Универзитет у Источном Сарајеву

Електротехнички факултет

ПРОЈЕКТНИ РАД

ПРЕДМЕТ: Вјештачка интелигенција

ТЕМА: Стратешка игра Доминација (Domineering)

Ментор:
Проф. Др Драгољуб Крнета

Студент:
Јована Голијанин

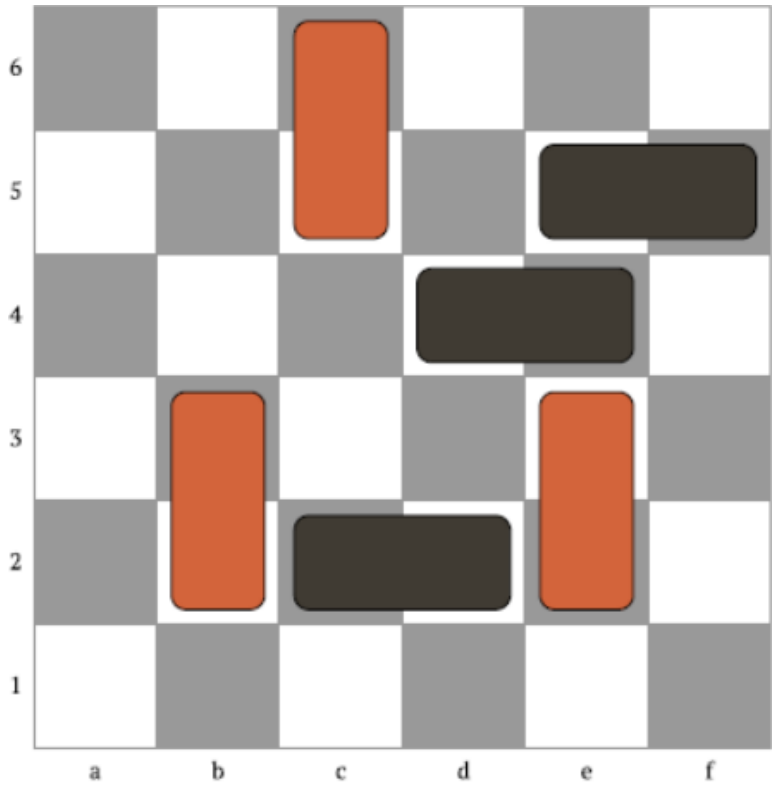
Источно Сарајево, јули 2023.

Садржај

Увод

Опис проблема

Проблем је стратешка игра постављања плочица на таблу која се зове Доминација (Domineering). Табла је произвољних димензија m×n, m врста и n колона (препоручено је 8×8). Плочице су величине 2×1. Два играча црни и бели (X и O) наизменично одигравају по један потез. Не постоји ограничење у броју плочица које играчи поседују. Један играч поставља плочице хоризонтално, док их други поставља вертикално. Поражен је онај играч који нема могућност да постави плочицу на таблу. Игра човек против рачунара и могуће изабрати да први игра човек или рачунар.

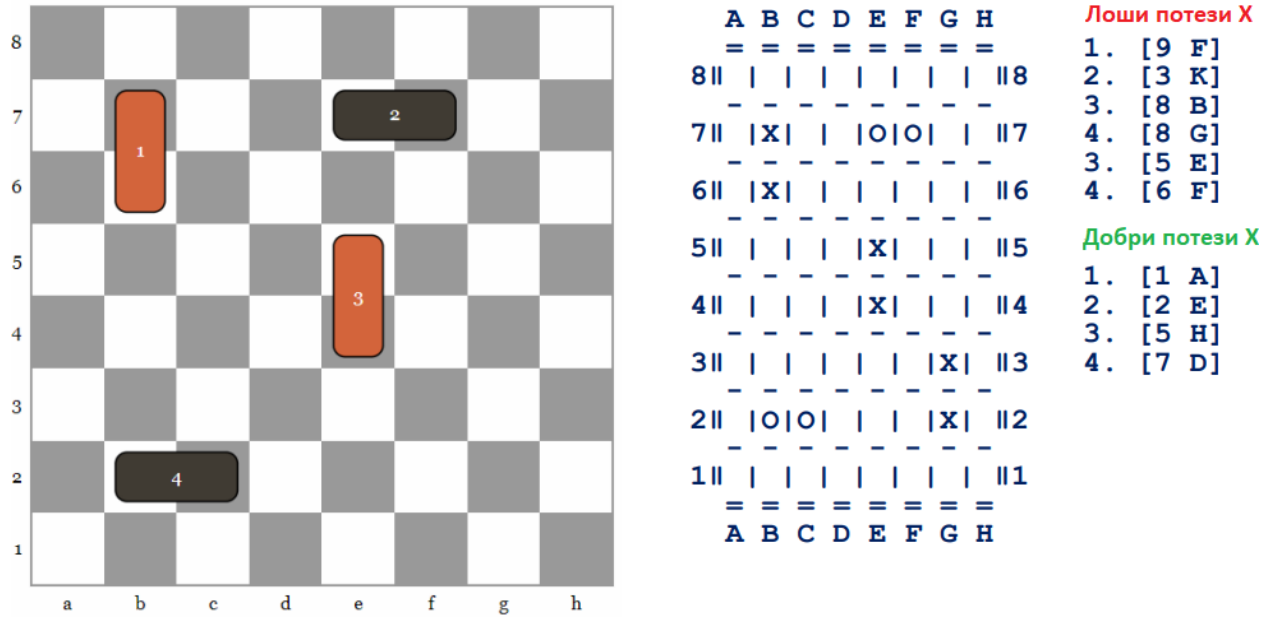


Слика1. Табла - пример стања игре

Правила игре

На почетку игре табла је празна. Играчи повлаче потезе наизменично. Један играч поставља плочице само хоризонтално, док их други поставља само вертикално. Први играч поставља плочице вертикално. Играч, у једном потезу, поставља само једну плочицу на таблу у одговарајућем смеру (хоризонтално и вертикално). Плочице се не смеју преклапати, нити вирити ван табле.

Плочице се не могу поставити тако да почињу на пола поља и завршавати се на пола поља. Поражен је онај играч који нема могућност да постави плочицу на таблу.



Слика 2. Начин постављања плочица

Циљ истраживања

Стећи увид у основне области и правце истраживања вештачке интелигенције. Стећи знање о основним алгоритмима из различитих области вештачке интелигенције (практична реалаизација софтверске апликације се базира на коришћењу мин-макс алгоритма са алфа-бета одсецањем), као и могућност њихове примене у решавању конкретних проблема (имплементирана игра "Доминација"). Стећи увид у могућности програмских језика вештачке интелигенције(конкретно могућности програмског језика Python) за имплементацију изабраног алгоритма.

Коришћене технологије

Visual Studio Code - Креирање едитовање, дебагирање апликације

Python - Програмски језик на којем је имплементирана логика игрице

ASP.NET Core - Вишеплатформски framework отвореног кода за креирање веб апликација

C# - Програмски језик уз помоћ којег су креиране класе за прихватање података из базе података

Entity framework - (EF) је ORM (Object-Relational Mapping) framework, који омогућава рад са подацима користећи објекте, без фокуса на базу података

EF Core - Entity framework Core је open-source, cross-platform, проширива верзија EF која се извршава на .Net Core платформи. Има подршку за велики број база: (SQL Server, Sqlite, InMemory, Cosmos, PostgreSQL, MySQL, и друге.

Model EF Core - Модел се састоји од класа које одговарају подацима у бази података и једне засебне класе, која представља контекст. Класа која се користи за контекст мора да наслеђује класу DbContext. Свака табела у бази у овој класи треба да има одговарајући property, који чува колекцију коју је база вратила. Треба да има и методу OnConfiguring у којој се подешава connection string на базу података која ће да се користи.

Azure Data Studio - SQL LocalDB - база података

Миграције - Миграције су начин да се одржи усклађеност између шеме база података и EF Core модела. Потребно је креирати EF Core класе за потребе апликације, а потом покренути миграцију која ће креирати одговарајућу шему базе података за задате класе. Након измене у доменским класама, потребно је покренути миграцију поново како би се ажурирала шема базе.

Кратак садржај

Пројекат је реализован кроз четири фазе. Прва фаза представља формулацију проблема и интерфејс. У другој фази је имплементиран оператор промене стања, реализован у функцији у којој се на основу задатог играча на потезу и задатог стање игре (табле) **формирају сва могућа стање игре** (све могуће табле). У трећој фази коришћен је Min-Max алгоритам и хеуристика. У четвртој фази коришћене су наведене технологије како би се обезбедило повезивање апликације са базом података.

Фаза I - Формулација проблема и интерфејс

У првој фази је потребно дефинисати начин за представљање стања проблема (игре): табла, позиције плочица на табли, написати функције за постављање почетног стања које се дефинише на основу задате величине табле, написати функције за проверу краја игре (играч не може поставити ни једну плочицу на таблу) и написати функције које проверавају исправност унетог потеза.

Потребно је омогућити избор **ко ће играти први** (човек или рачунар).

Наведени захтеви испуњени су имплементацијом функције koIgraPrvi() која од корисника очекује унос путем конзоле: 0 уколико човек игра први или 1 уколико рачунар игра први.

```
def koIgraPrvi():
    global igraPrvi
    print("Ko igra prvi? (0-covek, 1-racunar):")
    unos = int(sys.stdin.readline())
    while (unos not in range(0,2)):
        print("Uneli ste nevalidnu vrednost. Unesite O ili 1.")
        unos = int(sys.stdin.readline())
    igraPrvi = unos

    if(igraPrvi==0):
        print("Covek je prvi na potezu!")
    else:
        print("Racunar je prvi na potezu!")
```

Потребно је имплементирати функције које обезбеђују **унос почетних параметара игре**. Унос димензија табле (m и n), као и функције које обезбеђују **прављење иницијалног стања проблема** (игре) тј. прављење празне табле на основу задатих димензија (m и n).

Наведени захтеви испуњени су имплементацијом функције init() која обезбеђују унос почетних параметара игре тј. димензије табле (m и n) које корисник уноси путем конзоле и прављење празне табле на основу задатих димензија (m и n) која је предсатвљена матрицом

```
def init():
    global matrica
    global brojVrsta
    global brojKolona

    print("Unesite broj vrsta:")
    unos = int(sys.stdin.readline())
```

```
    while (unos < 2):
        print("Uneli ste nevalidnu vrednost. Unesite broj veći od
1.")
        unos = int(sys.stdin.readline())
    brojVrsta = unos

    print("Unesite broj kolona:")
    unos = int(sys.stdin.readline())
    while (unos < 2):
        print("Uneli ste nevalidnu vrednost. Unesite broj veći od
1.")
        unos = int(sys.stdin.readline())
    brojKolona = unos

    matrica = [ [ " " for i in range(brojKolona) ] for j in
range(brojVrsta) ]
```

Потребно је имплементирати функције које обезбеђују **приказ произвољног стања** проблема (игре) као и приказ задате табле.

Наведени захтеви испуњени су имплементацијом функција stampaj(), top(brojKolona) и bottom(brojKolona) које врше приказ произвољног стања и задате табле.

```
def stampaj(matrica):
    top(brojKolona)

    for x in range(0,brojVrsta):
        print(str(brojVrsta-x)+"||",end='')
        for num in range(0,brojKolona):
            print(matrica[x][num]+'|',end='')
        print("|"+str(brojVrsta-x))

    bottom(brojKolona)

def top(brojKolona):
    print('  ',end='')
    for num in range(0,brojKolona):
        print(' '+string.ascii_uppercase[num],end='')
    print('')
    print('  ',end='')
    for num in range(0,brojKolona):
        print(' =',end='')
    print('')


def bottom(brojKolona):
    print('  ',end='')
    for num in range(0,brojKolona):
        print(' =',end='')
    print('')
    print('  ',end='')
```

```
    for num in range(0,brojKolona):
        print(' '+string.ascii_uppercase[num],end='')
    print('')
```

Потребно је реализовати функције за **унос потеза.** Потез је облика играч (X или O) и позиција доњег левог поља где ће се наћи плочица [4,C]. Ако је играч X (вертикално), плочица би требало да се постави на поља [4,C] и [5,C]. Ако је играч O (хоризонтално), плочица би требало да се постави на поља [4,C] и [4,D].

Наведени захтеви испуњени су имплементацијом функције odigrajPotez(vrsta,kolona) која проверава да ли је потез ваљан и уколико јесте, проверава да ли је играч X или играч O на потезу; одиграва потез за одговарајућег играча тиме што на таблу поставља одговарајући знак (вертикалну домину за играча X или хоризонталну домину за играча O); након одиграног потеза штампа одиграни потез. Такође, проверава да ли смо дошли до краја игре или смо унели невалидан потез.

```
def odigrajPotez(vrsta,kolona):
    global matrica
    global naPotezu

    if(potezValjan(vrsta,kolona)):
        if(naPotezu=='X'):
            matrica[vrsta][kolona]='X'
            matrica[vrsta+1][kolona]='X'
            naPotezu='O'
        else:
            matrica[vrsta][kolona]='O'
            matrica[vrsta][kolona+1]='O'
            naPotezu='X'
        stampaj(matrica)
    else:
        print("Uneli ste nevalidni potez!")
```

Потребно је реализовати функције које проверавају **да ли је потез ваљан.** Проверити да плочица није ван табле. Проверити да се плочица не поклапа са неком већ постављеном плочицом.

Наведени захтеви испуњени су имплементацијом функције potezValjan(vrsta,kolona) која проверава да ли је плочица ван табле и да ли се плочица не поклапа са неком већ постављеном плочицом.

```
def potezValjan(vrsta,kolona):
    if(vrsta<0 or kolona<0):
        return False
    if(naPotezu=='X'):
        if((vrsta+1)>=brojVrsta or kolona>=brojKolona):
            return False
        if(matrica[vrsta][kolona]==' ' and
matrica[vrsta+1][kolona]==' '):
            return True
```

```
            else:
                return False
        else:
            if(vrsta>=brojVrsta or (kolona+1)>=brojKolona):
                return False
            if(matrica[vrsta][kolona]==' ' and
matrica[vrsta][kolona+1]==' '):
                return True
            else:
                return False
```

Потребно је написати функције за **проверу краја игре** који наступа уколико играч не може поставити ни једну плочицу на таблу.

Наведени захтеви испуњени су имплементацијом функције krajIgre() која проверава да ли постоје слободна поља за играча који је на потезу. Уколико нема слободних поља тада је крај игре.

```
def krajIgre(trenutno_stanje,na_potezu):
    if(na_potezu=='X'):
        for i in range(0,brojVrsta-1):
            for j in range(0,brojKolona):
                if(trenutno_stanje[i][j]==' ' and
trenutno_stanje[i+1][j]==' '):
                    return False
    else:
        for i in range(0,brojVrsta):
            for j in range(0,brojKolona-1):
                if(trenutno_stanje[i][j]==' ' and
trenutno_stanje[i][j+1]==' '):
                    return False
    return True
```

На крају је имплементирана функција start() која позива горе дефинисане функције; у почетку испитује који је играч први на потезу, поставља иницијално стање и штампа почетни изглед табле. Докле год играчи имају слободних места где могу да поставе своју домину, они наизменично одигравају потезе. Када више нема слободних места ни за једног играча, игра је готова. Победио је онај играч који је последњи поставио домину на таблу а изгубио онај који је први остао без могућности да постави своју домину.

```
def start():
    global kraj
    global matrica

    novaFunkcija()

    logovanje()

    koIgraPrvi()
```

```
    init()

    stampaj(matrica)

    while True:
        if(krajIgre(matrica,naPotezu)):
            pobedio=""
            if(naPotezu=='X'):  #Ukoliko je covek prvi i izgubi
ovo ce se pozvati
                pobedio='O'
                igracIzgubio()
            else:
                pobedio='X'
                igracPobedio()
            stampaj(matrica)
            print("Kraj Igre, pobedio je igrac "+pobedio+"!!!")
            break

        print("\nIgrac "+naPotezu+" na potezu")
        pocetno_stanje=[row[:] for row in matrica]
        vrednost, potez = minimax(pocetno_stanje, 5, -math.inf,
+math.inf,naPotezu)
        if (igraPrvi==1 and naPotezu=='X') or (igraPrvi==0 and
naPotezu=='O'):
            odigrajPotez(potez[0],potez[1])
        else:
            print("Unesite vrstu (broj 1..N)(-1 prekid igre, -2
stampanje svih mogucih stanja):")
            vrsta = int(sys.stdin.readline())
            if vrsta == -1:
                break
            else:
                if vrsta == -2:
                    svaMogucaStanja()
                    print("Unesite vrstu (broj 1..N)(-1 prekid
igre, -2 stampanje svih mogucih stanja):")
                    vrsta = int(sys.stdin.readline())
                    print("Unesite kolonu (slovo A..Z):")
                    kolona = sys.stdin.readline()
                    kol = ord(kolona[0])-65
                    vrs = brojVrsta-vrsta
                    odigrajPotez(vrs,kol)
                else:
                    print("Unesite kolonu (slovo A..Z):")
                    kolona = sys.stdin.readline()
                    kol = ord(kolona[0])-65
                    vrs = brojVrsta-vrsta
                    odigrajPotez(vrs,kol)
```

Фаза II -Оператор промене стања

У другој фази је потребно реализовати функције које на основу задатог играча на потезу и задатог стање игре (табле) **формирају сва могућа стање игре** (све могуће табле), коришћењем функција из претходне ставке.

Наведени захтеви испуњени су имплементацијом функције svaMogucaStanja() која на основу задатог играча на потезу и задатог стање игре (табле) штампа сва могућа стање игре тј. све могуће табеле.

```
def svaMogucaStanja():
    if(naPotezu=='X'):
        for i in range(0,brojVrsta-1):
            for j in range(0,brojKolona):
                if(matrica[i][j]==' ' and matrica[i+1][j]==' '):
                    matrica[i][j] = 'X'
                    matrica[i+1][j] = 'X'
                    stampaj(matrica)
                    matrica[i][j] = ' '
                    matrica[i+1][j] = ' '
    else:
        for i in range(0,brojVrsta):
            for j in range(0,brojKolona-1):
                if(matrica[i][j]==' ' and matrica[i][j+1]==' '):
                    matrica[i][j] = 'O'
                    matrica[i][j+1] = 'O'
                    stampaj(matrica)
                    matrica[i][j] = ' '
                    matrica[i][j+1] = ' '
```

Уколико у функцији start() корисник унесе за вредност врсте -2, тада се позива функција svaMogucaStanja() која штампа све могуће комбинације које играч може да одигра у тренутном потезу.

Фаза III – Min-Max алгоритам и хеуристика

У трећој фази је потребно имплементирати Min-Max алгоритам са алфа-бета одсецањем за задати проблем (игру): На основу задатог стања проблема, На основу дубине претраживања, На основу процене стања (хеуристике) која се одређује када се достигне задата дубина тражења, Враћа потез који треба одиграти или стање у које треба прећи

Реализовати функције које обезбеђују одигравање партије између човека и рачунара

Наведени захтеви испуњени су имплементацијом функције minimax() која има

улазне параметре: почетно стање, дубину, параметре алфа и бета који памте вредност најбољег потеза до датог тренутка за Max и Min играча, респективно. На самом почетку најбољи потез је иницијализован на None. Након тога, ако је игра завршена или смо достигли максималну дубину претраге, враћамо хеуристику тренутног стања. Иницијализујемо најбољу вредност на минус бесконачно за играча који максимизира(X) или плус бесконачно за играча који минимизира(O). Позивамо minimax за резултујуће стање, ажурирамо најбољу вредност и најбољи потез ако је потребно, као и вредност алфа и бета за алфа-бета одсецање, уколико је на потезу X, односно O. Ако је бета мање или једнако алфа, не вршимо даљу претрагу, што је и главна карактеристика алфа-бета одсецања. На крају враћамо најбољу вредност и одговарајући најбољи потез.

```
def minimax(pocetno_stanje, dubina: int, alpha: int, beta: int,
na_potezu):

    najbolji_potez = None

    kraj = krajIgre(pocetno_stanje,naPotezu)
    if kraj==True or dubina == 0:
        return heuristika(pocetno_stanje), None

    if na_potezu=='X':
        najbolja_vrednost = -math.inf
        for potez in validni_potezi_X(pocetno_stanje):
            sledece_stanje =
sledeceStanje(pocetno_stanje,potez[0], potez[1],'X')
            vrednost, _ = minimax(sledece_stanje, dubina - 1,
alpha, beta,'O')
            if vrednost > najbolja_vrednost:
                najbolja_vrednost = vrednost
                najbolji_potez = potez
            alpha = max(alpha, najbolja_vrednost)
            if beta <= alpha:
                break
    else:
        najbolja_vrednost = +math.inf
        for potez in valdini_potezi_O(pocetno_stanje):
            sledece_stanje =
sledeceStanje(pocetno_stanje,potez[0], potez[1],'O')
            vrednost, _ = minimax(sledece_stanje, dubina - 1,
alpha, beta,'X')
            if vrednost < najbolja_vrednost:
                najbolja_vrednost = vrednost
                najbolji_potez = potez
            beta = min(beta, najbolja_vrednost)
            if beta <= alpha:
                break
    return najbolja_vrednost, najbolji_potez
```

У minimax функцији је коришћена функција heuristika() за процену стања, које се

одређује када се достигне задата дубина тражења, као што је већ описано. Хеуристика враћа разлику броја валидних потеза за X и O. Позитивну вредност уколико постоји предност за X, негативну вредност уколико постоји предност за O, а нулу уколико нема предности. Функција за хеуристику користи две помоћне функције: valdini_potezi_X() и valdini_potezi_O(), које враћају листу валидних потеза за X и O, респективно.

```
def validni_potezi_X(stanje_matrice):
    potezi = []
    for i in range(0,brojVrsta-1):
        for j in range(0,brojKolona):
            if(stanje_matrice[i][j]==' ' and
stanje_matrice[i+1][j]==' '):
                potezi.append([i,j])
    return potezi

def valdini_potezi_O(stanje_matrice):
    potezi = []
    for i in range(0,brojVrsta):
        for j in range(0,brojKolona-1):
            if(stanje_matrice[i][j]==' ' and
stanje_matrice[i][j+1]==' '):
                potezi.append([i,j])
    return potezi

def heuristika(trenutno_stanje):
    validni_X = validni_potezi_X(trenutno_stanje)
    validni_O = valdini_potezi_O(trenutno_stanje)
    return len(validni_X) - len(validni_O)
```

Коришћена је и функција sledeceStanje() која враћа следеће стање на основу тренутно прослеђеног.

```
def sledeceStanje(stanje,vrsta,kolona,igrac):
    if(igrac=='X'):
        stanje[vrsta][kolona]='X'
        stanje[vrsta+1][kolona]='X'
    else:
        stanje[vrsta][kolona]='O'
        stanje[vrsta][kolona+1]='O'
    return stanje
```

Фаза IV- Повезивање са базом података

Потребно је креирати класе које одговарају одговарајућим табелама у бази.

У ту сврху користимо моделе јер они представљају класе које одговарају одговарајућим табелама у бази. Можемо рећи и да су модели објекти који одговарају подацима из базе. За сваку табелу која постоји у бази, креира се један модел. Податке које желимо да уписемо у базу, уписујемо у модел и обрнуто.

Моделе мапирамо на одговарајуце табеле у бази. Атрибуте које је потребно имати у бази,(име, презиме, број телефона...) је потребно написати овде. Касније ће позивом миграција одговарајуће табеле и колоне у њој да се креирају на основу модела који су овде написаи.

Наведени захтеви испуњени су имплементацијом класа Igrac.cs и Partija.cs

```
namespace Models;
public class Igrac
{
    [Key]
    public int ID { get; set; }
    public required string Ime { get; set; }
    public int BrojIndeksa { get; set; }
    public int BrojPobeda { get; set; }
    public int BrojNeresenih { get; set; }
    public int BrojPoraza { get; set; }
    public List<Partija>? ListaPartija { get; set; }
}
```

```
namespace Models;

public class Partija
{
    [Key]
    public int ID { get; set; }
    public DateTime VremePocetka { get; set; }
    public DateTime VremeKraja { get; set; }
    public DateTime DuzinaPartije {get; set; }

    public required string PobednikCovek { get; set; }
    public Igrac? Igrac { get; set; }
}
```

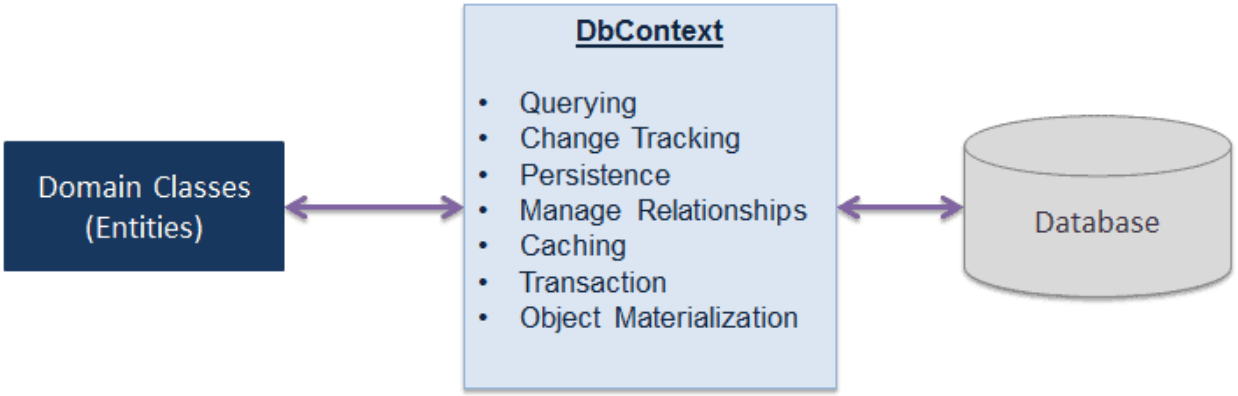
Потребно је успоставити конекцију са базом.
Наведени захтев испуњен је креирањем класе Context.cs

```
namespace Models;

public class Context : DbContext
{
    public required DbSet<Igrac> Igrac { get; set; }
    public required DbSet<Partija> Partija { get; set; }
    public Context(DbContextOptions options) : base(options)
    {
    }
}
```

DbContext инстанца представља сесију са базом и може се користити за учитавање и

памћење инстанци ентитета. DbContext успоставља конекцију са базом. DbContext је нека врста посредника између базе података и класа. -DbSet се мапира на табеле. Представља табелу у бази. За сваку табелу у бази, креирамо по један DbSet. Увек имамо класу Context.cs и увек је пишемо на исти начин као што је приказано на слици.



Слика 3. DbContext

Потребно је остварити упис, читање, измену и брисање података из базе.

Наведени захтеви испуњени су имплементацијом контролера. У контролеру је потребно написати CRUD (Create, Read, Update, и Delete) методе. Помоћу ових метода могуће је остварити упис, читање, измену и брисање података из базе.

[HttGet]

Метода за учитавање података из базе. Постоји два типа Get методе. Први тип повлачи све податке из табеле. ToListAsync() позива одговарајуће SQL упите ка бази и враћа селектоване податке. Тек кад се напише ToListAsync() се врши упит ка бази, пре тога не.

```
[HttpGet("VratiIgrace")]
public async Task<ActionResult> VratiIgrace()
{
     try
    {
        var igraci = await Context.Igrac.Select(p=>new{
            id = p.ID,
            ime = p.Ime,
            brojIndeksa = p.BrojIndeksa,
            brojPobeda = p.BrojPobeda
        }).ToListAsync();
        return Ok(igraci);
    }
    catch(Exception e)
    {
```

```
            return BadRequest(e.Message);
        }
    }

    [HttpGet("VratiIgraca")]
    public async Task<ActionResult> VratiIgraca(int brojIndeksa)
    {
         try
        {
            var igrac = await Context.Igrac.Where(p=>p.BrojIndeksa
== brojIndeksa).ToListAsync();

            return Ok(igrac);
        }
        catch(Exception e)
        {
            return BadRequest(e.Message);
        }
    }
```

[HttPost]

[HttpPost] служи за унос података у базу. Подаци се могу додавати као аргументи, FromBody или FromQuery. Аналогно важи и за остале CRUD операције Get и Put.

```
    [HttpPost("DodajIgraca")]
    public async Task<ActionResult> DodajIgraca(string ime, int
brojIndeksa)
    {
        try
        {
            var dohvati = await
Context.Igrac.Where(p=>p.BrojIndeksa ==
brojIndeksa).FirstOrDefaultAsync();

            if(dohvati != null)
                return StatusCode(202, "Igrac sa datim brojem
indeksa vec postoji u bazi");

            Igrac igrac = new Igrac
            {
                Ime = ime,
                BrojIndeksa = brojIndeksa
            };

            Context.Igrac.Add(igrac);
            await Context.SaveChangesAsync();
            return Ok($"Dodat je novi igrac sa brojem indeksa:
{igrac.BrojIndeksa}");
        }
        catch(Exception e)
        {
```

```
            return BadRequest(e.Message);
        }
    }
```

[HttPut]

```
    [HttpPut("DodajPobedu")]
    public async Task<ActionResult> DodajPobedu(int brojIndeksa)
    {
         try
        {
            var igrac = await Context.Igrac.Where(p=>p.BrojIndeksa
== brojIndeksa).FirstOrDefaultAsync();
            igrac!.BrojPobeda = igrac.BrojPobeda +1;
            Context.Igrac.Update(igrac);
            await Context.SaveChangesAsync();
            return Ok(igrac);
        }
        catch(Exception e)
        {
            return BadRequest(e.Message);
        }
    }

    [HttpPut("DodajPoraz")]
    public async Task<ActionResult> DodajPoraz(int brojIndeksa)
    {
         try
        {
            var igrac = await Context.Igrac.Where(p=>p.BrojIndeksa
== brojIndeksa).FirstOrDefaultAsync();
            igrac!.BrojPoraza = igrac.BrojPoraza + 1;
            Context.Igrac.Update(igrac);
            await Context.SaveChangesAsync();
            return Ok(igrac);
        }
        catch(Exception e)
        {
            return BadRequest(e.Message);
        }
    }
}
```

Потребно је имплементирати додатне функције у Python-у које обезбеђују успешан рад целокупне апликације након повезивања са базом података.

Наведени захтеви испуњени су имплементацијом функција novaFunkcija(), ulogujteSe(), igracPobedio(), igracIzgubio() и logovanje()

```
def novaFunkcija():
    print("Ovo su svi igraci koji su uspeli da pobede racunar")
```

```
    response = requests.get('http://localhost:5173/VratiIgrace')
    data = response.json()
    for item in data:
     print("Ime:", item['ime'], "|  BrojIndeksa:",
item['brojIndeksa'], "|  BrojPobeda:", item['brojPobeda'])


def ulogujteSe():
    print("Unesite ime i prezime:")
    ime = sys.stdin.readline().strip()
    print("Unesite broj indeksa:")
    global brIndeksa
    brIndeksa = int(sys.stdin.readline())
    response =
requests.post(f'http://localhost:5173/DodajIgraca?ime={ime}&brojIn
deksa={brIndeksa}')
    if response.ok:
        print("Uspesno ste kreirali nalog, mozete da pocnete sa
igrom")
    else:
        print("Nazalost doslo je do greske, igracete kao guest")


def igracPobedio():
    global brIndeksa
    response =
requests.put(f'http://localhost:5173/DodajPobedu?brojIndeksa={brIn
deksa}')
    data = response.json()
    print("Cestitamo na pobedi.")


def igracIzgubio():
    global brIndeksa
    response =
requests.put(f'http://localhost:5173/DodajPoraz?brojIndeksa={brInd
eksa}')
    data = response.json()
    print("Vise srece drugi put")


def logovanje():
    print("Unesite broj indeksa kako bi ste igrali, ako prvi put
igrate morate da napravite nalog")
    global brIndeksa
    brIndeksa = int(sys.stdin.readline())
    response =
requests.get(f'http://localhost:5173/VratiIgraca?brojIndeksa={brIn
deksa}')
    data = response.json()
    if(data == []):
```

```
        print("Vas indeks ne postoji, molimo vas da napravite
nalog")
        ulogujteSe()
    else:
        print("Dobar dan", data[0]['ime'], "Uzivajte u igri.")
```

Закључак

Остварени циљеви:

Након завршетка овог пројекта стекла сам увид у основне области вештачке интелигенције, стекла практично знање о програмском језику Python уз помоћ којег сам имплементирала мин-макс алгоритма са алфа-бета одсецањем. Такође, упознала сам се са могућностима повезивања апликације са базом података.

Предности реализованог софтвера:

С обзиром да табеле у бази података садрже податке који се не могу директно конзумирати из C# апликације, да би се олакшао приступ појединим колонама и врстама, уз помоћ C#-а могуће је креирати класе, које одговарају подацима у бази. Након креирања, колекције могу да се користе за прикупљање података из базе података. Зато је **C#** изабран као програмски језик уз помоћ којег су креиране класе за прихватање података из базе података.

С обзиром да су стринг упити несигурни, без провере типова, intellisense-а, коришћен је **Entity framework (EF)** јер елиминише потребу за писањем SQL кода за приступ подацима из базе података. Коришћен је **LINQ** (Language Integrated Query) који представља скуп технологија за интеграцију упита директно у C# и он има све горенаведене могућност.

LINQ упити се оптимизују и DBMS-у се шаљу тако оптимизовани упити. То значи да се приступ табелама и колонама врши само уколико је неопходан.

Такође, приступ је одложен, што значи да креирањем самих упита и додељивањем „резултата“ одговарајућој промењивој, не мора да значи да ће да се позове прослеђивање података из базе података.

Подаци ће се проследити у оном тренутку, када је без њих више немогуће извршавати код.

Главна предност реализоване апликације је то што је коришћен **Minimax** алгоритам, о обзиром да он има доста предности у односу на друге алгоритме за претрагу простора игре. Само неке од тих предности су:

Гаранција оптималности: Minimax алгоритам гарантује да ће пронаћи оптимални потез за играча, ако се претраже сва могућа стања игре. То значи да ће пронаћи

потез који ће довести до најбољег резултата за играча, без обзира на потезе противника.

Универзална примењивост: Minimax алгоритам може се применити на различите врсте игара, укључујући игре са нултим-сумама (где добитак једног играча представља губитак другог) и игре са делимичном информацијом.

Једноставност имплементације: Minimax алгоритам је релативно једноставан за имплементацију, посебно за игре са малим бројем могућих потеза. Потребно је само рекурзивно претраживати простор игре и евалуирати стања игре.

Могућност комбиновања са хеуристикама: Minimax алгоритам се често комбинује са хеуристикама како би се смањио број стања која се евалуирају. Хеуристике могу брзо проценити вредност стања игре и користити се као оцене приликом претраге простора игре.

Паралелизација: Minimax алгоритам може се паралелизовати како би се убрзао процес претраге. Различите гране претраге могу се извршавати истовремено на различитим процесорима или рачунарима.

Важно је напоменути да Minimax алгоритам има и одређене недостатке, као што су експоненцијално растући простор претраге игре са повећањем дубине претраге и немогућност ефикасног управљања великим простором претраге за комплексне игре. Међутим, уз одговарајуће оптимизације и додатне технике као што је ***алфа-бета одсецање*** и ограничење дубине претраге, Minimax алгоритам може бити веома ефикасан и користан за претрагу простора игре. Управо из тог разлога је коришћено алфа-бета одсецање како би алгоритам радио што оптималније.

Коришћена литература

[1] S. Russell, P. Norvig: Artificial Intelligence: A Modern Approach, Prentice Hall Series in AI, 2010.

[2] Л.Стоименов, А.Милосављевић, Практикум за вежбе на рачунару из Вештачке интелигенције, Електронски факултет, Ниш, 2004.

[3] Д.Бојић, Д.Велашевић, В.Мишић, Збирка задатака из експертских система, Научна књига Београд, 1996.

[4] Lucci, S., Kopec, D. Artificial Intelligence In the 21st Century, A Living Introduction (second edition), Mercury Learning and Information 2016

[5] Watson, M. Practical Artificial Intelligence Programming With Java (third edition) 2008

[6] Raschka, S. Python Machine Learning, Packt Publishing 2015

[7] Bowles, M. Machine Learning in Python, Wiley 2015

[8] Witten, I. H., Frank, E. Data Mining Practical Machine Learning Tools and Techniques (second edition), Elsevier 2005