre koje imaju jedinstveno rješenje.

Godine 2004. su Andrew Fowler i Andrew Young za svoj završni rad [8] na sveučilištu Washington State University pisali o raznim pristupima igranju minolovca koristeći linearnu algebru.

Godine 2015. David Becerra je za svoj završni rad [9] na sveučilištu Harvard University pisao o algoritamskim pristupima rješavanja minolovca.

Na internetskim repozitorijima, kao što su *GitLab* ili *GitHub,* postoje razni algoritmi otvorenog koda koji se bave rješavanjem minolovca. Dva pristupa za stvaranje algoritama za rješavanje igre minolovac koja se nalaze na internetskim repozitorijima su:

- korištenje umjetne inteligencije,
- razvoj algoritma za rješavanje problema.

Najbolje ocijenjeni algoritam za rješavanje minolovca na *GitHubu* [10] autora Bai Li (korisničko ime „*luckytoilet*“) je napisan u programskom jeziku Java. U algoritmu je za rješavanje korištena umjetna inteligencija.

3. ALATI ZA IZRADU ALGORITMA

Algoritam je napisan u integriranom razvojnom okruženju *Microsoft Visual Studio 2019*, kod je napisan u jeziku *C#*, a za spremanje napretka je korišten internetski repozitorij *GitLab* uz pomoć alata *Git*.

3.1. *Microsoft Visual Studio 2019*

Microsoft Visual Studio je integrirano razvojno okruženje koje se koristi za pisanje, uređivanje i otklanjanje pogrešaka u kodu te izgradnju i stvaranje izvršne datoteke projekata. Podržava razne jezike kao što su *C++*, *C#*, *JavaScript*, *TypeScript*, *Python*. Sadrži veliki broj alata koji omogućuju efikasnije kodiranje. Pri instalaciji omogućuje odabir alata ili jezika koje korisnik želi koristiti da se izbjegne instalacija onoga što korisnik ne planira koristiti. Ukoliko se korisnik predomisli, uvijek može naknadno instalirati sve što je preskočio u instalaciji.

Slika 3.1.1. *Microsoft Visual Studio* logo

3.2. *GitLab i Git*

Gitlab je platforma za razvoj, sigurnost i operacije (engl. *DevSpecOps*) koja je pokretana umjetnom inteligencijom. Najčešće se koristi za ekipni razvoj i pohranu projekata.

Slika 3.2.1. *GitLab* logo

Koristi se zajedno *Git-om* koji je besplatna i javno dostupna aplikacija koja omogućuje neovisni istovremeni rad na projektu. Glavna svrha *Git-a* je spajanje različitih verzija projekata tako da se spreme samo napravljene promjene. Osim toga, pomoću *Git-a* i *GitLab*-a vrlo se jednostavno vratiti na starije verzije projekta.

```
These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects
```

Slika 3.2.2. Često korištene *Git* naredbe

3.3. Osnove objektno orijentiranog programiranja i programski jezik *C#*

Objektno orijentirano programiranje je način programiranja koji omogućuje prilagodbu programskog jezika problemu, a ne prilagodbu problema programskog jeziku. Često se koristi za rješavanje složenih problema. Razlog tome je što ono omogućuje lako enkapsuliranje podataka. Tri najvažnije značajke objektno orijentiranog programiranja su: enkapsulacija, nasljeđivanje i polimorfizam.

Enkapsulacija je grupiranje podataka u jednu cjelinu koja ih povezuje te se postiže uporabom klase. Klasa je novi tip podataka kojeg stvara programer. U njoj se opisuju stanja i ponašanja nekog objekta iz stvarnog svijeta kojeg se želi opisati. U stanja spadaju atributi koji su varijable unutar klase, a u ponašanja spadaju metode koje su funkcije unutar klase.

Instanca klase naziva se objekt klase, njemu klasa služi kao „nacrt“. Objekt može biti atribut neke klase i taj odnos se naziva *has-a* jer jedna klasa sadrži objekt neke druge klase. Ako se želi napraviti klasa koja predstavlja žarulju može se napraviti ovako:

- stanje:
    - je upaljena (*boolean* tip atributa)
- ponašanje:
    - upali ili ugasi (*void* tip metode, pozivom mijenja stanje „je upaljena“),
    - dohvati stanje žarulje (*bool* tip metode, pozivom vraća vrijednost atributa „je upaljena“)

Objekt se stvara pozivom konstruktora. Konstruktor je posebna metoda koja ima isto ime kao klasa i nema povratni tip. Služi za postavljanje vrijednosti stanja objekta.

Nasljeđivanje je značajka objektno orijentiranog programiranja koje omogućuje da nova klasa koja se naziva izvedena klasa poprimi obilježja neke klase koja se naziva osnovna klasa. Taj odnos klasa naziva se *is-a* jer je jedna klasa podvrsta druge klase. Ako se želi napraviti klasa koja predstavlja *LED* žarulju može se napraviti ovako:

- led žarulja nasljeđuje klasu žarulja
- stanje:
    - boja *LED* žarulje (*string* tip atributa)
- ponašanje:
    - dohvati boju žarulje (*string* tip metode, pozivom vraća vrijednost atributa „boja *LED* žarulje“)

Na taj način će *LED* žarulja sadržati sva stanja i ponašanja osnovne klase i ona prethodno navedena.

Polimorfizam je značajka objektno orijentiranog programiranja koja omogućuje da postoji više metoda istog povratnog tipa i naziva, ali se razlikuju prema svojoj implementaciji. To se najčešće postiže na način da se postavi drugačija količina ili kombinacija tipova podataka u argumente tako da klasa može prepoznati koju od tih metoda algoritam želi pozvati, to jest preopterećivanjem metode.

Unutar klase postoje različita prava pristupa članovima klase kako bi se ograničila prava korisnika koji ne bi trebali imati pristup određenim članovima klase, a ona su:

- javno: svi mogu pristupiti članu klase,
- privatno: stanja i ponašanja pripadne klase mogu pristupiti članu klase,
- zaštićeno: stanja i ponašanja pripadne klase i od nje izvedene klase mogu pristupiti članu klase.

Programski jezik *C#* je moderni, objektno orijentirani i neovisan o platformi programski jezik. Temeljen je na jeziku *C*, to jest koriste sličnu sintaksu. Osim tipova podataka iz programskog jezika *C*, *C#* sadrži tip podatka *string* za zapis teksta. Za rukovanje pogreškama koristi *try-catch* blokove koji su jednostavniji za korištenje od načina koji se koristi u *C-u*.

C# je jezik više razine, to jest koristi „sakupljač smeća“. „Sakupljač smeća“ je automatski upravljač memorijom. Koristi se da programer ne mora misliti o ručnom zauzimanju i oslobađanju memorije.

4. ALGORITAMSKI PRISTUP RJEŠAVANJU PROBLEMA GENERIRANJA I RJEŠAVANJA MINOLOVCA

U ovom poglavlju je objašnjeno na koji način funkcionira algoritam za generiranje i rješavanje minolovca.

Algoritam je podijeljen u 5 klasa:

- *PredefinedNumbers,*
- *Field,*
- *Board,*
- *BoardSolver,*
- Program.

4.1. Klasa *PredefinedNumbers*

Klasa „*PredefinedNumbers*“ je statička klasa, što znači da se ne može *instancirati*, to jest napraviti objekt te klase. Statičke klase se koriste za dohvaćanje nekih vrijednosti i pozivanje određenih funkcija. U nju su spremljene vrijednosti dimenzija ploče i broj mina koje sadrži za različite težine minolovca. Jedini razlog njezinog postojanja je izbjegavanje „čarobnih brojeva“. „Čarobni broj“ je miris u kodu, pokušava se izbjeći jer uvodi nerazumljivost u čitanju algoritma. To je broj koji se nalazi negdje u kodu, a ne zna se što predstavlja.

```
9 references
public static class PredefinedNumbers
{
    public static readonly int rowsBeginner = 9;
    public static readonly int columnsBeginner = 9;
    public static readonly int mineCountBeginner = 10;

    public static readonly int rowsIntermediate = 16;
    public static readonly int columnsIntermediate = 16;
    public static readonly int mineCountIntermediate = 40;

    public static readonly int rowsExpert = 16;
    public static readonly int columnsExpert = 30;
    public static readonly int mineCountExpert = 99;

}
```

Slika 4.1.1. Klasa „*PredefinedNumbers*“

4.2. Klasa *Field*

Klasa „*Field*“ je klasa koja predstavlja jedno polje unutar ploče igre minolovca. Sadrži atribute:

- *row* (*int* tip atributa, predstavlja redak ploče u kojem se polje nalazi),
- *column* (*int* tip atributa, predstavlja stupac ploče u kojem se polje nalazi),
- *mine* (*boolean* tip atributa, predstavlja sadrži li polje minu),
- *adjacentMines* (*int* tip atributa, predstavlja vrijednost koliko susjednih polja tog polja sadrži minu),
- *revealed* (*bool* tip atributa, predstavlja je li polje otkriveno),
- *flagged* (*bool* tip atributa, predstavlja je li polje označeno sa zastavicom).

Atributima *row*, *column* i *mine* se pridružuje vrijednost kroz konstruktor, a ostalim vrijednostima je privremeno postavljena vrijednost 0 ili *false*.

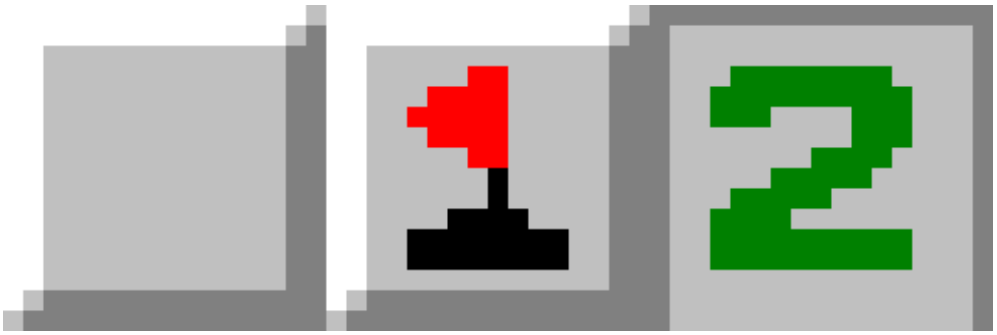
```
private int row;
private int column;
private bool mine;
private int adjacentMines;
private bool revealed;
private bool flagged;

3 references
public Field(int row, int column, bool mine)
{
    this.row = row;
    this.column = column;
    this.mine = mine;
    this.adjacentMines = 0;
    this.revealed = false;
    this.flagged = false;
}
```

Slika 4.2.1. Atributi i konstruktor klase „*Field*“

Sadrži metode:

- *IsMine* (*bool* tip metode, pozivom vraća vrijednost atributa „*mine*“, ova metoda se koristi za generiranje ploče, ispis ploče i otkrivanje pogreške u rješavanju ploče),

- *IsRevealed* (*bool* tip metode, pozivom vraća vrijednost atributa „*revealed*“, ova metoda se koristi za rješavanje ploče),
- *IsFlagged* (*bool* tip metode, pozivom vraća vrijednost atributa „*flagged*“, ova metoda se koristi za rješavanje ploče),
- *GetRow* (*int* tip metode, pozivom vraća vrijednost atributa „*row*“, ova metoda se koristi za navigaciju po ploči),
- *GetColumn* (*int* tip metode, pozivom vraća vrijednost atributa „*column*“, ova metoda se koristi za navigaciju po ploči),
- *GetAdjacentMines* (*int* tip metode, pozivom vraća vrijednost atributa „*adjacentMines*“, ova metoda se koristi za rješavanje ploče),
- *SetAdjacentMines* (*void* tip metode, pozivom pridružuje određenu vrijednost atributu „*adjacentMines*“, ova metoda se koristi za generiranje ploče),
- *Reveal* (*void* tip metode, pozivom pridružuje vrijednost „*true*“ atributu „*revealed*“, ova metoda se koristi za rješavanje ploče),
- *Flag* (*void* tip metode, pozivom pridružuje vrijednost „*true*“ atributu „*flagged*“, ova metoda se koristi za rješavanje ploče).



Slika 4.2.2. Primjeri polja u minolovcu

Na slici 4.2.2. je prikazano tri primjera polja. U klasi *Field* prvo polje bi imalo vrijednosti atributa:

- *revealed = false*,
- *flagged = false*.

Drugo polje bi imalo vrijednosti atributa:

- *revealed = false*,
- *flagged = true*.

Treće polje bi imalo vrijednosti atributa:

- *revealed = true,*
- *adjacentMines* = 2.

Ostale vrijednosti atributa nisu vidljive iz slike.

4.3. Klasa *Board*

Klasa „*Board“* je klasa koja predstavlja ploču igre minolovac. Ona ima uspostavljen odnos *has-a* s klasom „*Field“* jer sadrži listu njezinih objekata. Sadrži atribute:

- *rows* (*int* tip atributa, predstavlja broj redova koji se nalaze u ploči),
- *columns* (*int* tip atributa, predstavlja stupaca koje se nalaze u ploči),
- *mineCount* (*int* tip atributa, predstavlja broj polja koje sadrže minu u ploči),
- *fields* (*List<Field>* tip atributa, predstavlja vrijednost koja sadrži sva polja u ploči, lista je kolekcija koja služi za grupiranje podataka u jednu varijablu),
- *printFlag* (*bool* tip podatka, predstavlja dopuštenje ispisa postupka rješavanja minolovca, koristi se pri računanju stope pobjede jer je naredba za ispis u terminal spora)

Atributima *rows*, *columns* i *mineCount* se pridružuje vrijednost iz klase *PredefinedNumbers* ovisno o odabranoj težini ploče poslane kroz konstruktor. Atributu *printFlag* se pridružuje vrijednost kroz konstruktor.

Sadrži metode:

- *GetFlaggedCount* (*int* tip metode, pozivom vraća broj polja u ploči kojima atribut „*flagged*“ ima vrijednost *true*, ova metoda se koristi za rješavanje ploče),
- *GetRevealedCount* (*int* tip metode, pozivom vraća broj polja u ploči kojima atribut „*revealed*“ ima vrijednost *true*, ova metoda se koristi za rješavanje ploče),
- *GetTotalMineCount* (*int* tip metode, pozivom vraća vrijednost atributa „*mineCount*“, ova metoda se koristi za rješavanje ploče),
- *GetRows* (*int* tip metode, pozivom vraća vrijednost atributa „*rows*“, ova metoda se koristi za navigaciju po ploči),
- *GetColumns* (*int* tip metode, pozivom vraća vrijednost atributa „*columns*“, ova metoda se koristi za navigaciju po ploči),

- *GetTotalFieldCount* (*int* tip metode, pozivom vraća ukupan broj ploča na polju, ova metoda se koristi za rješavanje ploče),
- *IsInsideBoard* (*bool* tip metode, pozivom vraća nalazi li se polje u granicama ploče, ova metoda se koristi za sprječavanje pokušaja navigacije izvan granice ploče),
- *GetField* (*Field* tip metode, pozivom vraća polje koje se nalazi u retku i stupcu navedenim u argumentima metode, ova metoda se koristi za navigaciju po ploči),
- *GetFields* (*List<Field>* tip metode, pozivom vraća vrijednost atributa „*fields*“, ova metoda se koristi za rješavanje ploče),
- *GetNeighbours* (*List<Field>* tip metode, pozivom vraća listu polja koja sadrži susjedna polja onog polja navedenog u argumentu metode, ova metoda se koristi za rješavanje ploče),
- *GenerateBoard* (*void* tip metode, pozivom generira ploču minolovca),
- *AddAdjacentMines* (*void* tip metode, pozivom pridružuje odgovarajuću vrijednost „*adjacentMines*“ svakom polju unutar ploče, ova metoda se koristi za generiranje ploče),
- *PrintBoard* (*void* tip metode, pozivom u terminal ispisuje generirano polje),
- *PrintCurrentBoard* (*void* tip metode, pozivom ispisuje u terminal riješeni i neriješeni dio ploče na temelju atributa svakog polja),
- *PrintChange* (*void* tip metode, pozivom u terminal ispisuje na koji način i pomoću kojih se polja napravio korak u rješavaju ploče te poziva metodu „*PrintCurrentBoard*“, preopterećena je da postoji pet istoimenih metoda, poziva se ona prigodna situaciji).

```
6 references
public void PrintChange(Field field, String message)
{
    if (printFlag == true)
    {
        Console.WriteLine("\n" + message);
        Console.Write("ROW: ");
        Console.Write(field.GetRow() + 1);
        Console.Write(", COLUMN: ");
        Console.Write(field.GetColumn() + 1);
        Console.Write(", ADJACENT MINES: ");
        Console.Write(field.GetAdjacentMines());
        PrintCurrentBoard();
    }
}
```

Slika 4.3.1. Primjer metode „*PrintChange*“ s dva argumenta

```
3 references
public void PrintChange(Field field1, Field field2, String message)
{
    if (printFlag == true)
    {
        List<Field> fields = new List<Field>();
        fields.Add(field1);
        fields.Add(field2);
        Console.Write("\n" + message);
        foreach (Field field in fields)
        {
            Console.Write("\nROW: ");
            Console.Write(field.GetRow() + 1);
            Console.Write(", COLUMN: ");
            Console.Write(field.GetColumn() + 1);
            Console.Write(", ADJACENT MINES: ");
            Console.Write(field.GetAdjacentMines());
        }
        PrintCurrentBoard();
    }
}
```

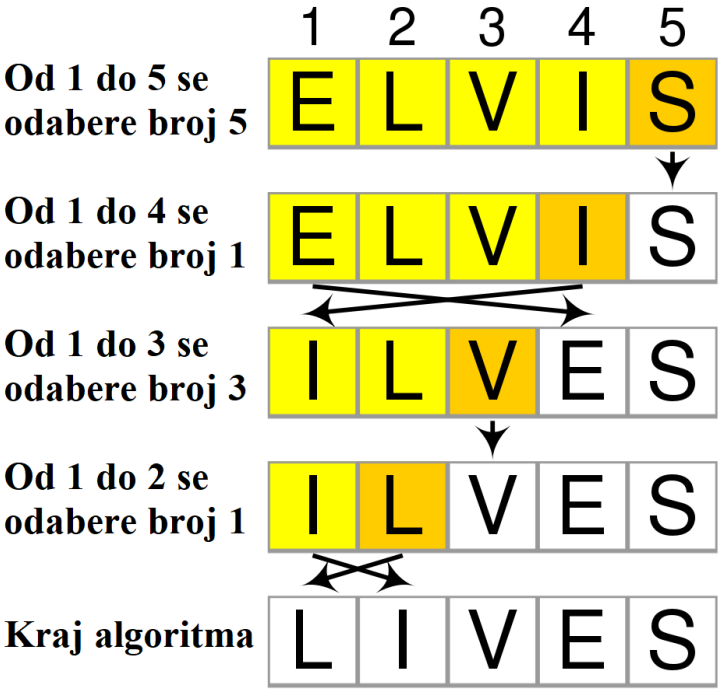
Slika 4.3.2. Primjer metode „*PrintChange*“ s tri argumenta

4.3.1. Generiranje ploče

Generiranje ploče se odvija u metodi „*GenerateBoard*“. Odvija se na način da se napravi lista *bool* vrijednosti zvana „*tempFields*“ te se popunjava s onoliko *true* vrijednosti koliko ploča treba sadržavati polja koja sadržavaju minu, to jest koliko iznosi varijabla „*mineCount*“. Nakon toga se popunjava za jednu manje od onoliko *false* vrijednosti koliko ploča treba sadržavati polja koja ne sadržavaju minu.

Onda se obavlja miješanje liste koristeći „*Fisher–Yates* algoritam za miješanje“. Taj algoritam funkcionira na način da se odabere zadnji član liste i jedan nasumičan član liste pa ta dva člana mijenjaju pozicije (osim ako se nasumično odabere zadnji član liste). Član na zadnjoj poziciji liste se nadalje ne koristi u algoritmu i član prije njega se počinje tretirati kao zadnji član te se postupak ponavlja dok ne ponestane članova koji se koriste u algoritmu.

Kada završi miješanje liste se na kraj liste dodaje vrijednost *false* te se cijela lista obrne tako da prva vrijednost uvijek bude *false*. To se radi iz razloga što je u minolovcu prvo otkrivanje polja uvijek sigurno, a algoritam će uvijek počinjati rješavati s prvim poljem. Prvo polje je najbolje polje za započeti rješavanje jer se logički nerješive situacije najčešće nalaze u kutovima ploče, pa se na taj način uklanja jedna od, potencijalno, logički nerješivih situacija.



Slika 4.3.1.1. *Fisher–Yates* algoritam za miješanje

Nakon toga se stvaraju objekti klase „*Field*“ tako da im se u konstruktor stavlja određeni broj retka, stupca i vrijednost pripadnog člana liste „*tempFields*“ koji predstavlja sadrži li polje minu. Svaki objekt klase „*Field*“ se sprema u listu *fields.*

Na kraju se poziva metoda „*AddAdjacentMines*“ koja pridružuje pripadajuću vrijednost „*adjacentMines*“ svakom članu liste *fields* nakon što provjeri koliko se mina nalazi u njemu susjednim poljima.

4.4. Klasa *BoardSolver*

Klasa „*BoardSolver*“ je klasa koja sadrži alate za rješavanje ploče minolovac. Ona ima uspostavljen odnos *has-a* s klasom „*Field*“ i klasom „*Board*“ jer sadrži njihove objekte. Sadrži atribute:

- *board* (*Board* tip atributa, predstavlja ploču za čije rješavanje se koristi),
- *dummyField* (*Field* tip atributa, predstavlja polje koje se ne nalazi u ploči, koristi se kao privremena vrijednost određenih varijabla),
- *currentlyUnsolvableField* (*Field* tip atributa, predstavlja prvo polje koje nije moguće riješiti osnovnom logikom igre nakon zadnje učinjene promjene tijekom rješavanja, koristi se za zaustavljanje beskonačne petlje u algoritmu).

Atributu „*board*“ se vrijednost pridružuje kroz konstruktor.

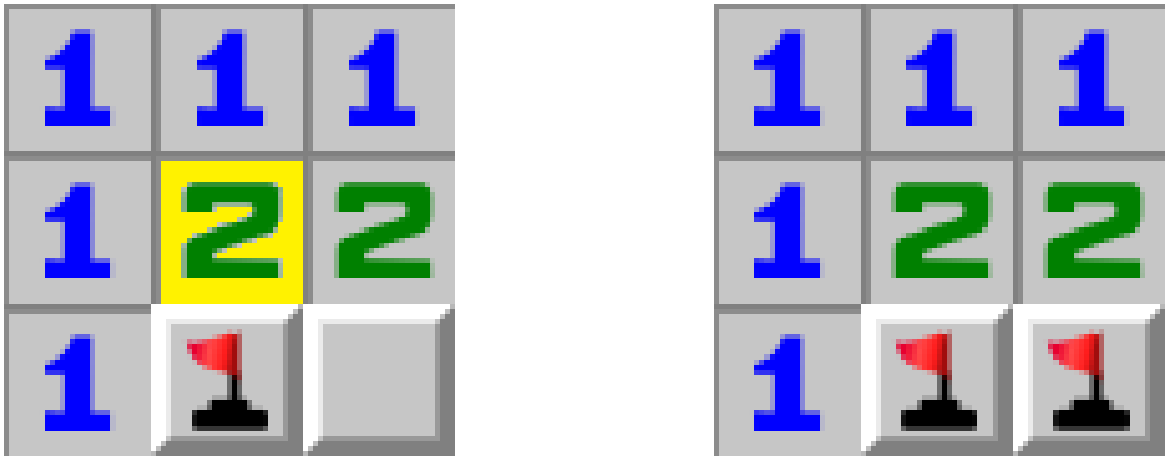
Sadrži metode:

- *FirstClock* (*List<Field>* tip metode, pozivom otkriva prvo polje u ploči, sprema ga u listu te ju vraća),
- *WasAlreadyTried* (*bool* tip metode, pozivom vraća vrijednost nalazi li se algoritam u beskonačnoj petlji),
- *IsTheSameField* (*bool* tip metode, pozivom vraća vrijednost jesu li dva polja navedena u argumentima metode isto polje),
- *RemoveOrMoveField* (*Field* tip metode, pozivom provjerava može li se postići još nekakav napredak u rješavanju ploče uz pomoć određenog polja te ga se na temelju vraćenog rezultata uklanja ili pomiče na kraj liste polja koja se trenutno, potencijalno, mogu riješiti osnovnom logikom igre),
- *FlagNeighbours* (*void* tip metode, pozivom označuje zastavicom, to jest pridružuje *true* vrijednost atributu „*flagged*“ svakom nepoznatom susjednom polju onog polja navedenog kao argument metode),
- *TryFlagging* (*void* tip metode, pozivom provjerava može li se svako susjedno polje onog polja navedenog kao argument metode označiti zastavicom te ako se može poziva metodu *FlagNeighbours*),
- *TrySolvingField* (*List<Field>* tip metode, pozivom provjerava mogu li se susjedna polja onog polja navedenog kao argument metode otkriti te ako se mogu ih otkriva i vraća listu novih polja koja se, potencijalno, mogu riješiti logikom),
- *LookForPattern11* (*List<Field>* tip metode, pozivom traži uzorak u ploči koji ima jedinstveno rješenje te ga rješava i vraća listu novih polja koja se, potencijalno, mogu riješiti osnovnom logikom igre),
- *LookForPattern12* (*List<Field>* tip metode, pozivom traži uzorak u ploči koji ima jedinstveno rješenje te ga rješava i vraća listu novih polja koja se, potencijalno, mogu riješiti osnovnom logikom igre),
- *LookForPattern14* (*List<Field>* tip metode, pozivom traži uzorak u ploči koji ima jedinstveno rješenje te ga rješava i vraća listu novih polja koja se, potencijalno, mogu riješiti osnovnom logikom igre),

- *LookForCorner131* (*List<Field>* tip metode, pozivom traži uzorak u ploči koji ima jedinstveno rješenje te ga rješava i vraća listu novih polja koja se, potencijalno, mogu riješiti osnovnom logikom igre),
- *LookForCorner222* (*List<Field>* tip metode, pozivom traži uzorak u ploči koji ima jedinstveno rješenje te ga rješava i vraća listu novih polja koja se, potencijalno, mogu riješiti osnovnom logikom igre),
- *LookForSquare0112* (*List<Field>* tip metode, pozivom traži uzorak u ploči koji ima jedinstveno rješenje te ga rješava i vraća listu novih polja koja se, potencijalno, mogu riješiti osnovnom logikom igre),
- *TryCompletingTheBoard* (*void* tip metode, pozivom pokušava završiti rješavanje ploče na temelju broja preostalih mina i nepoznatih polja),
- *TryGuessing* (*Field* tip metode, pokušava procijeniti koje je nepoznato polje najbolje za otkriti te ga otkriva).

4.4.1. Osnovna logika za označavanje polja zastavicom

Osnovna logika za označavanje polja zastavicom se obavlja na način da se u otkrivenom polju provjerava broj koji predstavlja koliko njemu susjednih polja sadrži minu te se broje nepoznata susjedna polja i zastavicom označena susjedna polja tog polja. Ako je zbroj broja nepoznatih susjednih polja i zastavicom označenih susjednih polja tog polja jednak broju koji predstavlja koliko njemu susjednih polja sadrži minu, tada se nepoznata susjedna polja tog broja označuju zastavicom.



Slika 4.4.1.1. Primjena osnovne logike za označavanje polja zastavicom

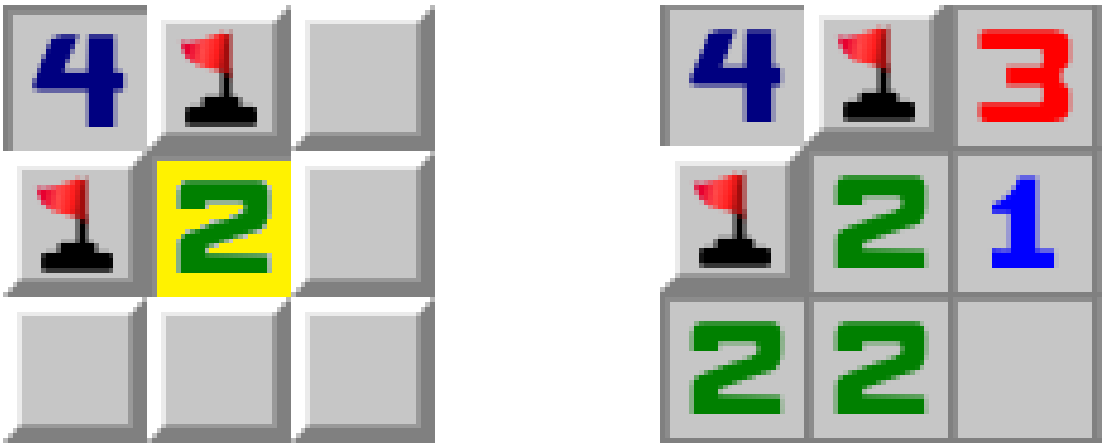
Primjena osnovne logike za označavanje polja zastavicom se u algoritmu odvija u metodi „*TryFlagging*“.

```
1 reference
public void TryFlagging(Field field)
{
    List<Field> neighbours = board.GetNeighbours(field);
    int flagCounter = 0;
    int possibleFlagCounter = 0;

    foreach (Field neigbour in neighbours)
    {
        if (neigbour.IsFlagged() == true)
        {
            flagCounter++;
        }
        else if (board.IsInsideBoard(neigbour) == true && neigbour.IsRevealed() == false)
        {
            possibleFlagCounter++;
        }
    }
    if (possibleFlagCounter + flagCounter == field.GetAdjacentMines())
    {
        FlagNeighbours(field);
    }
}
```

Slika 4.4.1.2 Metoda „*TryFlagging*“

4.4.2. Osnovna logika za otkrivanje polja

Osnovna logika za otkrivanje polja se obavlja na način da se u otkrivenom polju provjerava broj koji predstavlja koliko njemu susjednih polja sadrži minu te se broje zastavicom označena susjedna polja tog polja. Ako je broj zastavicom označenih susjednih polja tog polja jednak broju koji predstavlja koliko njemu susjednih polja sadrži minu, tada se nepoznata susjedna polja tog polja otkrivaju.



Slika 4.4.2.1. Primjena osnovne logike za otkrivanje polja

Primjena osnovne logike za otkrivanje polja se u algoritmu odvija u metodi „*TrySolvingField*“. Osim toga, metoda „*TrySolvingField*“ obavlja i bilježenje neuspješnog pokušaja postizanja napretka u rješavanju ploče promatrajući određeno polje i njegova susjedna polja te se vrijednost tog određenog polja sprema u atribut „*currentlyUnsolvableField*“ ako je trenutna vrijednost „*currentlyUnsolvableField*“ jednaka vrijednosti atributa „*dummyField*“. Nakon uspješnog otkrivanja polja, u „*currentlyUnsolvableField*“ se sprema polje „*dummyField*“ jer je došlo do promjene u ploči koja je, potencijalno, učinila dosadašnja nerješiva polja rješivim. Taj postupak postoji da omogući zaustavljanje algoritma kada se ponovno počnu provjeravati ista polja s kojima se ne može ostvariti napredak u rješavaju.

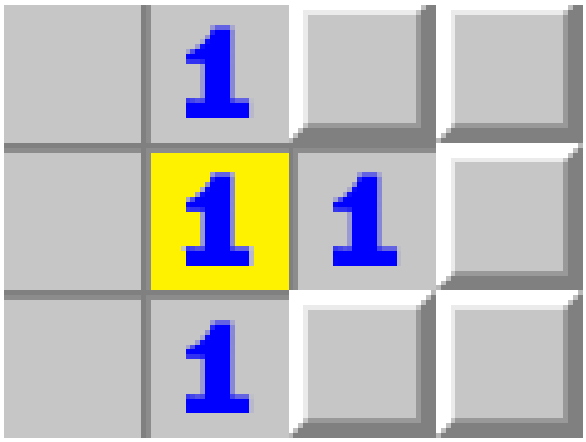
```
1 reference
public List<Field> TrySolvingField(Field field)
{
    List<Field> neighbours = board.GetNeighbours(field);
    List<Field> possibleFields = new List<Field>();
    int flagCounter = 0;
    bool printFlag = false;

    foreach (Field neigbour in neighbours)
    {
        if (neigbour.IsFlagged() == true)
        {
            flagCounter++;
        }

        else if (board.IsInsideBoard(neigbour) == true && neigbour.IsRevealed() == false)
        {
            possibleFields.Add(neigbour);
        }
    }
    if (flagCounter == field.GetAdjacentMines())
    {
        foreach (Field possibleField in possibleFields)
        {
            possibleField.Reveal();
            this.currentlyUnsolvableField = dummyField;
            printFlag = true;
        }

        if (printFlag == true)
        {
            board.PrintChange(field, "Changes made using the field: ");
        }
    }
    else
    {
        possibleFields = new List<Field>();
        possibleFields.Add(field);
        if (IsTheSameField(currentlyUnsolvableField, dummyField) == true)
        {
            this.currentlyUnsolvableField = field;
        }
    }
    return possibleFields;
}
```

Slika 4.4.2.2 Metoda „*TrySolvingField*“

4.4.3. Rješavanje prepoznavanjem uzoraka

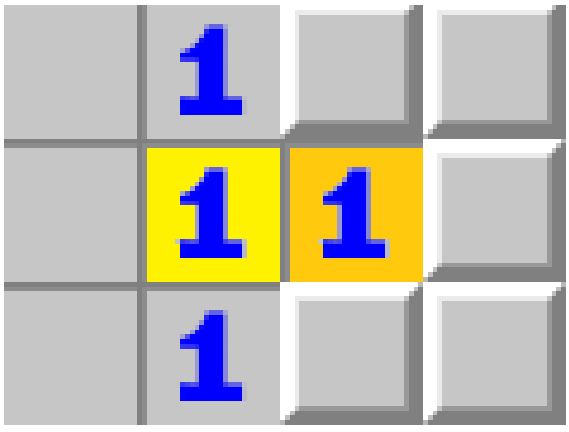
U minolovcu postoje uzorci koji opisuju određene situacije u ploči te imaju jedinstveno rješenje koje omogućuje napredak u ploči.

Uzorak 1-1 se rješava tako da prvo pronalazi otkriveno polje kojemu broj koji predstavlja koliko njemu susjednih polja sadrži minu iznosi 1 (na slici 4.4.3.1 označeno žutom bojom).



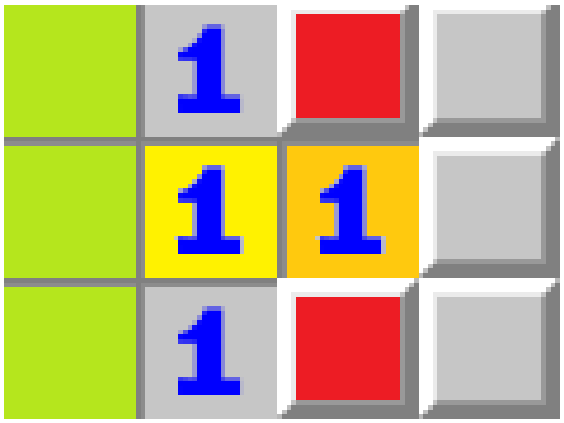
Slika 4.4.3.1. Prvi korak u rješavanju prepoznavanjem uzorka 1-1

Onda se traži njemu iznad, ispod, s lijeva ili s desna otkriveno polje kojemu broj koji predstavlja koliko njemu susjednih polja sadrži minu također iznosi 1 (na slici 4.4.3.2 označeno narančastom bojom).



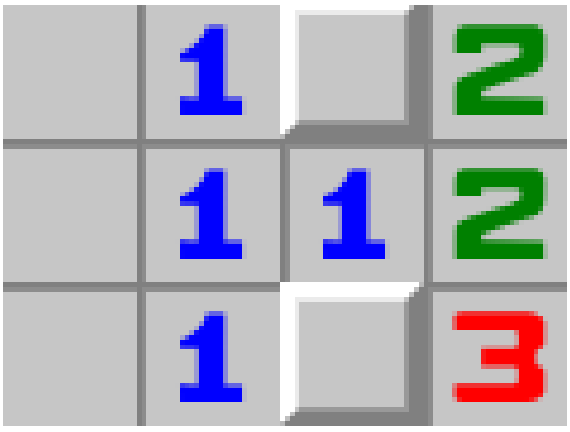
Slika 4.4.3.2. Drugi korak u rješavanju prepoznavanjem uzorka 1-1

Onda se gleda jesu li susjedna polja žuto označenog polja koja su sa suprotne strane od narančasto označenog polja otkrivena ili izvan granice ploče (na slici 4.4.3.3 označena zelenom bojom), ako jesu to znači da jedno od preostalih neotkrivenih susjednih polja žuto označenog polja mora sadržavati minu (na slici 4.4.3.3 označena crvenom bojom).



Slika 4.4.3.3. Treći korak u rješavanju prepoznavanjem uzorka 1-1

Kada se sada pogleda polje označeno narančastom bojom i zna se da jedno od polja označenih crvenom bojom sadržava minu tada je poznato da su njegova preostala nepoznata susjedna polja (neoznačena bojom) sigurna za otkriti jer ne sadržavaju minu te se otkrivaju.



Slika 4.4.3.4. Četvrti korak u rješavanju prepoznavanjem uzorka 1-1

Nekada uzorak ima drugačiji oblik nego što njegovo ime predlaže. Na primjer, uzorak 1-1 može biti zamaskiran u uzorak 1-2 sa zastavicom. To se zove redukcija. Kada se gleda broj koji predstavlja koliko polju susjednih polja sadrži minu, a jedno od njegovih susjednih polja je označeno sa zastavicom, tada to polje označeno sa zastavicom možemo promatrati kao otkriveno polje, a broj koji predstavlja koliko polju susjednih polja sadrži minu promatrati kao za jedan manji broj.

Slika 4.4.3.5. Uzorak 1-1 u reduciranom obliku

Primjena rješavanja prepoznavanjem uzorka 1-1 se u algoritmu odvija u metodi „*LookForPattern11*“. Prvo se stvara prazna lista polja „*fields*“ koja će služiti kao povratna vrijednost, petljom se prolazi kroz cijelu listu polja ploče i obavljaju se prva dva prethodno objašnjena koraka. Algoritam uzima u obzir reducirane oblike uzorka. Na slici 4.4.3.6. je prikazan dio koda koji provjerava desno susjedno polje pronađenog polja, a kasnije u kodu se provjerava donje susjedno polje pronađenog polja.

```
List<Field> fields = new List<Field>();

foreach (Field field in board.GetFields())
{
    if (field.IsRevealed() == false || field.IsFlagged() == true)
    {
        continue;
    }
    List<Field> neighbours = board.GetNeighbours(field);
    int flagCounter = 0;
    foreach (Field neighbour in neighbours)
    {
        if (neighbour.IsFlagged() == true)
        {
            flagCounter++;
        }
    }
    if (field.GetAdjacentMines() - flagCounter == 1)
    {
        Field adjacentField = board.GetField(field.GetRow() + 1, field.GetColumn() + 2);

        if (adjacentField.IsRevealed() == true)
        {
            neighbours = board.GetNeighbours(adjacentField);
            flagCounter = 0;
            foreach (Field neighbour in neighbours)
            {
                if (neighbour.IsFlagged() == true)
                {
                    flagCounter++;
                }
            }
            if (adjacentField.GetAdjacentMines() - flagCounter == 1)
            {
```

Slika 4.4.3.6. Prvi i drugi korak u rješavanju prepoznavanjem uzorka 1-1 u kodnom obliku

Nakon toga se polja bitna za rješavanje prepoznavanjem uzorka 1-1 spremaju u varijable i poziva se lokalna funkcija „*SolvePattern11*“ te se njena povratna vrijednost, što su polja koja su otkrivena tijekom izvođenja, sprema u listu „*fields*“. Drugim pozivom lokalne funkcije „*SolvePattern11*“ s različitim redoslijedom upisivanja polja bitnih za rješavanje u argumente se provjerava suprotni smjer rješavanja i zato nije potrebno posebno provjeravati gornje i lijevo susjedno polje.

```
if (adjacentField.GetAdjacentMines() - flagCounter == 1)
{

    Field adjacentField1 = board.GetField(field.GetRow(), field.GetColumn());
    Field adjacentField2 = board.GetField(field.GetRow() + 1, field.GetColumn());
    Field adjacentField3 = board.GetField(field.GetRow() + 2, field.GetColumn());
    Field safeField1 = board.GetField(field.GetRow(), field.GetColumn() + 3);
    Field safeField2 = board.GetField(field.GetRow() + 1, field.GetColumn() + 3);
    Field safeField3 = board.GetField(field.GetRow() + 2, field.GetColumn() + 3);

    fields.AddRange(SolvePattern11(field, adjacentField, adjacentField1, adjacentField2, adjacentField3, safeField1, safeField2, safeField3));
    fields.AddRange(SolvePattern11(field, adjacentField, safeField1, safeField2, safeField3, adjacentField1, adjacentField2, adjacentField3));
}
```

Slika 4.4.3.7. Priprema za treći korak u rješavanju prepoznavanjem uzorka 1-1 u kodnom obliku

U lokalnoj funkciji „*SolvePattern11*“ se obavlja treći i četvrti prethodno objašnjeni korak i vraća se lista polja koja su se otkrila. Na kraju metode „*LookForPattern11*“ se vraća lista polja „*fields*“ koja sadrži sva polja otkrivena tijekom izvođenja metode.

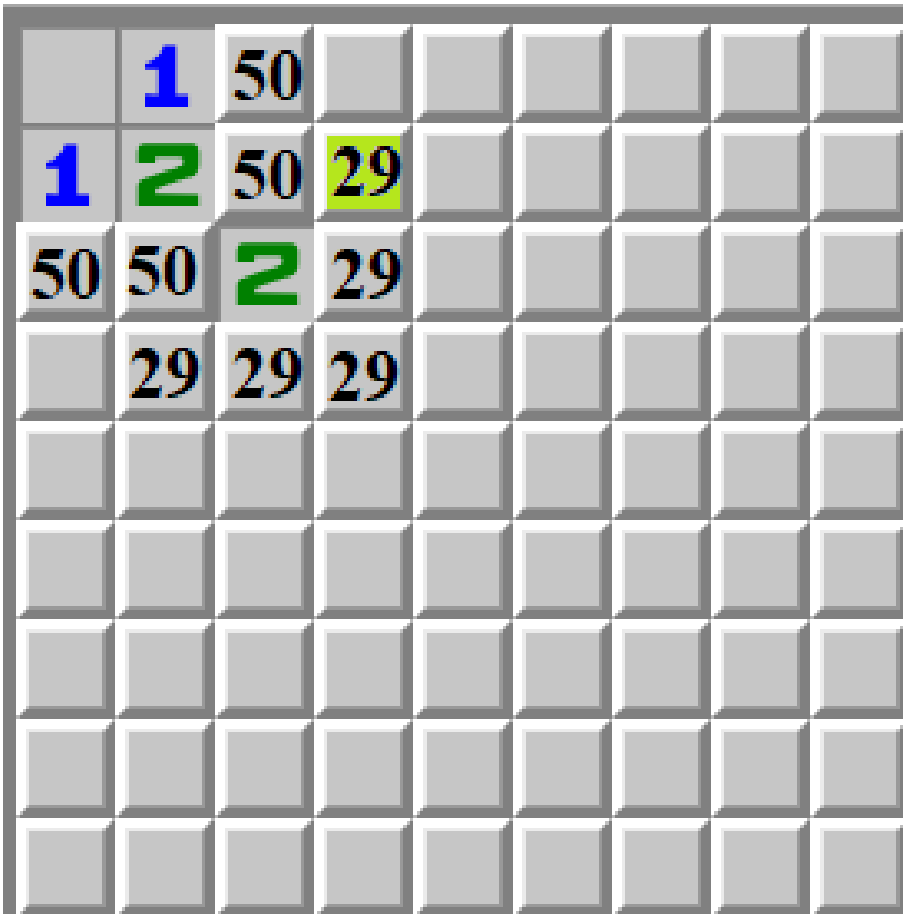
```
List<Field> SolvePattern11(Field field, Field adjacentField, Field adjacentField1, Field adjacentField2,
    Field adjacentField3, Field safeField1, Field safeField2, Field safeField3)
{
    List<Field> fields = new List<Field>();
    if ((board.IsInsideBoard(adjacentField1) == false || adjacentField1.IsRevealed() == true || adjacentField1.IsFlagged() == true) &&
        (board.IsInsideBoard(adjacentField2) == false || adjacentField2.IsRevealed() == true || adjacentField2.IsFlagged() == true) &&
        (board.IsInsideBoard(adjacentField3) == false || adjacentField3.IsRevealed() == true || adjacentField3.IsFlagged() == true))
    {
        List<Field> safeFields = new List<Field>();
        safeFields.Add(safeField1);
        safeFields.Add(safeField2);
        safeFields.Add(safeField3);
        bool printFlag = false;

        foreach (Field safeField in safeFields)
        {
            if (board.IsInsideBoard(safeField) == true && safeField.IsRevealed() == false && safeField.IsFlagged() == false)
            {
                safeField.Reveal();
                fields.Add(safeField);
                printFlag = true;
            }
        }
        if (printFlag == true)
        {
            board.PrintChange(field, adjacentField, "Changes made using the ''11 pattern'' with fields: ");
        }
    }
    return fields;
}
```

Slika 4.4.3.8. Lokalna funkcija „*SolvePattern11*“, treći i četvrti korak u rješavanju prepoznavanjem uzorka 1-1 u kodnom obliku

U algoritmu se nalazi šest različitih metoda za različita rješavanja prepoznavanjem uzoraka.

4.4.4. Otkrivanje polja pogađanjem

Kada tijekom rješavanja ploče ponestane logičnih poteza potrebno je pogađati koje polje otkriti sljedeće. U algoritmu se pogađanje obavlja na način da se provjerava svako nepoznato polje unutar ploče koje ima barem jedno otkriveno susjedno polje. Onda se za svako polje koje se provjerava izračunava vjerojatnost da se u tom polju nalazi mina na temelju svakog otkrivenog susjeda tog polja pojedinačno i u varijablu se sprema samo najveća dobivena vjerojatnost pojavljivanja mine od izračunatih vjerojatnosti. Ako je ta vjerojatnost manja od dotadašnje najniže spremljene vjerojatnosti, tada se vjerojatnost i polje za koje ona vrijedi spremaju kao trenutna najbolja opcija za odabir polja.



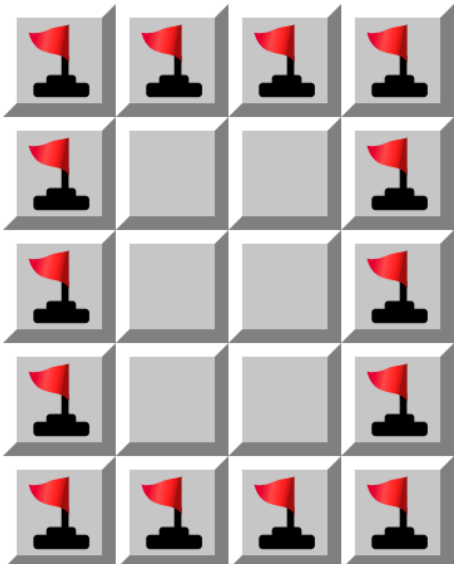
Slika 4.4.4.1. Odabir najbolje trenutne opcije za otkrivanje polja na temelju vjerojatnosti

Kada se provjere sva polja i vjerojatnost najbolje opcije iznosi 25% ili više, algoritam pokušava otvoriti polje koje se nalazi u kutu i sva su mu susjedna polja mu nepoznata zato da pokuša dobiti novo otvaranje jer na taj način algoritam ima veću stopu rješivosti. Vrijednost 25% je dobivena kao vrijednost koja je ostvarila najbolje rezultate tijekom isprobavanja raznih vrijednosti.

```
Revealing the first field:
ROW: 1, COLUMN: 1, ADJACENT MINES: 1

1 ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?

Calculated probability of activating a mine while guessing is too high, trying to get a new opening:
ROW: 1, COLUMN: 9, ADJACENT MINES: 0

1 ? ? ? ? ? ? 0
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
```

Slika 4.4.4.2. Uspješni pokušaj dobivanja novog otvaranja ploče

Ako se dogodi situacija da su sva preostala nepoznata polja od ostatka ploče odvojena minama se za otkrivanje odabire prvo pronađeno polje jer tada sva polja imaju istu vjerojatnost sadržavanja mine.



Slika 4.4.4.3. Dio ploče odvojen minama

Primjena otkrivanja polja pogađanjem se u algoritmu odvija u metodi „*TryGuessing*“.

```
List<Field> neighbours = board.GetNeighbours(field);
float mineProbability = -1;

foreach (Field neighbour in neighbours)
{
    if (neighbour.IsRevealed())
    {
        float currentMineProbability = -1;
        float flagCounter = 0;
        float unrevealedNeighbourCounter = 0;
        List<Field> neighbours2 = board.GetNeighbours(neighbour);
        foreach (Field neighbour2 in neighbours2)
        {
            if (neighbour2.IsFlagged() == true)
            {
                flagCounter++;
            }
            else if (board.IsInsideBoard(neighbour2) == true && neighbour2.IsRevealed() == false)
            {
                unrevealedNeighbourCounter++;
            }
        }
        if (unrevealedNeighbourCounter != 0)
        {
            currentMineProbability = (neighbour.GetAdjacentMines() - flagCounter) / unrevealedNeighbourCounter;
            if (currentMineProbability > mineProbability)
            {
                mineProbability = currentMineProbability;
            }
        }
    }
}
```

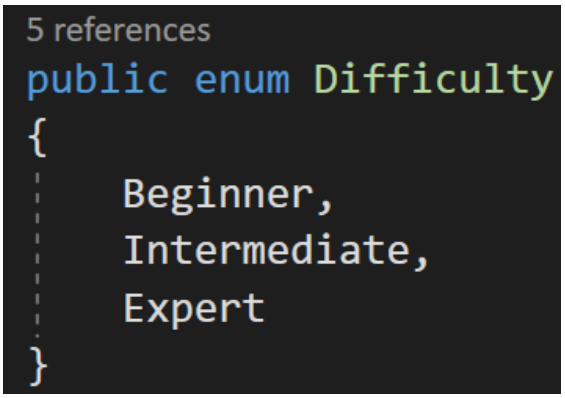
Slika 4.4.4.4. Dio metode „*TryGuessing*“ koji računa vjerojatnost otkrivanja mine na polju

4.5. Klasa Program

Klasa „Program*“* je klasa koja sadrži statičku metodu „*Main*“ u kojoj se odvija glavna procedura izvođenja programa. Ona ima uspostavljen odnos *has-a* s klasom „*Field“,* klasom „*Board“* i klasom „*BoardSolver*“ jer sadrži njihove objekte.

Statička metoda „*Main*“ započinje na način da se stvaraju varijable:

- *games* (*float* tip varijable, poprima vrijednost koja govori koliko će se puta izvesti simulacija rješavanja nasumičnih ploča minolovca),
- *wins* (*float* tip varijable, poprima vrijednost varijable „*games*“ te se smanjuje svaki put kada se neuspješno riješi ploča minolovca),
- *difficulty* (*Difficulty* tip varijable koji je prilagođeni tip podatka i predstavlja težinu ploče minolovca, poprima vrijednost odabrane težine ploče minolovca),
- *printFlag* (*bool* tip varijable, poprima vrijednost koja govori želi li se ispis koraka rješavanja u terminalu ili se samo prikupljaju statistički podaci).



Slika 4.5.1. Prilagođeni tip podatka „*Difficulty*“

Ukoliko je varijabla „*printFlag*“ postavljena na vrijednost *false*. tada se izvodi ispis stope pobjede na temelju broja simulacija postavljeno u varijabli „*games*“

```
if (printFlag == false)
{
    Console.WriteLine("Win rate = " + (wins / games * 100).ToString("N3") + "%");
}
```

Slika 4.5.2. Programski kod izvođenja ispisa stope pobjede

Simulacija rješavanja ploče minolovca odvija se unutar *for* petlje koja se obavlja onoliko puta koliko iznosi vrijednost varijable „*games*“. Na početku simulacije se stvara varijabla „*board*“ koja je objekt klase „*Board*“ i varijabla „*solver*“ koja je objekt klase „*BoardSolver*“. Onda se stvara lista polja „*fields*“ u koju se spremaju sva polja koja se koriste za, potencijalno, rješavanje ploče osnovnom logikom igre. Otkriva se prvo polje ploče za koje je sigurno da nije mina i sprema se u listu „*fields*“. Još se stvaraju neke varijable koje će se koristiti pri rješavanju ploče.

```
Board board = new Board(difficulty, printFlag);
BoardSolver solver = new BoardSolver(board);

board.GenerateBoard();
board.PrintBoard();

List<Field> fields = solver.FirstClick(board.GetField(1, 1));
List<Field> possibleFields = new List<Field>();

int flaggedCounter = 0;
int revealedCounter = 1;
```

Slika 4.5.3. Programski kod za prvi dio simulacije rješavanja polja minolovca

Nakon toga se ulazi u *do-while* petlju koja se ne zaustavlja dok se ne riješi ploča ili do pojave naredbe *break* koja se izvodi nakon otkrivanja polja s minom tijekom pogađanja.

U petlji se prvo provjerava je li došlo do situacije da su sva nepoznata polja odvojena minama od ostatka ploče i ako se to dogodi se poziva metoda „*TryGuessing*“ za otkrivanje polja pogađanjem. Ako se otvori polje u kojem se nalazi mina se smanjuje vrijednost varijable „*wins*“ za računanje stope pobjede i izvođenje simulacije se zaustavlja, a u protivnom se u listu „*fields*“ dodaje otkriveno polje.

```
if (fields.Count == 0)
{
    Field guess = solver.TryGuessing();
    if (guess.IsMine())
    {
        wins--;
        break;
    }
    else
    {
        fields.Add(guess);
    }
}
```

Slika 4.5.4. Programski kod za rješavanje situacije kada su sva nepoznata polja odvojena minama

Onda se provjerava je li došlo do situacije da se već prije pokušao postići napredak koristeći rješavanje osnovnom logiku igre s određenim poljem, a u međuvremenu nije došlo do nikakvog napretka u rješavanju. Ako se ta situacija dogodi algoritam poziva sve metode iz klase „*BoardSolver*“ koje služe za pronalaženje i rješavanje prepoznavanjem uzorka. Ukoliko se postigne napredak u rješavanju se u listu „*fields*“ dodaju polja koja se, potencijalno, mogu koristiti za rješavanje ploče osnovnom logikom igre. Ako nije došlo do napretka, u rješavanju se poziva metoda „*TryGuessing*“ za otkrivanje polja pogađanjem te se obavlja postupak objašnjen u situaciji kada se provjerava je li došlo do situacije da su sva nepoznata polja odvojena minama od ostatka ploče.

```
if (solver.WasAlreadyTried(fields[0]) == true)
{
    flaggedCounter = board.GetFlaggedCount();
    revealedCounter = board.GetRevealedCount();
    possibleFields = new List<Field>();
    possibleFields.AddRange(solver.LookForSquare0112());
    possibleFields.AddRange(solver.LookForPattern11());
    possibleFields.AddRange(solver.LookForPattern12());
    possibleFields.AddRange(solver.LookForPattern14());
    possibleFields.AddRange(solver.LookForCorner131());
    possibleFields.AddRange(solver.LookForCorner222());

    foreach (Field possibleField in possibleFields)
    {
        if (solver.RemoveOrMoveField(possibleField) != null)
        {
            fields.Add(possibleField);
        }
    }

    if (board.GetFlaggedCount() == flaggedCounter && revealedCounter == board.GetRevealedCount())
    {
        Field guess = solver.TryGuessing();
        if (guess.IsMine())
        {
            wins--;
            break;
        }
        else
        {
            fields.Add(guess);
        }
    }

    fields.Add(fields[0]);
    fields.RemoveAt(0);
}
```

Slika 4.5.5. Programski kod za rješavanje situacije kada nije moguć napredak u rješavanju osnovnom logikom igre

Nakon toga se obavlja pokušaj rješavanja ploče koristeći osnovnu logiku igre. Prvo se pokušavaju označavati polja zastavicom pozivom metode „*TryFlagging*“ pa se pokušava postići napredak u rješavanju ploče pozivom metode „*TrySolvingField*“ koristeći kao argument ono polje iz liste „*fields*“ koje se nalazi na početku liste. Na kraju iteracije petlje se to polje uklanja s početka liste „*fields*“ te se stavlja na kraj zajedno s poljima koja su pomoću njega otkrivena te se pomoću kojih, potencijalno, može postići napredak koristeći osnovnu logiku igre i pokušava se riješiti ploča prebrojavanjem nepoznatih polja i broja preostalih mina pozivom metode „*TryCompletingTheBoard*“.

```
possibleFields = new List<Field>();
solver.TryFlagging(fields[0]);

possibleFields.AddRange(solver.TrySolvingField(fields[0]));
foreach (Field possibleField in possibleFields)
{
    if (solver.RemoveOrMoveField(possibleField) != null)
    {
        fields.Add(possibleField);
    }
}

fields.RemoveAt(0);
solver.TryCompletingTheBoard();
```

Slika 4.5.6. Programski kod za rješavanje osnovnom logikom igre

Ako vrijednost „*printFlag*“ iznosi *true,* pokretanjem algoritma se u terminalu ispisuje generirana ploča i svaki korak rješavanja ploče te na koji način je taj korak obavljen.

```
Completing the board using the number of remaining mines:

2 3 F F F F 2 1 1 1 0 1 2 F F 3 F 2 1 0 2 F F F F 1 1 F F 2
F F 5 F 6 F 2 1 F 2 1 1 F 3 3 F 3 F 1 0 2 F 4 3 2 2 3 5 F 2
2 2 3 F 3 1 1 1 2 F 2 2 2 1 1 1 2 1 1 0 2 2 2 0 0 1 F F 3 2
0 0 2 2 2 0 1 1 3 2 3 F 2 1 0 0 0 0 1 1 2 F 1 0 0 1 2 3 F 1
1 1 2 F 1 0 1 F 3 F 4 3 F 1 0 1 1 1 1 F 2 1 1 0 0 1 1 2 2 2
2 F 3 1 1 0 1 1 4 F 5 F 3 1 0 1 F 2 2 3 2 1 0 0 1 2 F 1 1 F
2 F 2 1 1 1 0 1 3 F 5 F 3 0 1 2 2 2 F 3 F 1 0 0 1 F 2 1 1 1
1 1 1 1 F 3 2 2 F 3 4 F 3 2 4 F 3 2 2 F 2 1 0 0 1 1 1 1 1 1
1 1 1 2 4 F F 3 3 F 2 1 2 F F F F 1 1 1 1 0 0 0 0 0 0 1 F 1
1 F 2 3 F F 4 F 3 2 1 1 3 4 4 4 3 2 0 1 2 3 3 2 1 0 0 1 1 1
1 2 F 4 F 3 2 2 F 1 0 1 F F 1 2 F 2 0 2 F F F F 2 0 0 0 0 0
0 1 3 F 4 2 0 1 1 1 0 1 2 3 2 3 F 4 2 3 F 5 5 F 2 0 0 0 0 0
0 0 2 F F 1 0 0 0 1 1 2 1 2 F 2 2 F F 2 2 F 3 2 2 1 1 1 0 0
0 1 2 3 2 2 1 1 1 2 F 3 F 3 1 1 1 2 2 1 2 2 3 F 2 2 F 1 0 0
0 1 F 2 1 1 F 1 1 F 3 4 F 3 2 1 1 0 0 0 1 F 2 1 2 F 3 2 1 0
0 1 2 F 1 1 1 1 1 1 2 F 3 F 2 F 1 0 0 0 1 1 1 0 1 1 2 F 1 0
```

Slika 4.5.7. Ispis završnog koraka u rješavanju ploče na težini „stručnjak“

```
1 ? ? ? ? 1 1 1 1
? ? ? F ? ? 1 ? ?
2 F 4 3 3 1 1 1 1
1 1 2 F 1 0 0 0 0
0 0 1 1 1 0 0 0 0
1 1 2 1 1 0 0 0 0
1 F 2 F 1 0 0 0 0
1 1 2 1 1 0 0 1 1
0 0 0 0 0 0 0 1 F

Trying to guess a safe field with 33,333% chance of activating a mine:
ROW: 1, COLUMN: 2, ADJACENT MINES: 1

1 1 ? ? ? 1 1 1 1
? ? ? F ? ? 1 ? ?
2 F 4 3 3 1 1 1 1
1 1 2 F 1 0 0 0 0
0 0 1 1 1 0 0 0 0
1 1 2 1 1 0 0 0 0
1 F 2 F 1 0 0 0 0
1 1 2 1 1 0 0 1 1
0 0 0 0 0 0 0 1 F

Changes made using the ''11 pattern'' with fields:
ROW: 1, COLUMN: 1, ADJACENT MINES: 1
ROW: 1, COLUMN: 2, ADJACENT MINES: 1

1 1 3 ? ? 1 1 1 1
? ? 4 F ? ? 1 ? ?
2 F 4 3 3 1 1 1 1
1 1 2 F 1 0 0 0 0
0 0 1 1 1 0 0 0 0
1 1 2 1 1 0 0 0 0
1 F 2 F 1 0 0 0 0
1 1 2 1 1 0 0 1 1
0 0 0 0 0 0 0 1 F
```

Slika 4.5.8. Ispis nekoliko koraka u rješavanju ploče na težini „početnik"

```
Trying to guess a safe field with 20,000% chance of activating a mine:
ROW: 5, COLUMN: 9, ADJACENT MINES: 1

1 ? ? ? ? ? ? ? ? F 1 0 0 0 0 0
? ? ? ? ? ? ? ? ? 2 2 2 2 1 0 0
? ? ? ? ? ? ? ? ? 1 1 F F 2 1 1
? ? ? ? ? ? ? ? ? 1 2 5 ? 3 ? ?
? ? ? ? ? ? ? ? * ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
```

Slika 4.5.9. Ispis otkrivanja mine u rješavanju ploče na težini „srednji"

5. ZAKLJUČAK

Cilj ovog rada je bio izraditi algoritam za logičku igru minolovac u programskom jeziku *C#*. Uspješno je obavljena implementacija generiranja ploče za tri različite težine: „početnik“, „srednji“ i „stručnjak“ te simulacija rješavanja koja primjenom logike i prepoznavanjem određenih uzoraka koje imaju jedinstveno rješenje korak po korak rješava ploču igre uz ispis svakog koraka i način na koji je izveden. Osim toga, algoritam pruža opciju da umjesto ispisa svakog koraka simulacije računa stopu pobjede algoritma na postavljenoj težini u odabranom broju iteracija simulacije.

Postignute stope pobjede u sto tisuća simulacija iznose:

- na težini „početnik“: 88.686%,
- na težini „srednji“: 70.197%,
- na težini „stručnjak“: 22.059%

što je dobar rezultat, ali lošiji od rezultata koje postižu najbolji igrači.

Algoritam bi mogao postići veću stopu pobjede poboljšanjem logike odabira polja koje treba otvoriti kada ponestane logičnih rješenja i dodavanjem metoda prepoznavanja određenih uzoraka koje imaju jedinstveno rješenje, iako su vrlo rijetko pojavljuju.

Algoritam je javno dostupan na poveznici: https://gitlab.com/oliver.horvat/minesweeper-solver.

LITERATURA

[1] *Minesweeper Original* https://www.microsoft.com/en-us/p/minesweeper-original/9p7px702j5nh, [pristupljeno 15.6.2023.]

[2] *Minesweeper Online* (*Statistics*), https://minesweeper.online/statistics, [pristupljeno 15.6.2023.]

[3] *Coolmath Games* (*G. Bateson*: *The History Of Minesweeper*), https://www.coolmathgames.com/blog/the-history-of-minesweeper, [pristupljeno 15.6.2023.]

[4] *Minesweeper Research Papers*, https://minesweepergame.com/research-papers.php, [pristupljeno 6.7.2023.]

[5] *Minesweeper Math Papers*, https://minesweepergame.com/math-papers.php, [pristupljeno 6.7.2023.]

[6] *M. Holliday: Incremental Game Development in an Introductory Programming Course*, https://minesweepergame.com/research/incremental-game-development-in-an-introductory-programming-course-1994.pdf, [pristupljeno 6.7.2023.]

[7] *P. Crow: A Mathematical Introduction to the Game of Minesweeper*, https://minesweepergame.com/math/a-mathematical-introduction-to-the-game-of-minesweeper-1997.pdf, [pristupljeno 6.7.2023.]

[8] *A. Fowler, A. Young: Minesweeper: A Statistical and Computational Analysis*, https://minesweepergame.com/math/minesweeeper-a-statistical-and-computational-analysis-2004.pdf, [pristupljeno 6.7.2023.]

[9] *D.* Becerra*: Algorithmic Approaches to Playing Minesweeper*, https://minesweepergame.com/math/algorithmic-approaches-to-playing-minesweeper-2015.pdf, [pristupljeno 6.7.2023.]

[10] *Bai Li: Msolver*, https://github.com/luckytoilet/MSolver , [pristupljeno 6.7.2023.]

[11] *Microsoft Learn*, https://learn.microsoft.com, [pristupljeno 16.6.2023.]

[12] *Visual Studio*, https://visualstudio.microsoft.com, [pristupljeno 16.6.2023.]

[13] *GitLab*, https://about.gitlab.com, [pristupljeno 16.6.2023.]

[14] *Git*, https://git-scm.com, [pristupljeno 16.6.2023.]

[15] *TestProject* (*A. Draniceanu*: *10 Common Git Commands Everyone Should Know*), https://blog.testproject.io/2021/03/22/git-commands-every-sdet-should-know, [pristupljeno 16.6.2023.]

[16] Materijali s kolegija Objektno orijentirano programiranje (izv. prof. dr. sc. D. Blažević), [pristupljeno 16.6.2023.]

[17] *CodeGuru* (*N. Rini*: *C# versus C*), https://www.codeguru.com/csharp/c-sharp-versus-c, [pristupljeno 16.6.2023.]

[18] *GeeksforGeeks*, https://www.geeksforgeeks.org, [pristupljeno 16.6.2023.]

[19] *Minesweeper Online* (*Patterns*), https://minesweeper.online/help/patterns, [pristupljeno 18.6.2023.]

SAŽETAK

Minolovac je poznata računalna igra koja se rješava primjenom logičkog razmišljanja u svrhu donošenja odluke nalazi li se u polju mina ili ne. Cilj igre je otkriti ona polja koja ne sadrže minu u ploči koja sadrži redove i stupce polja koja mogu sadržavati minu. Jedino što se koristi pri rješavanju ploče jest vrijednost koja je zapisana u otkrivenim poljima, koja predstavlja koliko polja koja ga okružuju sadrže minu. Napisan je algoritam u programskom jeziku *C#* koji generira ploče igre te ih rješava primjenom logike i pogađanja. Algoritam može prikazivati svaki korak rješavanja ploče ili računati ostvarenu stopu pobjede u određenom broju simulacija igre.

Ključne riječi: algoritam, C#, logika, minolovac

ABSTRACT

Algorithm for logic puzzle minesweeper in programming language C#

Minesweeper is a famous computer game that is solved using logical reasoning to determine whether a field contains a mine or not. The goal of the game is finding fields that do not contain a mine inside a board which contains rows and columns of fields that can contain a mine. The only thing that is used in solving the board is a value that is written in the revealed fields, which represents how many fields that surround it contain a mine. An algorithm has been written in the C# programming language, which generates game boards and solves them using logic and guessing. The algorithm can show every step of solving the board or calculate the win rate in a certain number of game simulations.

Key words: algorithm, C#, logic, minesweeper