Математичка гимназија
МГ

Алгоритам *Minimax* кроз игру *Connect 4*

Матурски рад

из предмета Програмирање и програмски језици

Ученик:
Ђурђа Јаковљевић

Ментор:
Снежана Јелић

Београд, 2026.

Садржај

Глава 1

Увод

1.1. Вештачка интелигенција у видео играма

Вештачка интелигенција (енгл. Artificial Intelligence - AI) је способност рачунарских система да функционишу у непознатим ситуацијама и у неструктуисаном окружењу [1]. Захваљујући вештачкој интелигенцији рачунари могу да обављају задатке који се типично повезују са људском интелигенцијом, као што су учење, расуђивање, решавање проблема, перцепција, креативност и доношење одлука. Она све више мења индустрију видео игара и начин на који се игре праве и играју. Захваљујући AI, ликови којима не управља играч (енгл. Non-Playable Characters - NPC) понашају се паметније, а могу се правити и читави измишљени светови код којих се поједини елементи могу и сами генерисати. То омогућава да игре буду занимљивије, динамичније и реалистичније него раније.

Значајан број видео игара укључује два противника, од којих је један човек, а други рачунар. Приликом креирања ових видео игара јавља се проблем избора оптималног потеза од стране рачунара. Током решавања наведеног проблема потребно је размотрити више могућих будућих исхода. У ове сврхе често се користи *Minimax* алгоритам који омогућава симулацију потеза унапред и избор најповољније стратегије.

У овом раду објашњени су основни концепти *Minimax* алгоритма, а сам алгоритам је имплементиран у оквиру игрице *Connect 4* која је развијена у раду. Рад је структуриран у 5 поглавља. Након првог, уводног поглавља у другом поглављу су представљене теоријске основе и принцип рада *Minimax* алгоритма. Треће поглавље односи се на алфа-бета оптимизацију која је у оквиру развијене *Connect 4* игрице употребљена за претрагу стабла игре *Minimax* алгоритма. У четвртом поглављу описан је развијени и имплементиран код за реализацију *Connect 4* игрице, док су у петом поглављу дате закључне напомене.

Глава 2

Minimax алгоритам

2.1. Шта је *Minimax*

Minimax [2, 3] алгоритам је алгоритам претраге који спада у класу алгоритама вештачке интелигенције и примењује се у играма са два играча (MAX и MIN). Један играч има циљ да максимизује свој резултат, а други да га минимизује. У играма где човек игра против компјутера, компјутер функционише под претпоставком да ће човек одиграти оптимално, то јест да ће увек бирати потез који највише штети противнику.

2.2. Основни концепти *Minimax* алгоритма

У оквиру овог одељка биће наведени основни елементи *Minimax* алгоритма (стање и стабло игре) и објашњен начин на који се врши претрага могућих стања. Приликом претраге компјутер, тј. систем вештачке интелигенције (енгл. *Artificial Intelligence* - AI) има улогу MAX, а човек MIN играча.

2.2.1. Стање игре

Стање је положај игре у датом тренутку. Терминално стање, које се још назива и лист, је стање у ком игра има исход (победа, пораз, нерешено). У овом алгоритму стања добијају вредности, у складу са тим колико су повољна.

Стање игре представља комплетан опис положаја игре у одређеном тренутку. Оно садржи све информације потребне да би се одредили могући наредни потези и наставак игре. Типично, стање игре укључује [2, 3]:

- тренутну конфигурацију свих елемената игре,
- информацију о томе који је играч на потезу,

- скуп свих потеза који се могу извршити из тог стања.

Прелазак из једног стања у друго настаје применом неког дозвољеног потеза. Клонирање тренутног стања омогућава генерисање нових стања без мењања оригиналног чиме се задржава независност чворова у стаблу игре.

Посебна врста стања је терминално стање у којем је исход игре већ одређен (победа, пораз или нерешено). У тим случајевима алгоритам зауставља даљу претрагу и додељује одговарајућу вредност стању, што је основ за процену користи потенцијалних потеза у *Minimax* алгоритму.

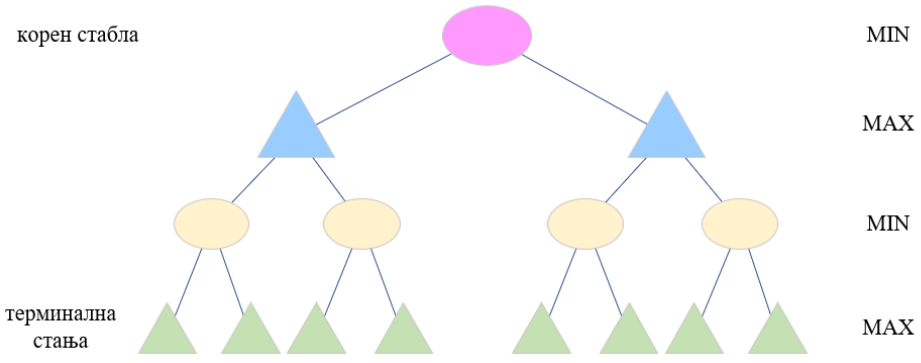
2.2.2. Стабло игре

Алгоритам врши анализу свих могућих потеза из тренутног стања и формира стабло игре. Стабло игре је графичка структура која представља сва могућа стања игре од тренутног па све до терминалног [2, 3]. Састоји се из чворова и грана. Чворови су појединачна стања игре, а гране су потези који воде од једног стања до другог. Тренутно стање игре назива се кореном, док се крајња (терминална) стања називају листовима.

Стабло је организовано у нивое, где сваки ниво одговара једном потезу. На пример:

- ниво 0 → корен (тренутно стање)
- ниво 1 → могући потези MAX играча
- ниво 2 → могући потези MIN играча

У стаблу се кроз нивое смењују потези MAX играча и MIN играча и омогућава се *Minimax* алгоритму да симулира различите сценарије игре.



Слика 2.1. Пример стабла игре

Чворови стабла носе вредности које се користе за одлучивање:

- Крајњи листови добијају фиксну вредност у складу са исходом игре (нпр. максимална вредност ако AI победи, минимална ако изгуби, 0 за нерешено).

- Унутрашњи чворови добијају вредност на основу *Minimax* правила: Уколико се чвор односи на потез MAX играча, тј. одговара нивоу у стаблу који описује потез MAX играча (у примеру на слици 2.1 означено троугловима), онда чвор добија максимум вредности својих директних потомака. У супротном, ако се чвор односи на MIN играча, он добија минималну вредност својих директних потомака.

Прелаз из једног стања у друго настаје кроз гране стабла где свака грана представља један могући потез. На пример, ако играч изврши одређени потез, од корена се формира грана која води ка новом чвору, што омогућава алгоритму да настави претрагу из тог новог стања.

2.2.3. Вредновање (евалуација) стања и дубина

За сложене игре, као што су шах или *Connect 4*, стабло игре може бити изузетно велико. Због тога се често користи ограничена дубина претраге и функција вредновања која додељује вредност чворовима који нису даље разгранати. На овај начин омогућава се алгоритму да донесе одлуку без потребе да истражи сва могућа терминална стања.

Дубина стабла представља број нивоа претраге које *Minimax* алгоритам истражује од корена (тренутног стања) ка листовима стабла [2, 3]. Користи се као ограничење претраге јер у комплексним играма стабло експоненцијално расте. Алгоритам не прегледа сва терминална стања, већ само она до нивоа који одговара изабраној дубини, а стањима која још нису довела до победе, пораза или нерешеног исхода додељује се вредност помоћу функције вредновања.

Функција вредновања (евалуације) је механизам који алгоритам користи да одреди колико је одређено стање игре повољно за једног од играча. Она је посебно важна када алгоритам не прегледа сва терминална стања, већ се користи ограничена дубина претраге.

Сваки чвор који није достигао терминално стање (победа, пораз или нерешено) добија вредност на основу функције вредновања. Вредност одређује „квалитет“ тог стања за AI или човека, омогућавајући алгоритму да на основу њих бира оптималан потез. Ово алгоритму даје могућност да „предвиди“ најбоље потезе чак и ако не може да истражи сва могућа терминална стања, чиме се значајно смањује време израчунавања уз минималне негативне последице по квалитет одлуке.

2.3. Принцип рада

Minimax алгоритам доноси одлуке кроз рекурзивну симулацију будућих потеза, користећи све претходно описане концепте – стање игре, стабло, вредновање и дубину претраге. Основна идеја је да AI „размишља унапред“ како би предвидео најбоље могуће исходе и изабрао оптималан потез [3].

Принцип рада може се приказати у неколико корака:

1. Идентификација тренутног стања

Алгоритам узима тренутну таблу као корен стабла игре. Свака могућа одлука (потез) за AI или човека представља гране стабла. Формира се стабло игре.

2. Рекурзивна симулација потеза
   Алгоритам иде кроз све валидне потезе до одређене дубине или до терминалних стања. На сваком новом чвору формира се ново стање игре које одражава потенцијални потез играча.

3. Вредновање стања
   Када алгоритам достигне лист или ограничену дубину, користи функцију процене да додели вредност стању.

4. Рекурзивно доношење одлуке
   Вредности се враћају уздужно ка корену стабла:

   - Уколико је на потезу MAX играч (AI) бира се потез са највећом вредношћу међу директним потомцима.
   - Уколико је на потезу MIN играч (човек) симулира се противников најбољи потез тако што се бира најмања вредност међу директним потомцима.

На овај начин AI узима у обзир не само сопствене могућности, већ и могуће одговоре противника.

Глава 3

Алфа-бета оптимизација

Алфа-бета оптимизација је техника која се користи у алгоритмима за претрагу стабла игре, као што је *Minimax* алгоритам, како би се смањио број позиција које алгоритам мора да испита, а да при томе не изгуби на тачности одлуке [2, 4]. Основна идеја је да алгоритам прескаче оне гране игре које не могу променити коначну одлуку о томе који је најбољи потез.

У оквиру ове технике уводе се две вредности:

- Алфа (α) представља најбољу вредност коју максимизујући играч (обично AI) може тренутно да обезбеди. Иницијална вредност му је -∞.
- Бета (β) представља најбољу вредност коју минимизујући играч (обично човек) може да наметне противнику. Иницијална вредност му је $+\infty$.

Током претраге, ако се догоди да је $\beta \leq \alpha$, то значи да даље гране у том делу стабла не могу донети бољи исход за максимизујућег играча. У том случају алгоритам прескаче ту грану и наставља анализу других опција. Ово прескакање, познато као алфа-бета оптимизација, значајно убрзава рад алгоритма јер елиминише проверу непотребних потеза.

Претрага стабла игре обавља се рекурзивно од листова ка корену. Алгоритам обрађује једног по једног потомка и не формира вредност родитеља све док се не обраде сви његови потомци. На тај начин се осигурава правилан редослед корака: сваки MAX или MIN чвор израчунава своју коначну вредност тек након што се изврши обрада све његове деце, а затим се резултат прослеђује навише ка родитељу и коначно ка корену стабла.

Процес евалуације се разликује у зависности од тога да ли се евалуира вредност чвора који припада MAX или MIN нивоу. Уколико чвор припада MAX нивоу, вредност α се одређује на основу вредности директних потомака ($value$) на следећи начин:

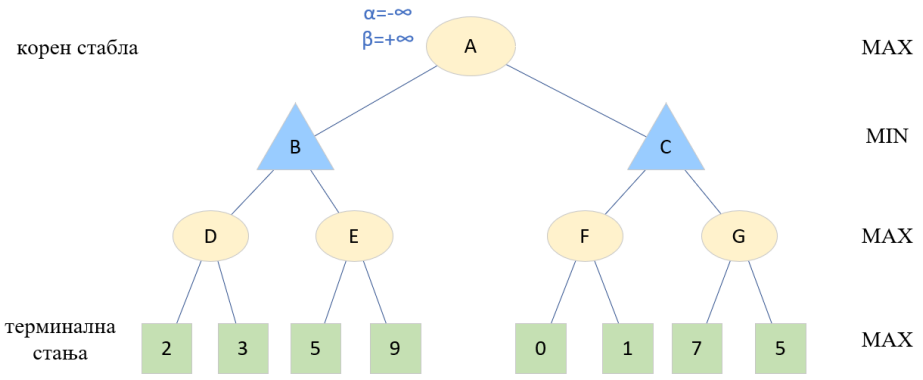
$$\alpha = max(alpha, value)$$

Ако се деси да је за неки од директних потомака $\alpha \geq \beta$ престаје се са провером вредности преосталих потомака. При том, директни потомци на почетку евалуације наслеђују вредности α и β од својих родитеља. Уколико чвор припада min нивоу, вредност β се одређује на основу вредности директних потомака ($value$) на следећи начин:

$$\beta = min(\beta, value)$$

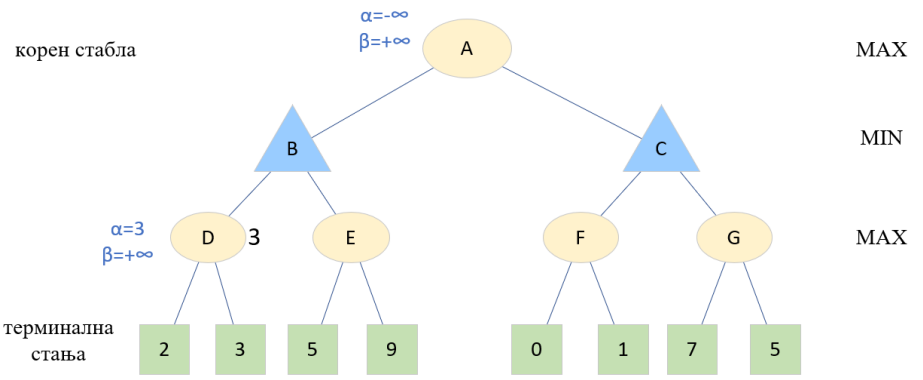
Уколико се деси да је $\beta \leq \alpha$ престаје се са евалуацијом преосталих потомака.

Ради бољег разумевања алфа-бета оптимизације, увешћемо пример једног стабла игре и објаснити на њему зашто се неке гране пресецају. У нашем примеру се стабло проверава слева надесно.



Слика 3.1. Стабло игре [4]

На слици 3.1 приказано је стабло игре пре било какве претраге и доношења одлука. У корену стабла, вредности алфа и бета су постављене на -∞ и +∞. Крећемо претрагу од чвора D, односно првог чвора у последњем реду стабла, пре терминалних вредности.



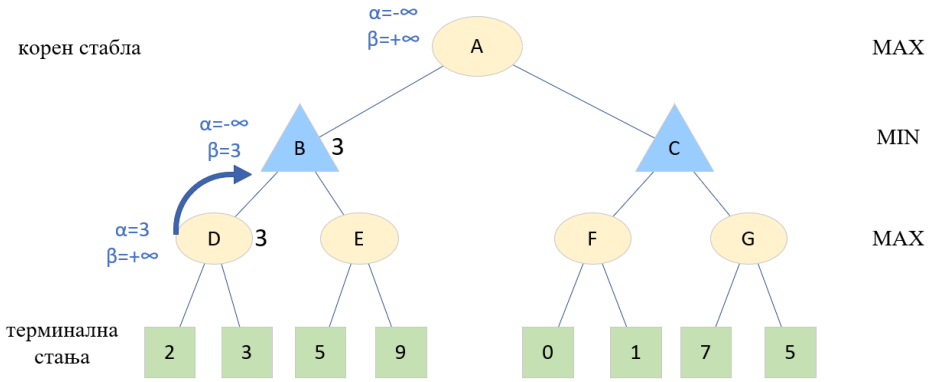
Слика 3.2. Обрада чвора D [4]

Чвор D је чвор максималног реда у стаблу игре са два потомка са терминалним вредностима 2 и 3. Алгоритам корак по корак израчунава вредност D:

- Почетна α вредност за D је $\alpha = -\infty$ (још није обрађено ниједно дете).
- Прво се обрађује прво дете (2), α се ажурира на $\max(-\infty,2) = 2$.
- Затим се обрађује друго дете (3), α се ажурира на $\max(2,3) = 3$.

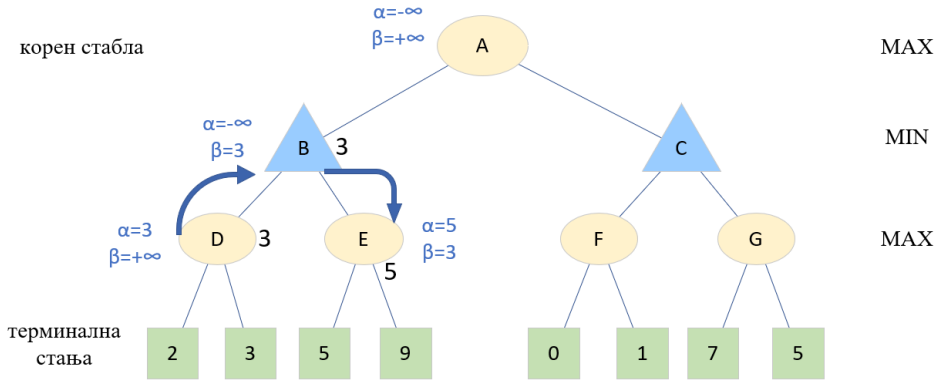
Коначна вредност MAX чвора D је $\alpha = 3$ (слика 3.2), што је максимална вредност међу његовим потомцима. Бета је $+\infty$, јер D још није ограничен ни једним MIN чвором који би смањио β.

Чвор D нема више потомака и самим тим његова коначна вредност је формирана. Прелазимо на обраду његовог родитеља, а то је чвор B.



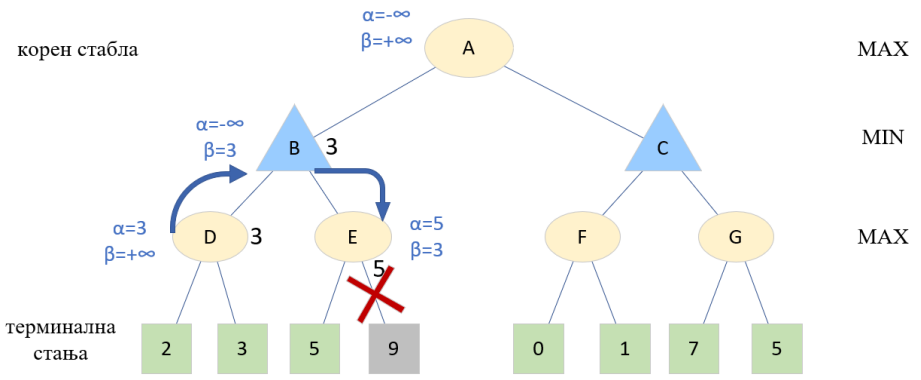
Слика 3.3. Обрада чвора B [4]

Чвор B као минимизирајући чвор за себе бира минималну вредност од вредности својих потомака. Чвор E још није обрађен, па је у оптицају за вредност само чвор D. За β чвор B узима вредност чвора D, тј. 3 (слика 3.3), јер је то тренутно минимални резултат који B може да обезбеди. Алфа је -∞, пошто B нема максимизирајући родитељски чвор који би му наметнуо другачију вредност.



Слика 3.4. Обрада чвора E [4]

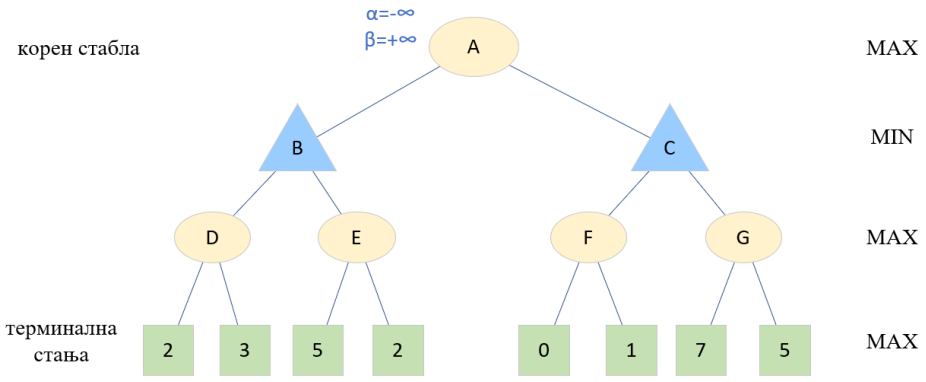
Даље прелазимо на обраду јединог преосталог потомка чвора B, што је чвор E. Чвору E β диктира његов родитељ, минимизирајући чвор изнад њега, па је $\beta = 3$. Пошто обрада иде слева надесно, прво за E узимамо вредност 5. То је тренутно највећа вредност коју чвор E може да обезбеди, па је $\alpha = 5$ (слика 3.4). Пошто је $\beta \leq \alpha$, врши се алфа-бета оптимизација и пресеца се преостали потомак чвора B, чвор са терминалним стањем 9, пошто његова вредност неће утицати на крајњи исход (слика 3.5).



Слика 3.5. Пресецање потомка [4]

Овде се поставља питање зашто се чвор са терминалним стањем вредности 9 пресеца и зашто он не утиче на крајњи исход. Ако бисмо кренули у његову обраду, чвор E би узео његову вредност, пошто је $\max(5,9) = 9$. Међутим ово не би имало утицаја на вредност чвора B који тренутно има вредност 3 (од свог потомка D) и та вредност ће остати непромењена јер је $\min(3,9) = 3$.

Случај кад би била другачија ситуација у стаблу игре, ако би овај чвор уместо вредности веће од 5 имао вредност мању од 5, на пример 2, стабло би изгледало као што је приказано на слици 3.6.



Слика 3.6. Промењено стабло игре [4]

При уласку у његову обраду, чвор E би имао да донесе одлуку коју вредност да узме за себе 2 или 5. Пошто је $\max(2,5) = 5$, вредност овог чвора би одмах била одбачена, и он не

би имао никакву улогу даље у претрази.

Главни разлог због којег користимо алфа-бета оптимизацију је ефикасност. Без ње, *Minimax* алгоритам морао би да испита сваки могући низ потеза до краја стабла, што је у игри као што је *Connect 4* непрактично због великог броја могућих комбинација. Алфа-бета оптимизација омогућава алгоритму да истовремено:

- задржи тачност и стратешку оптималност одлуке,
- обради дубље нивое игре за исто време,
- донесе боље и брже одлуке у реалном времену.

Глава 4

Minimax u *Connect 4*

4.1. *Connect 4*

Connect 4 (споји четири) је игра у којој играчи бирају једну од две боје - црвена и жута, и наизменично убацују токене своје боје у таблу димензија 6×7. Токени падају у први слободан ред одабране колоне. Циљ игре је да се хоризонтално, вертикално или дијагонално формира линија од 4 токена у изабраној боји. Игру је 1974. године креирао Хауард Весклер. Исте године продата је прва верзија под називом "*ConnectFour*" од стране компаније Милтон Бредли. Данас је њен произвођач једна од највећих светских компанија играчака и друштвених игара Хасбро (слика 4.1).

Слика 4.1. Connect 4 игра компаније Хасбро [5]

4.2. Имплементација *Connect 4*

У наставку рада, *Minimax* алгоритам биће објашњен на примеру игре *Connect 4*, имплементиране у програмском језику C# [6]. Најпре ће бити објашњена структура података и

логика игре, а потом и реализација *Minimax* алгоритма.

4.2.1. Класа Tabla

Табла је представљена као матрица са константним бројем редова и колона (6 и 7) - у класи атрибути `redovi` и `kolone`. Поља мреже (матрице) узимају једну од 3 вредности: 0, 1 или -1. Вредност 0 означава празно поље, 1 попуњено поље човековим токеном и -1 попуњено компјутеровим.

```
public bool kolona_puna(int kolona) {
    return mreza[0, kolona] != 0;
}
public bool puna() { //da li je tabla cela puna
    for (int i = 0; i < redovi; i++)
        for (int j = 0; j < kolone; j++)
            if (mreza[i, j] == 0)
                return false;
    return true;
}
```

Слика 4.2. Методе за проверу да ли су колоне или цела табла пуна

Методу `kolona_puna` (слика 4.2) користимо за проверу да ли играч може убацити свој токен у дату колону, тј. да ли постоји слободно поље у тој колони. С друге стране, метода `puna` (слика 4.2) нам говори о томе да ли уопште постоји неко празно место у табли, тј. да ли се игра може наставити или се прекида.

```
public Tabla Klon() {
    Tabla nova_tabla = new Tabla();
    nova_tabla.Mreza = new int[mreza.GetLength(0), mreza.GetLength(1)]; // nova matrica
    for (int r = 0; r < mreza.GetLength(0); r++)
       for (int k = 0; k < mreza.GetLength(1); k++)
          nova_tabla.Mreza[r, k] = mreza[r, k]; // kopiraj svaki element, jedan po jedan
    return nova_tabla;
```

Слика 4.3. Метода за клонирање стања

Метода `Klon` је кључна за Minimax алгоритам. Као што смо већ рекли, AI игра тако што симулира потезе и прави стабло игре. Он то мора да ради без мењања стварног стања игре. Како би се овај проблем решио користи се метода `Klon` класе `Tabla` (слика 4.3) која прави копију тренутног стања игре, тј. копију тренутног стања табле. Ово омогућава алгоритму да за сваки потенцијални потез направи независну грану стабла игре. На тај начин *Minimax* може рекурзивно да анализира све сценарије и процени оптималан потез, а да при том стварна игра остане нетакнута. Без ове функције свака симулација би мењала праву таблу, што би довело до погрешних резултата и непредвидивог понашања програма.

```
public void igraj(int kolona, int igrac) {
    if(!kolona_puna(kolona))
    for (int red = redovi - 1; red >= 0; red--) {//krecemo od dna kolone
        if (mreza[red, kolona] == 0)
        {
            mreza[red, kolona] = igrac;
            return;
        }
    }
}
```

Слика 4.4. Метода `igraj`

Кад један од играча одигра свој потез, позива се метода *igraj* (слика 4.4). Прво слободно поље у колони коју је играч изабрао попуњава се његовим индексом (1 или -1).

4.2.2. Класа Stanje

```
class Stanje
{
    Tabla tabla;
    int trenutni_id;
    public Stanje(Tabla tabla, int id) {
        this.tabla = tabla;
        trenutni_id=id;
    }
    public void promeni_igraca() {
        trenutni_id = trenutni_id * (-1);
    }
    public Stanje Klon() {
        Tabla nova_tabla = tabla.Klon();
        Stanje klonirano = new Stanje(nova_tabla, trenutni_id);
        return klonirano;
    }
```

Слика 4.5. Класа `Stanje`

Класа `Stanje` (слика 4.5) за атрибуте има таблу и идентификациони број тренутног играча. Она садржи методу за мењање идентификационог броја тренутног играча. Ова метода се позива када један играч заврши са својим потезом и служи да се зна који је тренутно на потезу. Метода `Klon` се налази и у класи `Stanje`, чиме се прави клонирано стање, чији су атрибути клонирана табла и тренутни играч. Ова метода је заправо она која се користи у Minimax алгоритму.

4.2.3. Класа PravilaIgre

Класа `PravilaIgre` имплементира основну логику игре *Connect 4*. Основна улога класе је да омогући правилну и сигурну симулацију игре. Атрибути ове класе су број редова и

колона (имају константне вредности 6 и 7) и вредност дубине стабла - `redovi`, `kolone` и `dubina`.

```
public int proveri_pobedu(Stanje stanje) {
    int[,] mreza=stanje.Tabla.Mreza;
    for (int i = 0; i < redovi; i++)//horizontalno
        for (int j = 0; j < kolone - 3; j++)
            if (mreza[i,j]!=0&&mreza[i, j] == mreza[i, j + 1] &&
                mreza[i, j] == mreza[i, j + 2] && mreza[i, j] == mreza[i, j + 3])
                return mreza[i, j];
```

Слика 4.6. Метода за проверу победе

Ова класа садржи методу `proveri_pobedu` (слика 4.6) која служи за проверу да ли се на табли налазе 4 токена исте боје један поред другог, у хоризонталном, вертикалном или дијагоналним положајима. На слици је дат део кода који се односи на проверу хоризонталног положаја, а аналогно се проверава и за остале положаје. Метода враћа 0 ако нико није победио или индекс победника.

```
private int ProceniSegment(int ai_brojac, int covek_brojac)//segmenti od 4, njihova ocena
{
    double faktor = (double)dubina / 10; // veca dubina = veci uticaj
    int vrednost = 0;

    if (ai_brojac > 0 && covek_brojac > 0) return 0; // blokirani segment
    //brojac je koliko tokena odredjenog igraca ima u segmentu
    if (ai_brojac == 4) vrednost = (int)(10000 * faktor);
    else if (ai_brojac == 3) vrednost = (int)(100 * faktor);
    else if (ai_brojac == 2) vrednost = (int)(10 * faktor);

    if (covek_brojac == 4) vrednost = -(int)(10000 * faktor);
    else if (covek_brojac == 3) vrednost = -(int)(100 * faktor);
    else if (covek_brojac == 2) vrednost = -(int)(10* faktor);

    return vrednost;
}
```

Слика 4.7. Метода за процену сегмента

Метода `ProceniSegment` нам је потребна за функцију евалуације. Она оцењује сегменте од 4 узастопна поља на табли, хоризонтално, вертикално или дијагонално. Од параметара прима бројаче за човека и компјутер (`ai_brojac` и `covek_brojac`), тј. колико токена које боје се налази у изабраном сегменту (бројачи у збиру могу максимално дати 4).

У овој методи улогу има вредност `faktor`, која зависи од дубине претраге. Што је већа дубина, то је већи фактор и биће већа оцена сегмента. Већа дубина значи да AI види даље у будућност игре. У *Minimax* алгоритму вредност позиције се одређује на основу анализе будућих могућих потеза, а не само тренутног стања игре. Иако неки потез може деловати повољно на почетном нивоу стабла игре, његова стварна вредност зависи од исхода до којих доводи на већим дубинама. Позиције на мањој дубини представљају само међукораке

који се могу променити у наредним потезима, док позиције на већој дубини дају поузданију процену јер су ближе коначном исходу игре.

Уколико се у неком сегменту од четири поља налазе токени и једног и другог играча, њега оцењујемо са 0. Он није повољан ни за човека, ни за компјутер, јер у њему ни један ни други не могу остварити победу. Овај сегмент називамо блокирани сегмент. У супротном, ако се у сегменту од четири поља не налазе токени и једног и другог играча, овај сегмент за AI оцењујемо тако што фактор помножимо са једном од следећих вредности:

- 2 у низу → 10
- 3 у низу → 100
- 4 у низу → 10000

Приликом оцењивања вредности за вредности човека испред вредности се ставља знак минус.

```
public int Evaluacija(Stanje s) { //procena koliko je neko stanje povoljno za ai
    int pobednik = proveri_pobedu(s);
    if (pobednik == 1) return -1000000; //covek pobedio
    if (pobednik == -1) return 1000000; //ai pobedio
    if (s.Tabla.puna()) return 0;
    int vrednost = 0;
    //bonus za centar
    int[,] mreza = s.Tabla.Mreza;
    int centar = kolone / 2;
    int centar_brojac = 0;//koliko tokena ai ima u centru
    for (int r = 0; r < redovi; r++)
    {
        if (mreza[r, centar] == -1) centar_brojac++; // ai
    }
    vrednost += centar_brojac * 3;
    //horizontalno
    for (int i = 0; i < redovi; i++)
        for (int j = 0; j < kolone - 3; j++)
        {
            int ai_count = 0;
            int covek_count = 0;
            for (int k = 0; k < 4; k++)
            {
                if (mreza[i, j + k] == -1) ai_count++;
                else if (mreza[i, j + k] == 1) covek_count++;
            }
            vrednost += ProceniSegment(ai_count, covek_count);
        }
```

Слика 4.8. Метода евалуације

Метода `Evaluacija` (слика 4.8) служи за оцењивање неког стања. Она у самом старту проверава да ли у датом стању имамо победника. Уколико га имамо, стање оцењује много

високом вредношћу у случају победе AI (MAX играча) или са много ниском вредношћу у случају победе човека (MIN играча). Уколико нема победника и табла је пуна, стање оцењује са 0, пошто није повољно ни за MAX, ни за MIN играча.

У игри *Connect 4* централна колона има већу стратешку вредност у односу на друге колоне, јер омогућује више потенцијалних победничких комбинација. Позиције у центру табле могу бити део више хоризонталних, вертикалних и дијагоналних линија, што повећава вероватноћу формирања низа од четири поља. Самим тим, и у функцији евалуације централна колона се вреднује више.

Редове, колоне и дијагонале оцењујемо преко функције `ProceniSegment` тако што на укупну вредност додамо вредност сваког сегмента појединачно.

4.2.4. Класа AI

```
private int Minimax(Stanje stanje, int depth, bool max_igrac, int alfa, int beta) {
    PravilaIgre pravila = new PravilaIgre(dubina);
    if (pravila.proveri_pobedu(stanje)!=0) {
        int pobednik = pravila.proveri_pobedu(stanje);
        if (pobednik == -1) return 1000000;  // AI
        if (pobednik == 1) return -1000000; // covek
    }
    if (depth == 0 || stanje.Tabla.puna()) //ako je dosegao max dubinu pretrage ili je tabla puna
        return pravila.Evaluacija(stanje);
    if (max_igrac) {
        int maxEval = int.MinValue;
        foreach (int kolona in stanje.Tabla.validne_pozicije())
    {
        Stanje novoStanje = stanje.Klon();
        novoStanje.Tabla.igraj(kolona, -1);
        novoStanje.promeni_igraca();

        int eval = Minimax(novoStanje, depth - 1, false, alfa, beta);
        maxEval = Math.Max(maxEval, eval);
        alfa = Math.Max(alfa, eval);
        if (beta <= alfa) break; // alfa-beta pruning
    }
    return maxEval;
}
```

Слика 4.9. Метода `Minimax`

Метода `Minimax` враћа завршну процену квалитета датог стања, користећи раније поменуте класе и њихове методе, као и рекурзију. Она у старту проверава да ли је један од играча победио. Уколико јесте, враћа се вредност. Уколико није, иде се даље са проверама. Ако је програм досегао максималну дубину претраге (стигао је до унапред задатог ограничења колико потеза унапред гледа) или је у својој претрази дошао до стања у ком је табла пуна, враћа вредност тог стања (методом `Evaluacija` класе `PravilaIgre`). У случају да ништа од овога није достигао, иде се даље на имплементацију *Minimax* алгоритма.

У делу кода који имплементира *Minimax* алгоритам (слика 4.9) прво се проверава да ли је на потезу AI играч (`max_igrac == true`) или човек. Када је на потезу AI играч, метода иницијализује промењљиву `maxEval` на минималну могућу вредност и пролази кроз све валидне колоне на табли. За сваку колону прави се клон тренутног стања (`Stanje novoStanje`) и симулира се потез AI играча помоћу `novoStanje.Tabla.igraj(kolona, -1)`. Након тога се мења активни играч (`novoStanje.promeni_igraca()`) и позива се рекурзивно функција `Minimax` са смањеном дубином. Добијена вредност се упоређује са тренутним максимумом (`maxEval`), а промењљива `alfa` се ажурира за потребе алфа-бета одсецања. Ако важи услов `beta <= alfa`, петља се прекида јер даље гране не могу донети бољи исход. На крају се враћа `maxEval`, односно најбоља предвиђена вредност позиције за AI играча.

С друге стране, када је на потезу човек (`else`), алгоритам симулира његову оптималну игру, готово исто као у случају AI. Иницијализује се `minEval` на максималну могућу вредност и пролази кроз све валидне колоне. За сваку колону прави се клон стања, симулира потез човека (`novoStanje.Tabla.igraj(kolona, 1)`) и мења активни играч. Рекурзивно се позива `Minimax`, добијена вредност се упоређује са тренутним минимумом (`minEval`), а промењљива `beta` се ажурира. Алфа-бета одсецање се примењује на исти начин као код AI играча. На крају се враћа `minEval`, односно најнеповољнија предвиђена вредност за AI играча, под претпоставком да човек игра оптимално.

На овај начин код симулира будуће потезе оба играча, максимизује шансе вештачког играча и минимизује његове губитке, уз оптимизацију броја обрађених чворова кроз алфа-бета одсецање.

Метода `NajboljiPotez` враћа потез играча који ће водити ка најповољнијем исходу. За разлику од методе `Minimax` која враћа само нумеричку процену квалитета стања, метода `NajboljiPotez` користи те процене да би одредила конкретан потез у виду колоне. У старту иницијализују се вредности `najboljaVrednost`=`int.MinValue` и `najboljaKolona`=-1. Метода пролази кроз све валидне колоне (`foreach (int kolona in stanje.Tabla.validne_pozicije())`), за сваку колону клонира тренутно стање и симулира убацивање AI токена у ту колону. Позивањем методе `Minimax` проласком кроз петљу добијамо различите вредности промењљиве `vrednost`, које поредимо и тиме добијамо вредност за најбољу колону (она за коју је `vrednost`) највећа.

```
public int NajboljiPotez(Stanje stanje)
{
    int najboljaKolona = -1;
    int najboljaVrednost = int.MinValue;

    foreach (int kolona in stanje.Tabla.validne_pozicije())
    {
        // napravi klon trenutnog stanja
        Stanje novoStanje = stanje.Klon();
        novoStanje.Tabla.igraj(kolona, -1); // AI potez
        novoStanje.promeni_igraca(); // menja igraca za sledeci nivo

        int vrednost = Minimax(novoStanje, dubina - 1, false, int.MinValue, int.MaxValue);

        if (vrednost > najboljaVrednost)
        {
            najboljaVrednost = vrednost;
            najboljaKolona = kolona;
        }
    }

    return najboljaKolona;
}
```

Слика 4.10. Метода `NajboljiPotez`

4.2.5. Кориснички интерфејс

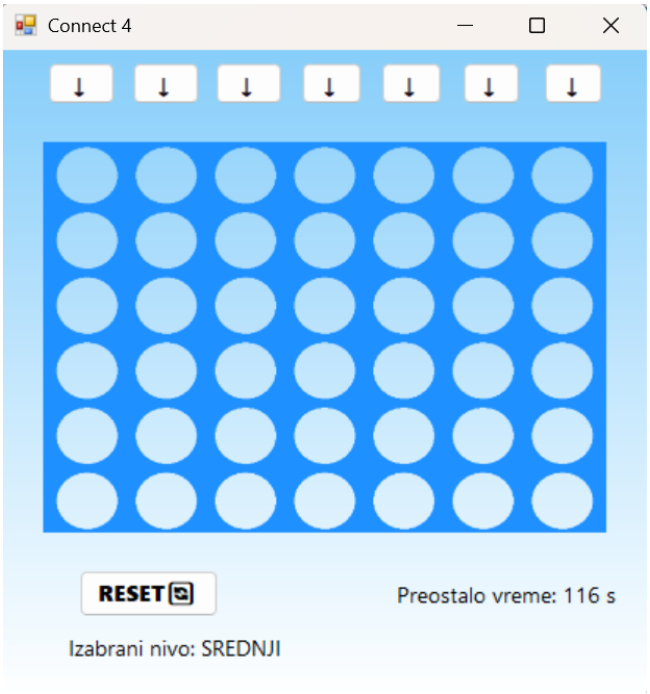
Кориснички интерфејс је дизајниран тако да омогућава кориснику да управља игром и прати њено стање на једноставан и прегледан начин. Креиран је у окружењу *Visual Studio Express 2012* коришћењем *Windows Forms* технологије.

На самом почетку, након покретања игре, (слика 4.11), корисник мора одабрати један од три понуђена нивоа тежине игре - лако, средње или тешко. Кликом на један од три дугмета, `dubina` у класи AI се подешава на адекватну вредност:

- лак ниво → `dubina`=2
- средњи ниво → `dubina`=3
- тежак ниво → `dubina`=5

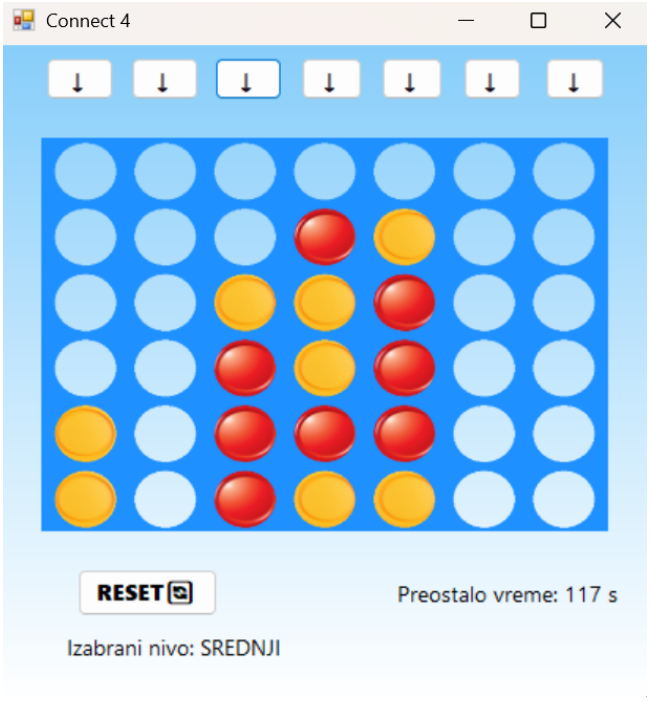
Што је већа дубина, то AI прави веће стабло игре, чиме "више гледа унапред" и отежава игру.

Слика 4.11. Почетни екран, одабир нивоа



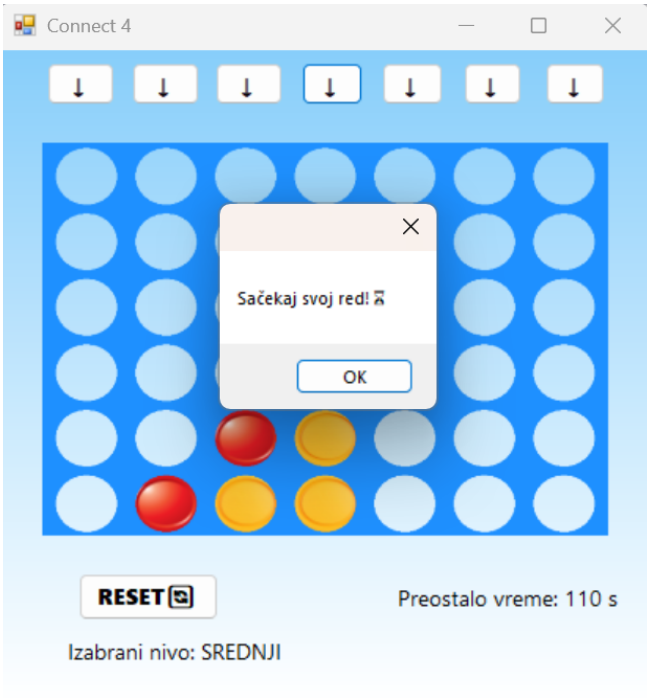
Слика 4.12. Почетак игре

Након одабира нивоа, отвара се основни екран за игру који изгледа као на слици 4.12. По узору на праву игру *Connect 4*, овде је такође постављена плава табла димензије 6x7 поља. Током игре токени AI су црвени, док су корисникови жути. Корисник убацује свој токен у жељену колону кликом на дугме са стрелицом изнад колоне. Он има ограничено време да одигра свој потез (2 минута, тј. 120 секунди). Ако не одигра потез у задатом временском оквиру аутоматски губи игру. Овај део је имплементиран да би се онемогу-



Слика 4.13. Изглед игре током партије

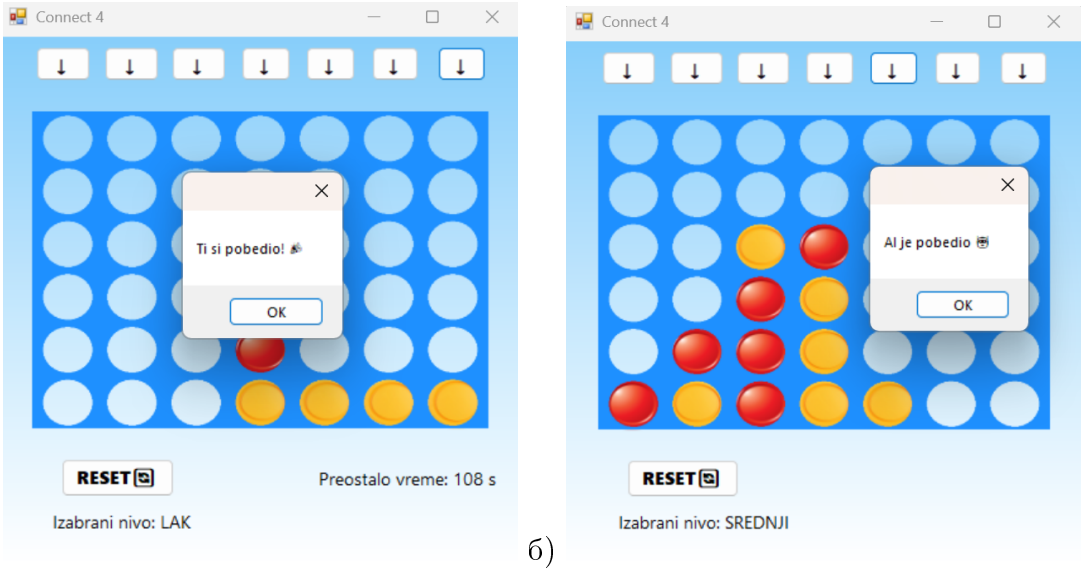
ћило неограничено разматрање могућих потеза и да би се обезбедио динамичан ток игре. Корисник може да види преостало време у доњем десном углу екрана (доњи десни угао слике 4.12). Такође му је омогућено да види који ниво је изабрао на почетку. Уколико из било ког разлога пожели да почне игру испочетка, кликом на дугме `RESET` вратиће се на почетни екран и биће у могућности да поново изабере ниво и започне нову партију.



Слика 4.14. Порука кориснику да сачека свој потез

Изглед екрана током игре приказан је на слици 4.13. Ако корисник покуша да притисне

дугме са стрелицом и убаци свој токен док је још увек на потезу рачунар, биће спречен у томе и појавиће се порука приказана на слици 4.14. Ово је имплементирано како корисник не би могао да убацује више токена од AI играча.



а) б)

Слика 4.15. Резултат игре: а) Победа човека; б) Победа AI

На крају игре, где је исход победа неког играча, појављује се *MessageBox* са одговарајућом поруком, као што је приказано на сликама 4.15 а) и 4.15 б).

Глава 5

Закључак

У оквиру овог рада реализована је игра са вештачком интелигенцијом заснована на *Minimax* алгоритму са алфа-бета одсецањем. Циљ рада је био да се што боље објасни принцип рада *Minimax* алгоритма, да се овај алгоритам имплементира, као и да се прикаже пример његовог коришћења.

Имплементацијом игре омогућено је да играч игра против рачунара који користи *Minimax* алгоритам за избор оптималног потеза. На тај начин је демонстрирано како рачунар може да анализира могуће исходе потеза, процени њихове последице и изабере потез који води ка најповољнијем резултату.

Током израде рада, посебан значај је дат ефикасности алгоритма и оптимизацији његовог рада, као и корисничком интерфејсу који омогућава прегледно и интуитивно играње. Уведени елементи попут тајмера додатно доприносе динамици игре и реалистичнијем искуству корисника.

Закључно, овај рад показује да се *Minimax* алгоритам успешно може применити у једноставнијим стратешким играма као што је *Connect 4*, али и представља основу за разумевање сложенијих система вештачке интелигенције у играма и доношењу одлука.

Литература

[1] A. M. Meystel and J. S. Albus, *Intelligent Systems: Architectures, Design, Control.* John Wiley & Sons, 2002.

[2] N. Sharma, J. Hug, J. Liang, and H. Zhu. (2026) Introduction to artificial intelligence. University of California at Berkeley. Датум последњег приступа: 03.05.2026. [Online]. Available: https://inst.eecs.berkeley.edu/~cs188/textbook/

[3] E. Mayefsky, F. Anene, , and M. Sirota. (2026) Strategies and tactics for intelligent search. Stanford University. Датум последњег приступа: 03.05.2026. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/intro.html

[4] Great Learning Editorial Team. (2025) Alpha beta pruning in ai. Датум последњег приступа: 03.05.2026. [Online]. Available: https://www.mygreatlearning.com/blog/alpha-beta-pruning-in-ai/

[5] Amazon. (2026) Hasbro gaming the classic game of connect 4, grid, get 4 in a row strategy game for 2 players ages 6 & up, multicolor. Датум последњег приступа: 03.05.2026. [Online]. Available: http://www.url-to-website.com

[6] J. Liberty, *Programming C#.* O'Reilly, 2002.