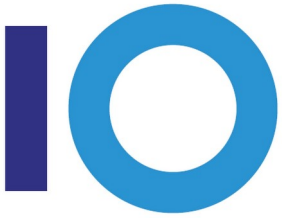


УНИВЕРЗИТЕТ „СВЕТИ КИРИЛ И МЕТОДОЈ“ - СКОПЈЕ
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖИНЕРСТВО

ИЗВЕШТАЈ ЗА НАПРЕДЕН ДЕЛ ПО ВЕШТАЧКА ИНТЕЛИГЕНЦИЈА

тема:

Спротивставено пребарување – Симулација на играта Дама

Ментор:
Проф. д-р Соња Гиевска

Студент:
Драгана Трифунова

Скопје, декември 2024.

Содржина

Вовед

Почитувани,

во оваа документација која го опишува напредниот дел по вештачка интелигенција ќе се занимаваме со примена на алгоритмите за спротивставено пребарување и тестирање на нивните перформанси.

За таа цел ќе ја разгледаме играта дама на која се имплементирани 2 алгоритми и 3 хевристички функции. Ќе направиме споредба на алгоритамот minimax со expectimax за да видиме кој е подобар и ќе ги истестираме на најразлични хевристики. Оваа игра е направена според американската верзија и има 12 пиончиња на компјутерот и 12 пиончиња на лицето кое ја игра. При започнување на играта, на самиот почеток се дадени правилата и начинот на конфигурација на играта. Документацијава содржи детално објаснување за начинот на играње како и најразлични симулации за споредба на нејзините перформанси за алгоритмите и хевристиките кои ги користи.

Почетна фаза на играта и конфигурација

Имплементацијата на целата игра е зададена во myProject.py. При компајлирање на играта и нејзино извршување се добива следниот прозорец:

```
 *** DOBREDOJDOVTE VO IGRATA DAMA ***
Ova e implementacijata na studentot Dragana Trifunova [223044] za igrata dama i ovde e prikazana amerikanskata verzija od igrata
Vo Amerikanskata verzija postojat 12 pioncinja na sekoi od protivnicite i vo sekoi red ima po osum kelii

PRAVILA NA IGRANJE
1. Dokolku sakate da zapocnete so igranje napisete 'start'
2. Dokolku sakate da prekinete so igranje vo bilo koe vreme pritisnete 'Enter'
3. Dokolku sakate da se predadete napisete 'surrender'
4. Koordinatite 'i' i 'j' gi vnesuvate so zapirka pomegu niv bez prazno mesto
Pred da zapocnete so igrata (pred da napisete 'start') najprvin mora da ja napravite nejzinata konfiguracija.

!!! KONFIGURACIJA !!!

Dali sakate igra covek-algoritam ili pak sakate algoritmite da igraat megu niv 1/2?
```

Слика 1: Оригиналниот код за овој прозорец е превземен од:
https://github.com/dimitrijekaranfilovic/checkers.git
Прилагодено за потребите на проектот.

Во овој прозорец јасно може да се видат техничките правила на играта за нејзиното подесување. Играта се започнува со испишување на зборчето „start“, завршува со притискање на копчето „Enter“ а, доколку сакате да се предадете (само кога игра човек наспроти алгоритам и човекот одлучи дека сака да се предаде) треба да напишете „surrender“.
Координатите на таблата каде што стојат пионите изгледаат како координати на матрица односно почнуваат од горниот лев агол (0,0) а, завршуваат во долниот десен агол (7,7). Помегу нив се одвоени со запирка без празно место.

Конфигурација
Кога ќе ја извршите програмата во PyCharm под делот со правилата на играта се наоѓа дел за конфигурација. Во овој дел треба да одговорите на неколку едноставни прашања со кои одлучувате кои ќе бидат противниците во играта и со какви перформанси ќе се игра.

Првото прашање на кое треба да се одговори е „*Dali sakate igra covek-algoritam ili pak sakate algoritmite da igraat megu niv 1/2?* “

- Првата опција е да играте против алгоритам. И во тој случај човекот го прави првиот чекор, алгоритамот вториот и така најизменично.

- Втората опција е да играте алгоритам против алгоритам. Во тој случај кога ќе ја изберета втората опција ќе ја добиете следната порака:

Expectimax igra so pionceto oznaceno kako #men
Minimax igra so pionceto oznaceno kako comp

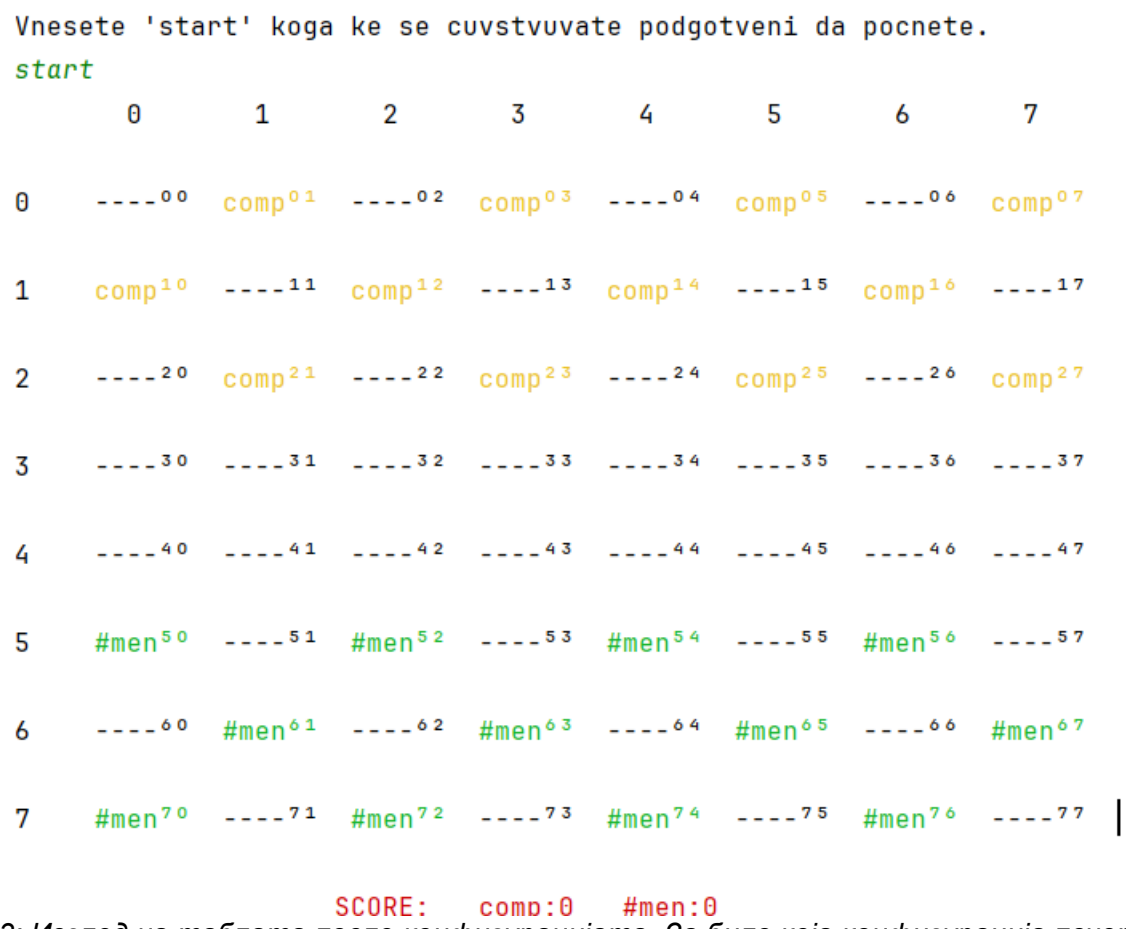
Ова значи дека секогаш прв на потег ќе биде Expectimax а, втор Minimax и така најизменично.

Второто прашање на кое треба да се одговори е хевристиката со која ќе се игра. Овде имаме можност од 3 понудени избори за кои ќе зборуваме подоцна во оваа документација.

На крајот пишуваме „start“ и започнуваме со играње.

```
!!! KONFIGURACIJA !!!

Dali sakate igra covek-algoritam ili pak sakate algoritmite da igraat megu niv 1/2?
2
Expectimax igra so pionceto oznaceno kako #men
Minimax igra so pionceto oznaceno kako comp
So koja hevristika ke igra minimax 1/2/3?
1) Best
2) Almost King
3) Edge Position
1
So koja hevristika ke igra expectimax 1/2/3?
1) Best
2) Almost King
3) Edge Position
2
Vnesete 'start' koga ke se cuvstvuvate podgotveni da pocnete.
```

Слика 2: Избирање на начин на играње и избирање на хевристика. Слика генерирана од сопствениот код.



Слика 3: Изглед на таблата после конфигурацијата. За било која конфигурација почетниот изглед на таблата е прикажан на оваа слика. Слика генерирана од сопствениот код.

Minimax алгоритам

Минимакс алгоритмот е еден од најважните алгоритми во областа на вештачката интелигенција, особено кога станува збор за игри со двајца играчи. Овој алгоритам пресметува оптимален потег од тековната состојба, користејќи рекурзивна пресметка на минимакс вредностите за секоја следна состојба. Минимакс вредноста на јазолот е вредноста (за MAX) на соодветната состојба, под претпоставка дека двата играчи играат оптимално до крајот на играта. За терминалните состојби, минимакс вредноста е едноставно нивната корист, што значи дека MAX ќе преферира да се префрли во состојба со максимална вредност, додека MIN ќе преферира состојба со минимална вредност.

Минимакс алгоритмот врши целосно длабинско пребарување на дрвото на играта. Ако максималната длабочина на дрвото е m, а бројот на легални потези на секој чекор е b, временската сложеност на минимакс алгоритмот е $O(b^m)$. Сепак, иако овој алгоритам е моќен, тој е непрактичен за сложени игри како шах, бидејќи бројот на состојби е експоненцијален. Од ова следува дека минимакс алгоритмот служи како основа за математичка анализа на игрите и за практични алгоритми, но не е доволен сам по себе за големи и сложени игри.

Оптималните стратегии во игрите со повеќе играчи бараат замена на една вредност за секој јазол со вектор на вредности. Минимакс алгоритмот претпоставува дека MIN игра оптимално, односно го максимизира најлошиот исход за MAX. Меѓутоа, ако MIN не игра оптимално, тогаш MAX ќе има уште подобар исход. Ова значи дека минимакс алгоритмот не само што е ефикасен, туку и флексибилен во различни сценарија.

Минимакс алгоритмот ги игнорира деловите од дрвото кои не влијаат на конечниот избор, што го прави ефикасен за игри како тик-так-тое, каде што целото дрво може да се пребарува. Минимакс вредноста на коренот е максималната вредност од минимакс вредностите на неговите деца, што значи дека овој алгоритам е основа за разбирање на оптималните стратегии во игрите.

Сепак, минимакс алгоритмот е популарен во истражувањата на вештачката интелигенција поради неговата едноставност и ефикасност. Тој може да се користи за да се осигура дека MAX ќе има најдобар можен исход, дури и против силен противник. Овој алгоритам е особено корисен за игри со мал број на можни потези и кратки игри, како што е тик-так-тое.

Минимакс вредноста на јазолот е дефинирана како максимум или минимум од вредностите на неговите деца, во зависност од тоа дали е MAX или MIN јазол. Оваа дефиниција е клучна за разбирање на тоа како минимакс алгоритмот функционира. Минимакс алгоритмот е основа за развој на понапредни техники како алфа-бета отсекување, кое ја намалува сложеноста на пребарувањето.

Според ова, минимакс алгоритмот е популарен во истражувањата на вештачката интелигенција поради неговата едноставност и ефикасност. Тој може да се користи за да се осигура дека MAX ќе има најдобар можен исход, дури и против силен противник.

Забелешка: Информациите и описот на овој алгоритам се извадени од книгата на Расел и Норвиг објавена на: https://courses.finki.ukim.mk/pluginfile.php/252202/mod_resource/content/1/Adversarial_search.pdf

```python
@staticmethod
def minimax(board, depth, alpha, beta, maximizing_player):
    if depth == 0:
        return GameCheckers.calculate_heuristics(board, 'minimax')
    current_state = NewGameState(deepcopy(board))
    if maximizing_player is True:
        max_eval = -math.inf
        for child in current_state.get_children(True):
            ev = GameCheckers.minimax(child.get_board(), depth - 1, alpha, beta, False)
            max_eval = max(max_eval, ev)
            alpha = max(alpha, ev)
            if beta <= alpha:
                break
        current_state.set_value(max_eval)
        return max_eval
    else:
        min_eval = math.inf
        for child in current_state.get_children(False):
            ev = GameCheckers.minimax(child.get_board(), depth - 1, alpha, beta, True)
            min_eval = min(min_eval, ev)
            beta = min(beta, ev)
            if beta <= alpha:
                break
        current_state.set_value(min_eval)
        return min_eval
```

Слика 4: Имплементација на minimax алгоритам. Оригиналниот код е превземен од: https://github.com/dimitrijekaranfilovic/checkers.git

Прилагодено за потребите на проектот.

Expectimax алгоритам

Expectimax е алгоритам кој се користи за игри кои вклучуваат елемент на случајност, како што се игри со коцки или карти. Овој алгоритам е проширување на минимакс алгоритмот, но наместо да претпоставува дека противникот (MIN) игра оптимално, тој ги зема предвид сите можни исходи од случајните настани, како што е фрлањето на коцки. Во суштина, Expectimax го пресметува очекуваниот (expected) исход на играта, наместо да се фокусира на најлошиот можен исход.

Во игри како backgammon, каде што коцките ја одредуваат можноста за потези, Expectimax алгоритмот вклучува chance nodes (шанси јазли) во дрвото на играта. Овие јазли претставуваат точки каде што случајноста влегува во игра, и секој гранка од chance node претставува еден можен исход од коцките. На пример, ако играчот фрли коцки и има 21 можни исходи, секој од нив ќе биде претставен како посебна гранка во дрвото.

Очекуваната вредност на chance node се пресметува како збир на вредностите на неговите деца, пондерирани според веројатноста за секој исход. Формулата за пресметување на вредноста на chance node е следна:

$$\text{EXPECTIMINMAX}(n) = \sum_{s \in \text{Successors}(n)} P(s) \cdot \text{EXPECTIMINMAX}(s)$$

каде P(s) е веројатноста за секој исход s. Ова значи дека Expectimax не само што ги зема предвид сите можни исходи, туку и нивната веројатност, што го прави идеален за игри со случајни елементи.

Една од клучните разлики помеѓу Expectimax и минимакс е тоа што Expectimax не претпоставува дека противникот игра оптимално. Наместо тоа, тој ги зема предвид сите можни потези на противникот, пондерирани според нивната веројатност. Ова го прави Expectimax попрецизен за игри каде што противникот може да прави грешки или да не игра оптимално.

Сепак, Expectimax има свои ограничувања. Бидејќи ги зема предвид сите можни исходи, вклучувајќи ги и оние со мала веројатност, временската

сложеност на овој алгоритам е значително поголема од онаа на минимакс. Ако бројот на можни исходи (n) е голем, како што е случајот со

backgammon (каде n = 21), Expectimax може да стане непрактичен за длабоко пребарување.

За да се справи со ова, Expectimax често се користи во комбинација со хевристички функции за евалуација. Овие функции ја проценуваат вредноста на состојбата без да ја истражуваат целата игра до крај. На пример, во backgammon, евалуациската функција може да ги земе предвид фактори како бројот на фигури на таблата, позицијата на фигурите и веројатноста за победа.

Важно е да се напомене дека евалуациската функција мора да биде позитивна линеарна трансформација на веројатноста за победа. Ова значи дека ако ја промениме скалата на вредностите на евалуациската функција, изборот на потези не треба да се промени. Ова е клучно за да се осигура дека алгоритмот ќе ги избере најдобрите потези без да биде чувствителен на произволни промени во вредностите.

Во пракса, Expectimax често се користи за игри како backgammon, каде што случајноста игра голема улога. На пример, ако играчот има четири можни потези, Expectimax ќе ги оцени сите четири, земајќи ги предвид сите можни исходи од коцките и нивната веројатност. Ова му овозможува на играчот да го избере потегот со највисока очекувана вредност.

Сепак, иако Expectimax е моќен алатка, тој не е секогаш најдобриот избор за игри со голема сложеност. На пример, во игри како покер или бриџ, каде што информациите се нецелосни, Expectimax може да биде непрактичен поради огромниот број на можни состојби и исходи. Во такви случаи, се користат понапредни техники, како што се методи за пребарување во просторот на верувања (belief states).

Како заклучок, Expectimax е моќен алгоритам за игри кои вклучуваат случајност. Тој ги комбинира принципите на минимакс со пресметка на очекувани вредности, што го прави идеален за игри како backgammon.

Сепак, неговата временска сложеност и потребата од прецизни евалуациски функции го прават помалку практичен за многу сложени игри.

```python
@staticmethod
def expectimax(board, depth, maximizing_player):
    if depth == 0:
        return GameCheckers.calculate_heuristics(board, 'expectimax')

    current_state = NewGameState(deepcopy(board))

    if maximizing_player:
        max_eval = -math.inf
        for child in current_state.get_children(True):
            ev = GameCheckers.expectimax(child.get_board(), depth - 1, False)
            max_eval = max(max_eval, ev)
        current_state.set_value(max_eval)
        return max_eval
    else:
        children = current_state.get_children(False)
        if not children:
            return GameCheckers.calculate_heuristics(board, 'expectimax')

        total_value = 0
        for child in children:
            ev = GameCheckers.expectimax(child.get_board(), depth - 1, True)
            total_value += ev
        expected_value = total_value / len(children)
        current_state.set_value(expected_value)
        return expected_value
```

Слика 5: Имплементација на expectimax алгоритам. Оригиналниот код е превземен од https://www.geeksforgeeks.org/expectimax-search-algorithm-in-ai/ Прилагоден за потребите на проектот.

Во мојата имплементација на Expectimax алгоритмот, не користев веројатности бидејќи се работи за играта дама (checkers), која е детерминистичка игра. Во дама, нема елемент на несигурност или случајност, како што е фрлање на коцки или случајно делење на карти.

Секој потег е целосно предвидлив и зависи само од логиката и стратегијата на играчите.

Забелешка: Информациите и описот на овој алгоритам се извадени од книгата на Расел и Норвиг објавена на: https://courses.finki.ukim.mk/pluginfile.php/252202/mod_resource/content/1/Adversarial_search.pdf

Објаснување на кодот

Имплементацијата на оваа игра се базира врз основа на minimax и expectimax алгоритамот. И овде морам да нагласам дека во овие алгоритми иако најчесто се зема дека компјутерот е максимален играч а, човекот минимален овде, во мојата игра компјутерот е минимален играч. Причината за ова е што при истражувањето на интернет за оваа проектна задача се послужив со такви имплементации каде компјутерот беше минимален играч.

Судејќи според ова, верувам дека веќе претпоставувате дека кога игра човекот наспроти алгоритамот првиот потег го прави човекот, вториот компјутерот и така најизменично. Нормално на ова евристиката е направена во корист на човекот односно кога човекот има добра положба евристиката враќа повеќе поени а, кога човекот има лоша положба евристиката враќа помалку поени. Целта на компјутерот е да ја минимизира својата штета односно да ја избере положбата во која човекот ќе се чувствува најлошо. Човекот зема максимум поени од евристиката а, компјутерот зема минимум.
Кога игра алгоритам наспроти алгоритам, сегогаш максимизирачкиот играч е **expectimax** а, минимизирачкиот е **minimax** односно expectimax игра прв.

Пред да почнеме со објаснувањето на секој метод поединечно сакам да ве наведам во тоа дека човекот игра со пионите означени како „#men“ а, компјутерот со пионите означени како „comp“.
Доколку играат алгоритмите меѓусебно тогаш **expectimax** игра со „#men“.

Во оваа имплементација на играта користени се две класи:
- NewGameState ова е класа која ја означува секоја нова положба на таблата. Се користи за да компјутерот открие кој потег треба да го преземе. Со помош на оваа класа
и методот во неа „get_children’’ компјутерот знае и ги тестира кои се можните потези на него и на човекот. Тој креира нова табла (матрица) за сите положби на него или на човекот и потоа со помош на хевристиката во minimax и expectimax алгоритамот им задава поени на тие табли.

- GameCheckers од оваа класа се креира само една истанца со помош на која се игра играта.

Оваа класа содржи: матрица односно таблата на играта, содржи булова променлива за тоа дали е на ред човекот или компјутерот иницијално означена како True затоа што прв започнува човекот, има две променливи една за поените на компјутерот а, другата за поени на играчот, има листа за можни поместувања на човекот или компјутерот.
Понатаму, оваа класа содржи број на пиони на човекот и на компјутерот иницијализирани на 12 затоа што ја играме американската верзија.
Исто така имаме и променлива која ја користиме исклучиво во ситуација кога играат човек алгоритам „minimaxAlgorithm“ која е иницијализирана на True, и го означува алгоритамот кој го користиме. Ако човекот го избере алгоритамот expectimax оваа променлива ќе се смени во False.
Во последните редови на конструкторот на GameCheckers ја иницијализираме матрицата односно ја трансформираме да изгледа како почетната состојба на играта Дама и ги иницијализираме времетраењата на алгоритмите на нула.

main()

Програмата започнува од main(). Таму правиме единствената истанца од класата GameCheckers и на таа истанца ја повикуваме функцијата startGame().

startGame()

Од оваа функција започнува извршувањето на играта. Напочетокот во неа се печати упатство за играње, се пополнува конфигурацијата на играта во која се одлучува како ќе се игра.

Доколку играта е помеѓу човек-алгоритам, програмата чека влез од корисникот. Овде се повикува посебна функција getMensInput() која има за цел да го земе влезот од корисникот.
Откако ќе се изврши таа функција (корисникот го поместил пионче) доаѓа ред на компјутерот. Се повикува функцијата artificialIntelligence() која ќе ја разгледаме подоцна.
По завршувањето на artificialIntelligence() се проверува дали на компјутерот и на човекот им останале пиони. Ако барем еден од нив има нула пиони, се печати соодветна порака и играта завршува.

Доколку играта е помеѓу алгоритам-алгоритам, тогаш како што беше споменато претходно, прв игра алгоритамот expectimax а, по него алгоритамот minimax. Најпрвин се повикува функцијата expectimaxInsteadOfPerson() каде што алгоритамот expectimax го прави првото поместување а, minimax повторно се повикува преку artificialIntelligence().
По завршувањето на artificialIntelligence() се проверува дали на expectimax и на minimax им останале пиони. Ако барем еден од нив има нула пиони, се печати соодветна порака и играта завршува.

getMensInput()

Најпрвин повикува друга функција со која ги проверува легалните потези на човекот. За легални потези се сметаат оние со кои човекот нема да излезе од таблата, нема да го помести пиончето на недозволена позиција или пак нема да ги поместува пионите на компјутерот. Потоа, се внесуваат индексите на пиончето кое корисникот сака да го помести, па се внесуваат индексите на полето на кое треба да се помести. Потоа проверувам колкава е разликата меѓу old_I, new_I (старта и новата редица на пиончето).
Ако разликата е еден се работи за обично поместување и во листата ги внесувам:

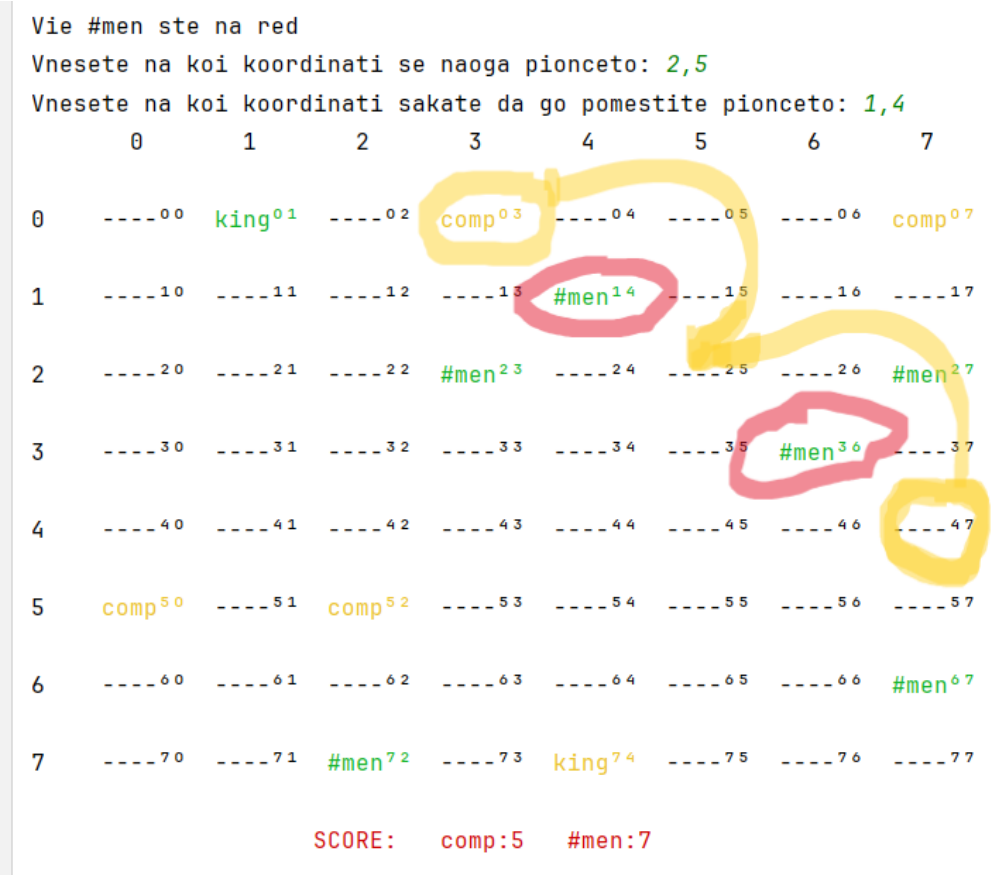
```
move = [old_I, old_J, new_I, new_J, 100, 100, 100, 100]
```

Каде што четирите стотки ми претставуваат координати на противникот. Во овој случај затоа што човекот не скока противник или противници координатите се 100.
Ако разликата е два се работи за скокање на еден противник и во листата ги внесувам:

```
move = [old_I, old_J, new_I, new_J, (old_I + new_I)//2, (old_J + new_J)//2, 100, 100]
```

Каде што последните 2 стотки ми претставуваат координати на вториот противник. Во превод вториот противник не постои.

Ако разликата е четири се работи за скокање на 2 противници. И овде е малку покомплицирано да ги најдеме кординатите на тие двајца противници. Затоа ја користиме функцијата defineAllPosibleOpponents() за да ни ги дефинира 16-те можни комбинации на координати на противниците. Доколку корисникот навистина внел легален влез, само една од овие 16 комбинации ќе биде точна. Ако корисникот внел нелегален влез тогаш функцијата ќе врати None.
Доколку движењето е дозволено, програмата го прави тоа движење и се повикува функција за пресметување на поените на играчите.



Слика 6: Скокање преку две противнички полиња (горна слика)
Слика 7: Изглед на таблата после скокањето (долна слика) [Сликите се превземени од симулација на играта]

Во оваа имплементација се дозволени поместувања на пиончето за едно поле на лево или на десно, скокање преку едно противничко поле на лево или на десно и скокање преку две противнички полиња на лево или на десно. Нормално пионите на човекот и компјутерот се движат во спротивна насока пред да станат кралеви што значи дека дозволена насока за движење на човекот е:
- од повисок индекс на редицата кон понизок индекс на редицата
а, дозволена насока за движење на компјутерот е:
- од понизок индекс на редицата кон повисок индекс на редицата

Редицата во која човекот станува крал е 0 а, редицата во која компјутерот станува крал е 7. (Слика 6 и 7)

artificialIntelligence()

Оваа функција се користи во два случаи.

Првиот случај, кога игра човек против алгоритам - заменува било кој од алгоритмите без разлика дали е minimax или expectimax.

Вториот случај, кога игра алгоритам против алгоритам - првиот потег го прави алгоритамот expectimax со функцијата expectimaxInsteadOfPerson() а, вториот потег го прави artificialIntelligence() заменувајќи го алгоритамот minimax.

Како работи artificialIntelligence() ?
Најпрвин прави длабока копија од моменталната табла на која се игра играта. Зошто е тоа така? За да креира објект од класата NewGameState со помош на таа табла.
Класата NewGameState има метод get_children().
Ќе ги испробува сите можни комбинации на таблата во функцијата get_children(). Ќе го земе секој можен потег на секое пионче на компјутерот и ќе направи нова табла. Затоа при правењето на таа нова табла многу е важно да не ја уништиме оригиналната па, правиме длабока копија. Доколку не направиме длабока копија туку при креирање на објект од NewGameState ја ставиме оригиналната табла тогаш ќе ја измениме неа многу пати без да се консултираме воопшто со алгоритмот minimax или expectimax. Во превод компјутерот ќе прави рандом потези без да размисли.

Го повикуваме методот get_children() од објектот кој го направивме. Сега ги имаме сите можни потези што може да ги направат сите пиони/кралеви во една листа. Листата содржи одбекти од типот NewGameState.

Најпрвин проверуваме дали оваа листа е празна односно дали компјутерот има услови да направи барем едно двежење. Доколку нема, компјутерот признава пораз.

Потоа во циклус интерираме низ сите објекти од тип NewGameState во листата. Го повикуваме алгоритмот minimax или expectimax кој враќа број. Тој број го ставаме во речник како клуч а, вредноста на клучот е објектот од типот NewGameState. Во превод правиме проценка на секое движење. Го оценуваме секое движење дали е добро за компјутерот или не.

```
for i in range(len(first_computer_moves)):
    child = first_computer_moves[i]
    value = GameCheckers.minimax(child.get_board(), depth: 1, -math.inf, math.inf, maximizing_player: False)
    dict[value] = child
```

Слика 8: Извадок од кодот за функцијата artificialIntelligence(). Оригиналниот код за овој прозорец е превземен од: https://github.com/dimitrijekaranfilovic/checkers.git Прилагодено за потребите на проектот.

На крајот го бараме клучот кој е **најголем** и ја земаме неговата вредност. Нормално вредноста е од тип NewGameState па од тој објект ја земаме таблата. Оваа табла ја ставаме да биде оригиналната тековна табла на класата.
На крајот повикуваме функција за пресметување на поените.

expectimaxInsteadOfPerson()

Оваа функција е слична како artificialIntelligence() со тоа што има една клучна разлика.

Оваа функција се користи само кога играат алгоритам против алгоритам. Функцијата expectimaxInsteadOfPerson() го заменува алгоритамот expectimax кој игра како прв (секое непарно поместување на таблата е направено поради оваа функција) против minimax кој игра како втор. (секое парно поместување)

А, artificialIntelligence() се користи и во двата случаи. И кога се игра алгоритам против алгоритам (го заменува minimax) и кога се игра алгоритам против човек. (го заменува minimax или expectimax соодветно)

Работи на 100% истиот принцип како artificialIntelligence().

MakeAMoveForMen() и MakeAMoveForComputer()

Овие методи сами по себе се многу слични. Едниот се користи за движење на човекот а, другиот на компјутерот. Како аргументи и двата методи ги примаат таблите, координатите на пиончето/кралот што треба да се помести, координатите на кој треба да се помнести, и 4 координати по две за секој од противниците. Доколку пиончето/кралот скока само еден

противник тогаш, последните две координати ќе бидат 100. Доколку пиончето/кралот прави обично движење без да скока противници тогаш последните 4 координати ќе бидат 100.

Како работат овие функции?
Најпрвин мора да кажеме дека секое движење што се прави во овие функции е легално двиќење. Зошто? Претходно пред да пристигнат сите овие аргументи на функцијата до самата функција има проверка на конзистентноста. Проверката се прави во две функции getMensAvailableMoves() и getCompAvailableMoves() соодветно. Таму се проверуваат движењата што ги имав споменато и погоре:

„Во оваа имплементација се дозволени поместувања на пиончето за едно поле на лево или на десно, скокање преку едно противничко поле на лево или на десно и скокање преку две противнички полиња на лево или на десно.“

Прво во овие функции се проверува дали се работи за крал или за пион. Затоа што доколку се работи за пион можеби во ова движење пионот треба да стане крал. (Нормално доколку со поместувањето се наоѓа во соодветната редница)
Потоа се проверува дали прави обично движење, скокање преку еден или преку два. Врши промена на таблата соодветно.

CheckPlayerMoves(), checkComputerMoves(), checkPlayerJump(), checkComputerJump(), checkPlayerDoubleJump() и checkComputerDoubleJump()

Овие методи генерално се методи за конзистентност на движењата. Прават проверка за апсолутно секое движење дали е дозволено или не. Дел од нив се користат за проверка на движењата на компјутерот а, дел на човекот.

NewGameState: get_children()

Оваа функција служи за генерирање на сите можни состојби на таблата. Во главно таа се повикува во 4 функции: artificialIntelligence, expectimaxInsteadOfPerson, minimax, expectimax.

Алгоритам-човек
Најпрво се повикува во artificialIntelligence() а, потоа ја повикуваат и самите minimax и expectimax алгоритми.
Кога artificialIntelligence() ја повикува оваа функција ја повикува за следниот потег. Односно на ред дошол компјутерот и се разгледуваат сите можни движења на тековните пиони и кралеви. Додека кога ја повикуваат minimax или expectimax алгоритмите ја повикуваат длабочински. Доколку бил компјутерот на ред кои пиони може да ги помести, па кога ќе ги помести тие пиони кои пиони човекот може да ги помести, па кога ќе ги помести човекот неговите пиони повторно, кои пиони компјутерот да ги помести... Во minimax/expectimax оваа функција се повикува за да гледаме длабочински додека во artificialIntelligence() ја повикуваме затоа што компјутерот е на ред и сакаме да видиме кои пиони тековно можат да се поместат сега моментално, без да размислуваме што понатаму.

Алгоритам-алгоритам
Најпрво се повикува во expectimaxInsteadOfPerson() а, оттаму во expectimax() функцијата. Работи на истиот принцип како artificialIntelligence() со двата алгоритми. Expectimax ја повикува длабочински а, expectimaxInsteadOfPerson() ја повикува само еднаш за да ги види сите можни потези за моменталниот изглед на таблата без да размислува што потоа.
Кога ќе заврши expectimax и на ред ќе дојде minimax се повикува во artificialIntelligence() кои во овој случај го претставува само minimax алгоритамот и акциите се одвиваат на истиот начин опишан погоре.

Зошто некој методи се статички а, некој не?

При истражување на овој проблем на интернет посебно внимание ми остави тоа што во различни имплементации различни методи беа статички. Една од причините за тоа е што некои методи мора да се повикуваат надвор од класата GameCheckers. Во тој случај имаме 2 опции:

- Да креираме истанци од оваа класа
- Да ги направиме тие методи статички

Ако направиме нови истанци од GameCheckers тоа може да биде фатално. Затоа што ние немаме повеќе игри „Дама“. Ние во еден момент играме само една игра „Дама“. Ако направиме истанци ќе постојат многу повеќе такви и всушност сите поместувања на пионите/кралевите ќе бидат на многу табли наместо на една. Затоа мора да користиме статички методи.

best_heuristics()

Хевристиката best_heuristics е најкомплексната и ги зема предвид сите клучни аспекти на играта дама. Таа ги оценува бројот на фигури, типот на фигурите (обични или кралеви), нивната позиција на таблата (на пример фигурите на работ од таблата се постабилни) и можностите за напад и одбрана. Оваа хевристика им помага на алгоритмите како Expectimax или Minimax да донесуваат најдобри можни одлуки, земајќи ги предвид сите важни фактори кои влијаат на исходот на играта.

almostKing()

Оваа функција ја претставува втора хевристика на играта „Дама“: „almostKing“.

Хевристиката almostKing ги оценува позициите на фигурите на таблата, давајќи поголема важност на фигурите кои се блиску до можноста да станат кралеви. Исто така, таа го зема предвид и бројот на фигури на компјутерот наспроти противникот, што е клучно за стратегијата во дама. Оваа хевристика им помага на алгоритмите како Expectimax или Minimax да донесуваат подобри одлуки врз основа на позицијата и бројот на фигури.

edgePosition()

Оваа функција ја претставува трета хевристика на играта „Дама“: „edgePosition“.

Хевристиката edgePosition ги оценува позициите на фигурите на работ на таблата, сметајќи ги фигурите на работ се помалку стабилни за компјутерот (како минимизирачки играч) и постабилни за противникот (како максимизирачки играч). Исто така, таа го зема предвид и бројот на фигури на компјутерот наспроти противникот, што е клучно за стратегијата во дама. Оваа хевристика им помага на алгоритмите како Expectimax или Minimax да донесуваат подобри одлуки врз основа на позицијата на фигурите и бројот на фигури.

Симулации на играта и нивни статистики

Со цел да ги покажеме перформансите и резултатите на алгоритмите, направивме најразлични симулации и тестирања на истите. Детален преглед на симулациите можете да видите на следните линкови:

- Човек наспроти expectimax со хевристика 1
- Човек наспроти expectimax со хевристика 2
- Човек наспроти expectimax со хевристика 3
- Човек наспроти minimax со хевристика 1
- Човек наспроти minimax со хевристика 2
- Човек наспроти minimax со хевристика 3
- expectimax со хевристика 1 наспроти minimax со хевристика 1
- expectimax со хевристика 2 наспроти minimax со хевристика 1
- expectimax со хевристика 3 наспроти minimax со хевристика 1
- expectimax со хевристика 1 наспроти minimax со хевристика 2
- expectimax со хевристика 2 наспроти minimax со хевристика 2
- expectimax со хевристика 3 наспроти minimax со хевристика 2
- expectimax со хевристика 1 наспроти minimax со хевристика 3
- expectimax со хевристика 2 наспроти minimax со хевристика 3
- expectimax со хевристика 3 наспроти minimax со хевристика 3

Забелешка: За поголема визуелизација на симулациите, најдобро е кодот од myProject.py да се стартува во рамките на самиод PyCharm се со цел да се прикажат боите на екранот. На тој начин симулациите ќе бидат многу попрегледни и многу повоочливи за самото око.

Со цел да ги сумаризираме резултатите од симулациите погоре можеме да направиме соодветни табели. Првата ќе биде посебно за перформансите на minimax алгоритамот а, втората посебно за перформансите на expectimax алгоритамот. (Слика 9 и слика 10)

Во првата табела според описот на редиците можеме да видиме дека minimax најпрвин се натпрварува со човекот испробувајќи ги сите 3 хевристики, потоа се натпреварува со expectimax комбинирајќи ги сите 3 негови хевристики со хевристиките на expectimax.
Табелата содржи податоци за тоа дали е победа или пораз на minimax (0 означува пораз додека 1 победа), освоените поени на minimax (односно колку противнички човечиња отстранил од таблата), поените на

противникот (односно колку пиончиња противникот успеал да му земе на minimax), бројот на цртежи (колку пати се сменила таблата поради поместувањата од почетната состојба на играта до крајниот исход) и времетраење (вкупното време што му било потребно на алгоритамот minimax за да ги направи сите негови поместувања. Во времетраењето се смета исклучиво само времето на алгоритамот без времето на противникот)

	id	win(0-no, 1-yes)	minimax_socre	opponent_score	draws	duration(sec)
0	human_minimax1	1	12	9	79	3.021
1	human_minimax2	1	12	6	101	5.201
2	human_minimax3	1	10	7	87	6.864
3	minimax1_expectimax1	1	11	2	39	2.395
4	minimax1_expectimax2	1	12	1	31	1.865
5	minimax1_expectimax3	1	11	1	37	2.835
6	minimax2_expectimax1	1	12	1	35	3.089
7	minimax2_expectimax2	1	11	0	43	3.110
8	minimax2_expectimax3	1	12	1	35	3.179
9	minimax3_expectimax1	1	12	1	41	4.089
10	minimax3_expectimax2	1	12	1	41	3.811
11	minimax3_expectimax3	1	10	1	37	4.254

*Слика 9: Табела со резултати на **minimax** алгоритамот наспроти противници од најразличен карактер (човек, друг алгоритам) [Табелата е генерирана од статистики на симулациите во проектот.]*

Во втората табела дел од податоците се повторуваат со првата табела. Првите 3 редици ни опишуваат како алгоритамот expectimax се справува со човекот и овие редици се уникатни. Додека пак следните 9 редици содржат податоци кои се исти со табелата на minimax но, во корист на expectimax. Овојпат колоната „win(0-no, 1-yes) “ означува дали е победа или пораз на expectimax, потоа ги имаме освоените поени на expectimax од секоја симулација соодветно па, ги имаме противничките поени, бројот на цртежи кои во последните 9 редици целосно се совпаѓа со првата табела и

вкупното време на expectimax за да ги направи сите свои потези од почетокот на играта до крајот.

	id	win(0-no, 1-yes)	expectimax_score	opponent_score	draws	duration(sec)
0	human_expectimax1	0	9	12	90	4.706
1	human_expectimax2	0	7	12	48	4.962
2	human_expectimax3	1	12	8	51	4.199
3	minimax1_expectimax1	0	2	11	39	2.065
4	minimax1_expectimax2	0	1	12	31	2.105
5	minimax1_expectimax3	0	1	11	37	2.812
6	minimax2_expectimax1	0	1	12	35	1.597
7	minimax2_expectimax2	0	0	11	43	2.504
8	minimax2_expectimax3	0	1	12	35	2.267
9	minimax3_expectimax1	0	1	12	41	2.332
10	minimax3_expectimax2	0	1	12	41	2.959
11	minimax3_expectimax3	0	1	10	37	3.119

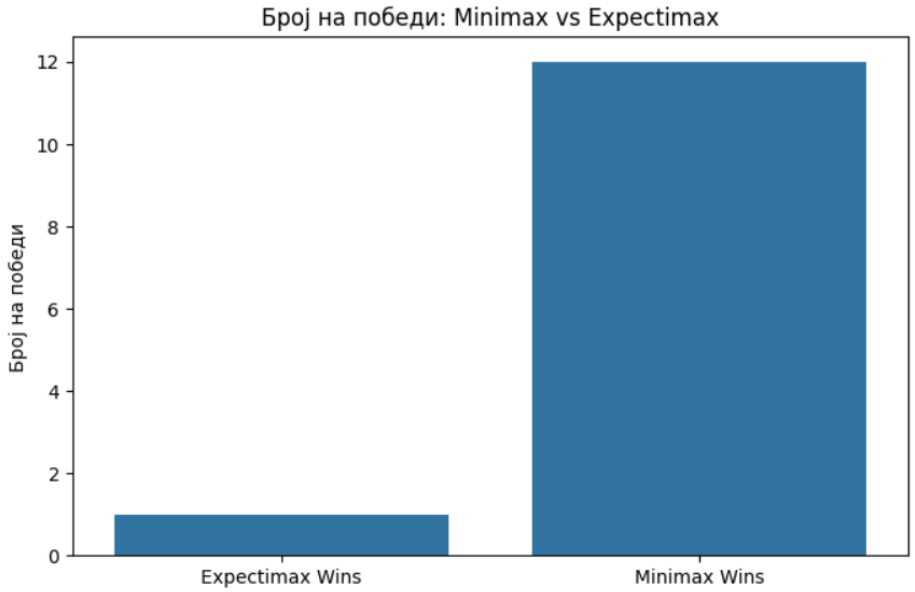
*Слика 10: Табела со резултати на **expectimax** алгоритамот наспроти противници од најразличен карактер (човек, друг алгоритам) [Табелата е генерирана од статистики на симулациите во проектот.]*

Од дадените табели можеме да изведеме голем број заклучоци за тоа кој алгоритам е подобар, кој алгоритам носи побрзи одлуки, колку човекот е супериорен во однос на алгоритмите како и дали постојат карактеристики (атрибути) од колоните кои се клучни за исходот на алгоритмите.

За таа цел ќе нацртаме неколку фигури за да направиме споредба. На слика 11 јасно може да се забележи дека креираме хистограм со цел да видиме кои алгоритам има подобри резултати.

Според податоците алгоритамот minimax има водечки резултати во споредба со expectimax. Minimax успеал да освои 12 победи од вкупно 12 симулации додека пак expectimax успеал да освои само една победа од 12 симулации.

```
#Histogram za broj na pobedi
plt.figure(figsize=(8,5))
sns.barplot(x=['Expectimax Wins', 'Minimax Wins'], y=[expectimaxStatistic['win(0-no, 1-yes)'].sum(), minimaxStatistic['win(0-no, 1-yes)'].sum()])
plt.ylabel("Број на победи")
plt.title("Број на победи: Minimax vs Expectimax")

plt.show()
```

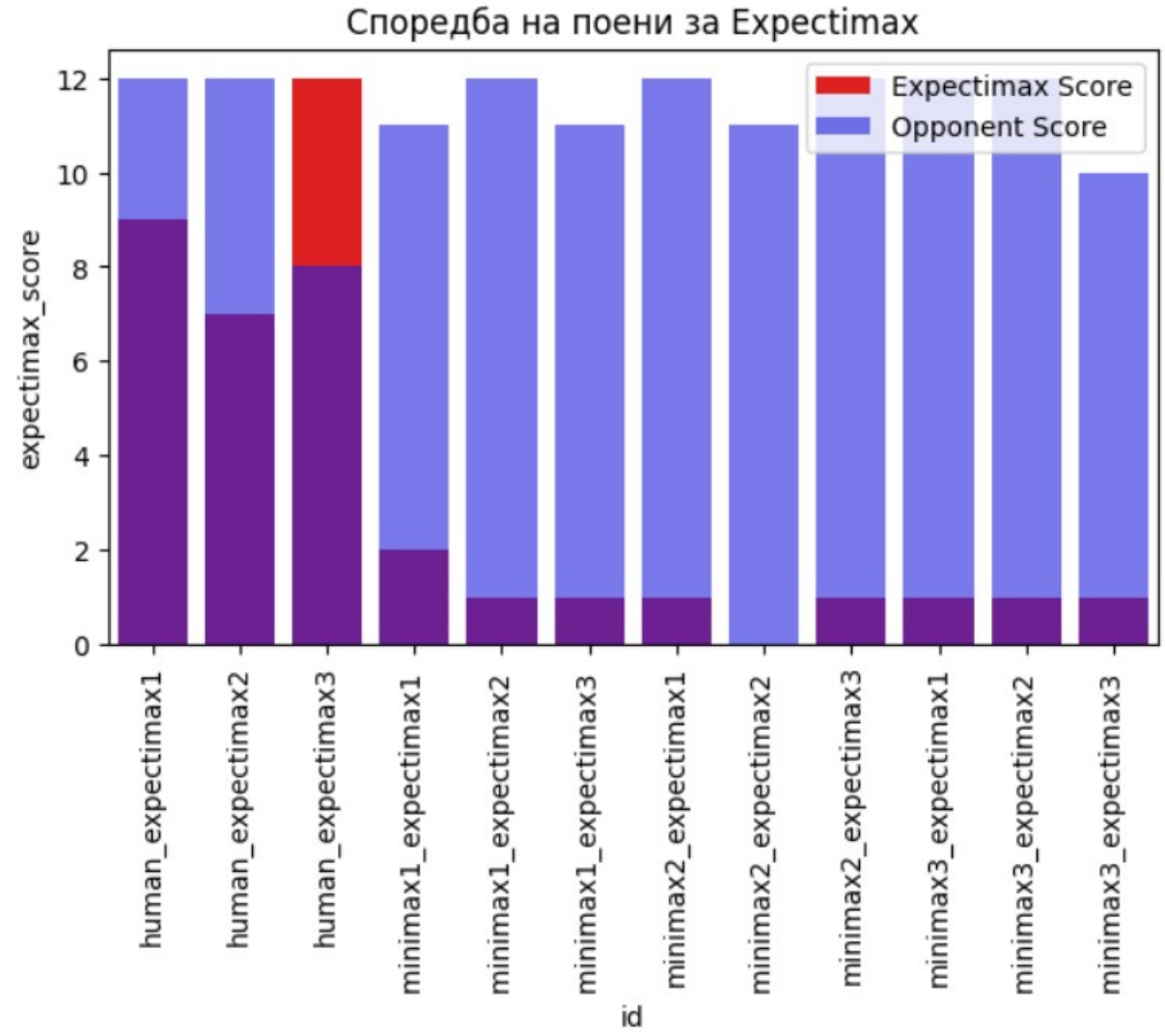


Слика 11: Хистограм за бројот на победи на двата алгоритми [Хистограмот е креиран за потребите на проектот врз база на податоците од симулациите на играта]

Сега ќе направиме уште една слична статистика прво за да ги видиме освоените поени на expectimax во однос на противниците па, освоените поени на minimax во однос на противниците. (слика 12, 13, 14, 15)

```
# Bar chart za sporedba na postignati poeni na exceptimax vo odnos na protivnicite
plt.figure(figsize=(7, 4))
sns.barplot(x='id', y='expectimax_score', data=expectimaxStatistic, color='red', label='Expectimax Score')
sns.barplot(x='id', y='opponent_score', data=expectimaxStatistic, color='blue', alpha=0.6, label='Opponent Score')
plt.xticks(rotation=90)
plt.legend()
plt.title("Споредба на поени за Expectimax")
plt.show()
```

*Слика 12: Код од Bar chart за споредба на постигнатите поени на **expectimax** во однос на противниците (Кодот е генериран специјално за потребите на проектот)*



*Слика 13: Bar chart график за визуелизација на освоените поени на алгоритамот **expectimax** и освоените поени на противниците*

Од слика 13 може да забележиме дека со црвено е означен просторот со освоени поени само на expectimax. Со сина боја е означен просторот со освоени поени само на противникот додека пак, со виолетова боја е означен просторот со освоени поени на алгоритамот и противникот истовремено.
Од овој график можеме да забележиме дека expectimax освојува значително помалку поени во однос на противниците. Водство има само кога игра против човекот со третата хевристика.

Исто така од сликата јасно може да се забележи дека expectimax освојува повеќе поени кога игра против човекот наспроти тоа кога игра против minimax алгоритмот. Дури и се доведува во ситуација да неможе да освои ниту еден поен во случајот кога expectimax игра со хевристика број 2 против minimax со хевристика број 2.

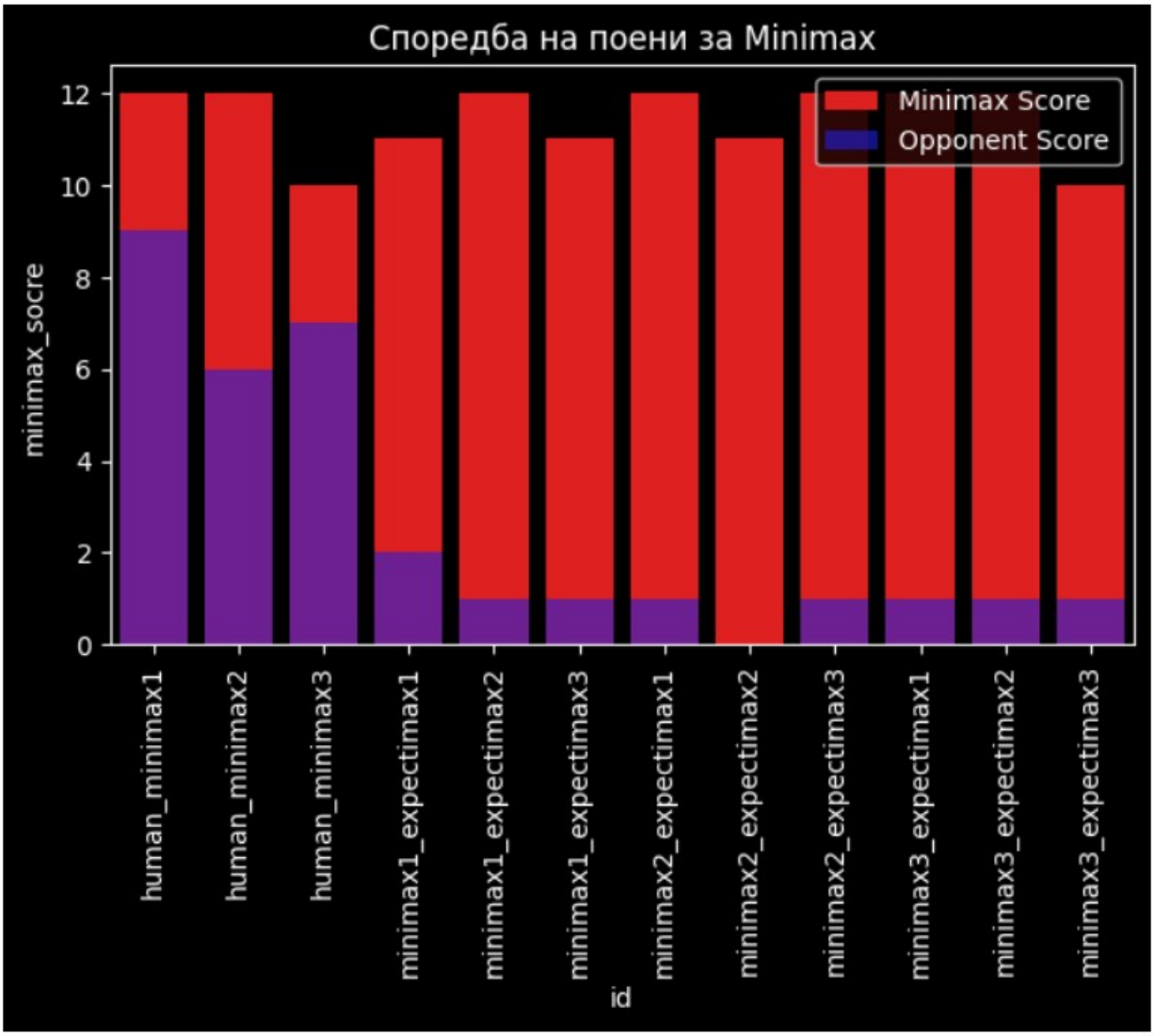
На следните слики (слика 14 и 15) е прикажан кодот за графикот за освоените поени на minimax во однос на противниците како и самиот график.

```
# Bar chart za sporedba na postignati poeni na minimax vo odnos na protivnicite
plt.figure(figsize=(7,4))
sns.barplot(x='id', y='minimax_socre', data=minimaxStatistic, color='red', label='Minimax Score')
sns.barplot(x='id', y='opponent_score', data=minimaxStatistic, color='blue', alpha=0.6, label='Opponent Score')
plt.xticks(rotation=90)
plt.legend()
plt.title("Споредба на поени за Minimax")
plt.show()
```

*Слика 14: Код од Bar chart за споредба на постигнатите поени на **minimax** во однос на противниците (Кодот е генериран специјално за потребите на проектот)*

Од слика 15 може да забележиме дека со црвено е означен просторот со освоени поени само на minimax. Со сина боја е означен просторот со освоени поени само на противникот додека пак, со виолетова боја е означен просторот со освоени поени на алгоритамот и противникот истовремено.
Од графикот можеме да забележиме дека бројот на освоени поени на minimax е голем. Дури во неколку случаи minimax ги освојува сите поени. За разлика од претходниот график во кои се појавуваат сите 3 бои овој график нема простор за сина боја. Причината за ова непојавување на синаата боја е што нема ниту еден случај во кои противникот има повеќе поени од minimax што го прави овој алгоритам совршен за оваа игра.

Исто така може да се забележи континуираната промена на виолетовата боја од која што јасно можеме да видиме колку поени освоил противникот. Поголема предност во улога на противник наоѓаме кај човекот додека пак, expectimax навистина покажува сериозен неуспех.



*Слика 15: Bar chart график за визуелизација на освоените поени на алгоритамот **minimax** и освоените поени на противниците*

Следните 2 графици се однесуваат на времетраењето на алгоритмите. Подолу ни се дадени кодови за графици кои ги опишуваат промените во времетраењето на алгоритмите за различни противници. (слики 16, 17, 18 и 19)

Најпрвин е даден кодот и линискиот дијаграм на expectimax а, потоа на minimax.

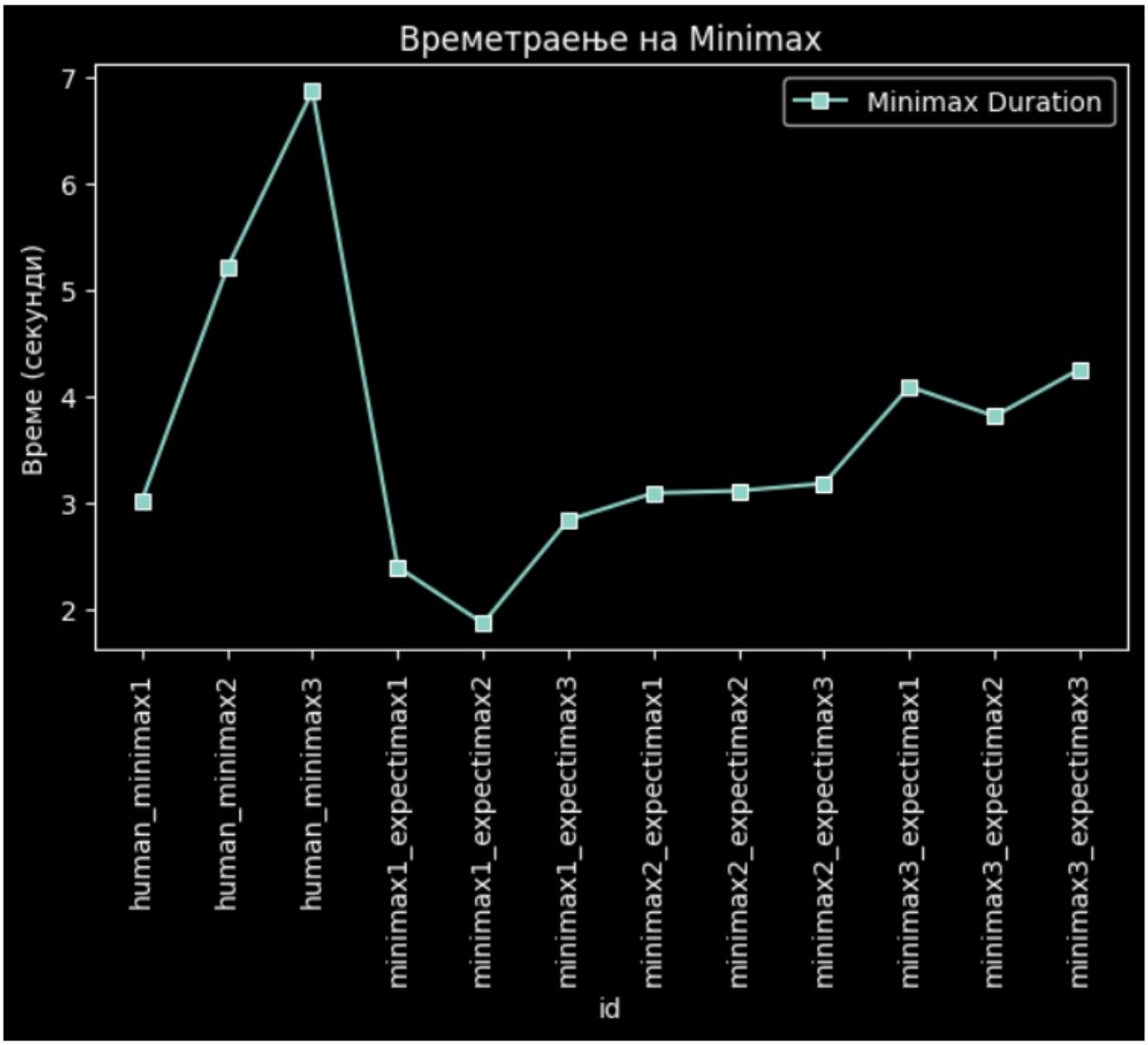
```
#Line plot za vremetraenje na partiite
plt.figure(figsize=(10, 5))
sns.lineplot(x='id', y='duration(sec)', data=expectimaxStatistic, marker='o', label='Expectimax Duration')
plt.xticks(rotation=90)
plt.title("Времетраење на Expectimax")
plt.ylabel("Време (секунди)")
plt.legend()
plt.show()
```

*Слика 16: Код за линиски дијаграм со кој се опишува текот на времетраењето на **expectimax** во зависност од противниците (Кодот е генериран специјално за потребите на проектот)*

*Слика 17: Линиски дијаграм со кој се опишува текот на времетраењето на **expectimax** во зависност од противниците*

```
#Line plot za vremetraenje na partiite
plt.figure(figsize=(7, 4))
sns.lineplot(x='id', y='duration(sec)', data=minimaxStatistic, marker='s', label='Minimax Duration')
plt.xticks(rotation=90)
plt.title("Времетраење на Minimax")
plt.ylabel("Време (секунди)")
plt.legend()
plt.show()
```

*Слика 18: Код за линиски дијаграм со кој се опишува текот на времетраењето на **minimax** во зависност од противниците (Кодот е генериран специјално за потребите на проектот)*



*Слика 19: Линиски дијаграм со кој се опишува текот на времетраењето на **minimax** во зависност од противниците*

Од линиските дијаграми за двата алгоритми можеме да заклучиме дека тие трошат многу повеќе време на човекот како противник отколку на другиот алгоритам. Исто така можеме да видиме дека minimax троши повеќе време за одлучување од expectimax. (слика 20) На пример, во последната игра (minimax со хевристика 3 и expectimax со хевристика 3) вкупното време на одлучување на minimax за сите негови потези изнесува околу 4 секунди додека пак на expectimax околу 3,2 секунди.

```
averageDurationMinimax = minimaxStatistic['duration(sec)'].mean()
averageDurationExpectimax = expectimaxStatistic['duration(sec)'].mean()
print("Prosecnoto vreme na minimax e: ", averageDurationMinimax)
print("Prosecnoto vreme na expectimax e: ", averageDurationExpectimax)
```

```
Prosecnoto vreme na minimax e:  3.64275
Prosecnoto vreme na expectimax e:  2.968916666666667
```

Слика 20: Просечното време на minimax за одлучување е многу поголемо од тоа на expectimax

Претпоследниот график кои го правиме од симулациите е поврзан со хевристичките функции односно ќе видиме дали и колку влијаат врз освоените поени. Најпрвин формираме две табели со податоци. Едната за expectimax алгоритамот а, другата за minimax алгоритамот соодветно. Секоја од нив содржи 2 колони од кои едната е описот на симулацијата (кого против кого), додека пак другата е освоените поени на алгоритамот во таа симулација.
По креирањето на двете табели додаваме 2 нови колони во истите. Вредностите во новите колони се всушност редните броеви на хевристиките кои ги користат алгоритмите. Нив ги добиваме од колоната за описот на симулацијата и доаѓаат веднаш после името на алгоритамот. (слика 21)

```
data_expectimax = {
    "id": ["human_expectimax1", "human_expectimax2", "human_expectimax3",
           "minimax1_expectimax1", "minimax1_expectimax2", "minimax1_expectimax3",
           "minimax2_expectimax1", "minimax2_expectimax2", "minimax2_expectimax3",
           "minimax3_expectimax1", "minimax3_expectimax2", "minimax3_expectimax3"],
    "expectimax_score": [9, 7, 12, 2, 1, 1, 1, 0, 1, 1, 1, 1]
}

# Податоци за Minimax
data_minimax = {
    "id": ["human_minimax1", "human_minimax2", "human_minimax3",
           "minimax1_expectimax1", "minimax1_expectimax2", "minimax1_expectimax3",
           "minimax2_expectimax1", "minimax2_expectimax2", "minimax2_expectimax3",
           "minimax3_expectimax1", "minimax3_expectimax2", "minimax3_expectimax3"],
    "minimax_score": [12, 12, 10, 11, 12, 11, 12, 11, 12, 12, 12, 10]
}

df_expectimax = pd.DataFrame(data_expectimax)
df_minimax = pd.DataFrame(data_minimax)
```

```
df_expectimax["expectimax_heuristic"] = df_expectimax["id"].str.extract(r'expectimax(\d+)').astype(int)
df_minimax["minimax_heuristic"] = df_minimax["id"].str.extract(r'minimax(\d+)').astype(int)
```

Слика 21: Креирање на соодветни табели со цел споредба на хевристичките функции. (Кодот е генериран специјално за потребите на проектот)

```
[46] scores_by_expectimax_heuristic = df_expectimax.groupby("expectimax_heuristic")["expectimax_score"].sum()
     scores_by_minimax_heuristic = df_minimax.groupby("minimax_heuristic")["minimax_score"].sum()
```

```
[47] scores_by_minimax_heuristic
```

minimax_heuristic	minimax_score
1	46
2	47
3	44

dtype: int64

```
[48] scores_by_expectimax_heuristic
```

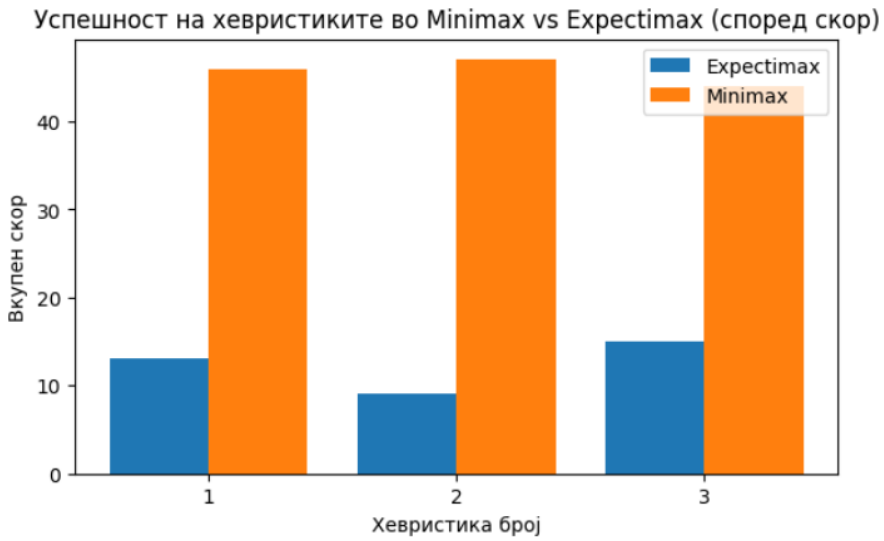
expectimax_heuristic	expectimax_score
1	13
2	9
3	15

dtype: int64

Слика 22: Код за групирање на соодветните хевристики и пресметување на нивните освоени поени. (Кодот е генериран специјално за потребите на проектот)

Потоа откако ги креиравме и пополнивме двете табели, правиме групирање според соодветната хевристика и ги пресметуваме освоените поени. (слика 22)

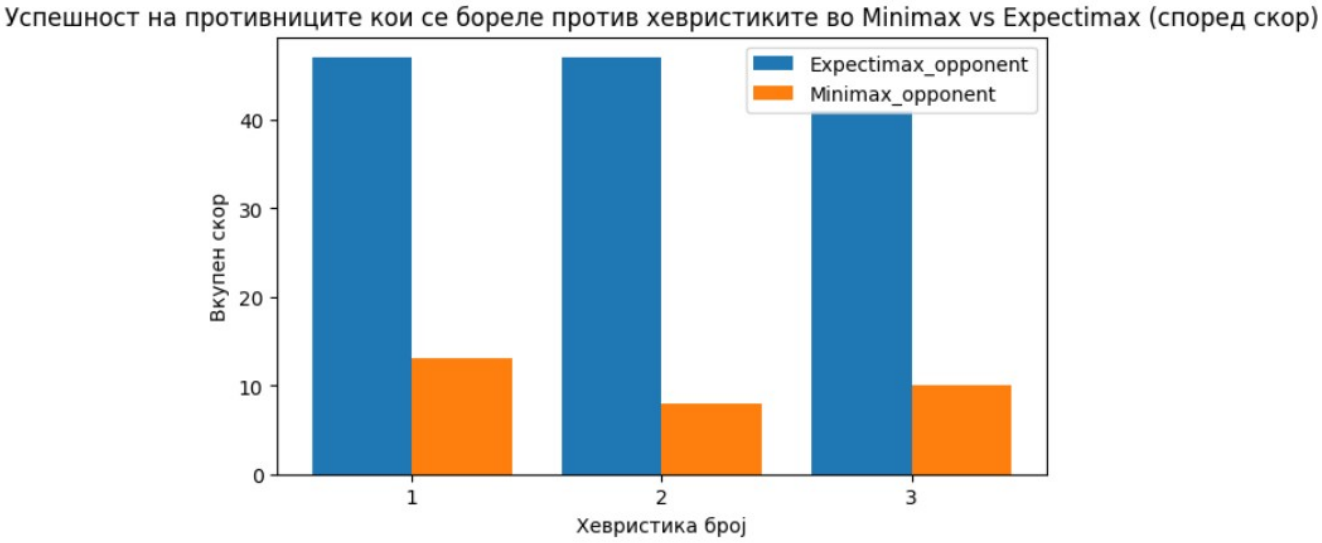
```
plt.figure(figsize=(7, 4))
plt.bar(scores_by_expectimax_heuristic.index - 0.2, scores_by_expectimax_heuristic.values, width=0.4, label="Expectimax")
plt.bar(scores_by_minimax_heuristic.index + 0.2, scores_by_minimax_heuristic.values, width=0.4, label="Minimax")

plt.xlabel("Хевристика број")
plt.ylabel("Вкупен скор")
plt.xticks(range(1, int(max(scores_by_expectimax_heuristic.index.max(), scores_by_minimax_heuristic.index.max())) + 1))
plt.legend()
plt.title("Успешност на хевристиките во Minimax vs Expectimax (според скор)")
plt.show()
```



Слика 23: Код за цртање на графикот со цел визуелизација на успехот на хевристиките. (Кодот е генериран специјално за потребите на проектот)

Според цртежот на слика 23 можеме да го видиме успехот на хевристиките за двата алгоритми посебно. Алгоритамот minimax има благо водство во освоени поени кај втората хевристика наспроти другите. Меѓутоа алгоритамот expectimax има најслаби резултати кај втората хевристика и највисоки кај третата.
За подобра споредба на квалитетот на хевристиките ние направивме уште еден график кои ги покажува поените на противникот за секоја хевристика посебно. Овој график може да се види на слика 24 а, кодот за него ќе биде достапен во следниот документ. Според графикот можеме да забележиме

дека без разлика која и да е хевристиката противникот на expectimax секојаш има високи резултати. Противниците на minimax најмногу се издвојуваат кај првата хевристика каде имаат најдобри резултати а, најслабите резултати им се кај втората хевристика.

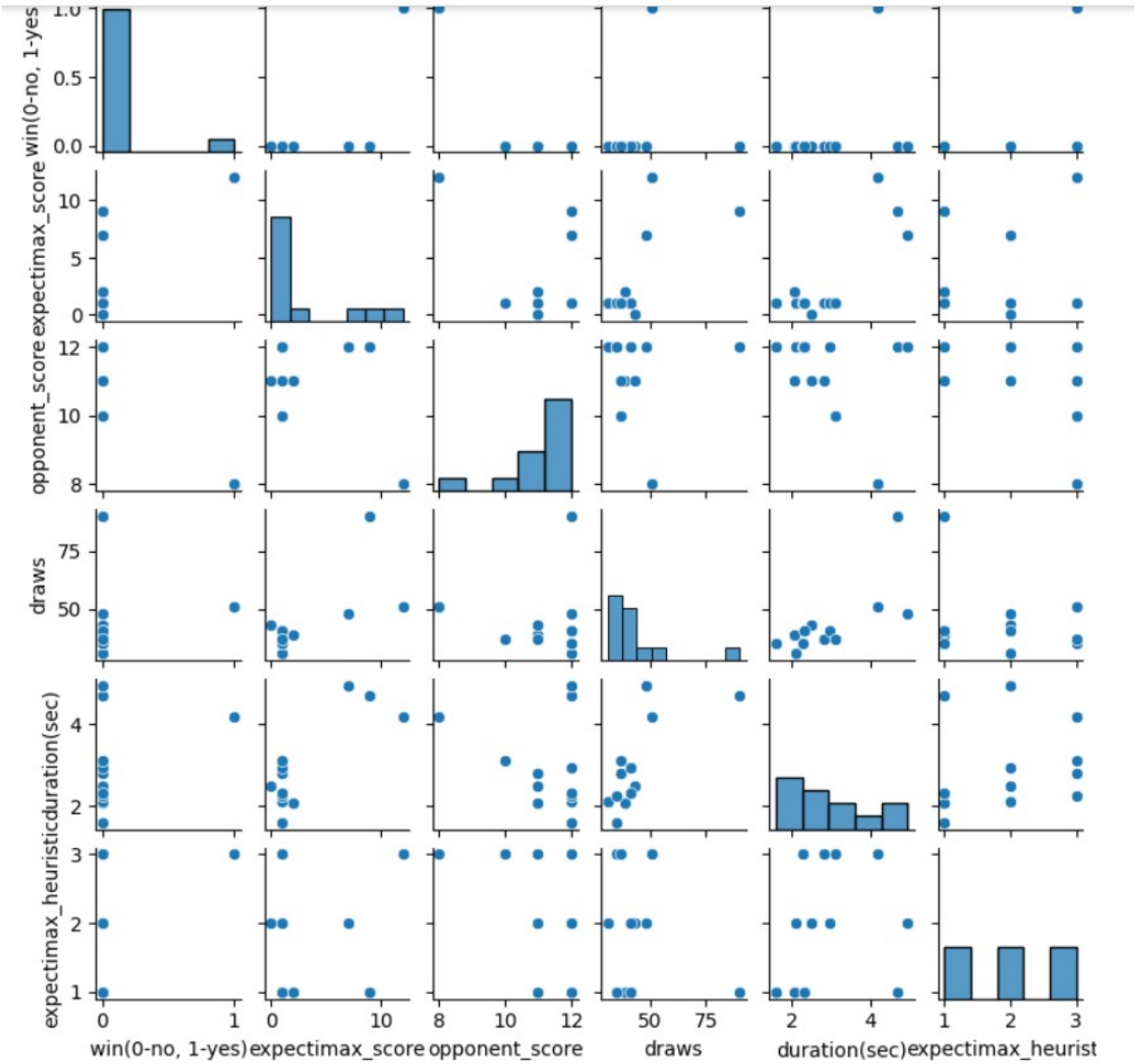


Слика 24: Код за цртање на графикот со цел визуелизација на успехот на противниците (Кодот е генериран специјално за потребите на проектот)

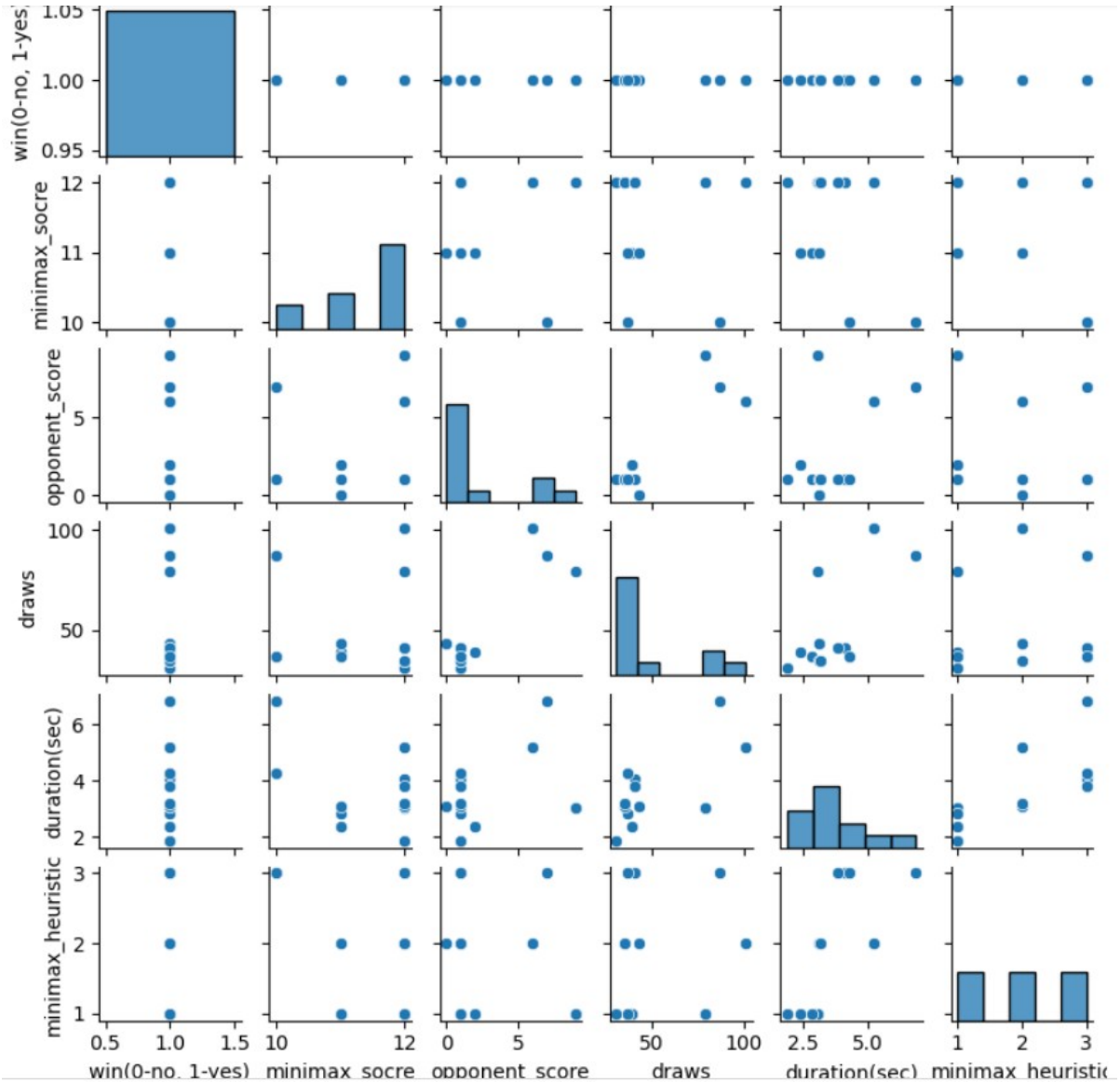
Како едни од последните графици кои ќе ги конструираме за да направиме споредба за нашите симулации се графиците за зависности на двата алгоритми посебно. Преку нив ќе видиме од кои карактеристики најмногу зависи дали алгоритмите ќе победат или не.

Според слика број 25 на која е претставен графикот на зависности на алгоритамот **expectimax** можеме да заклучиме дека бројот на поените кои ќе ги освои алгоритамот најмногу зависат од времетраењето на играта.

За разлика од expectimax, **minimax** има малку поинаков график на зависности. Времетраењето на играта и хевристиката на алгоритамот се многу тесно поврзани. Исто така има одредена поврзаност и меѓу бројот на цртежи и времетраењето како и меѓу бројот на цртежи и поените на противникот. Споредено со графикот на expectimax овде немаме голема поврзаност меѓу бројот на освоените поени на алгоритамот и другите карактеристики. (слика 26)



*Слика 25: График на зависности од карактеристики на алгоритамот **expectimax** (Графикот е генериран специјално за потребите на проектот)*



*Слика 26: График на зависности од карактеристики на алгоритамот **minimax** (Графикот е генериран специјално за потребите на проектот)*

Заклучок

Овој проект беше фокусиран на имплементација на вештачка интелигенција за играта дама (checkers), користејќи ги алгоритмите Minimax и Expectimax. Иако двата алгоритми се моќни алатки за донесување одлуки во игри, за играта дама, Minimax се покажа како подобар избор. Ова е поради фактот што дама е детерминистичка игра, каде што сите потези и исходи се предвидливи и нема елемент на несигурност или случајност. Minimax алгоритмот е дизајниран токму за вакви игри, каде што секој потег е резултат на логика и стратегија, а не на случајни настани.

Од друга страна, Expectimax е подобар за игри кои вклучуваат елемент на несигурност, како што се игрите со коцки или карти. Бидејќи во дама нема такви елементи, Expectimax не може да ги искористи своите предности, како што е пресметката на очекувани вредности врз основа на веројатности. Наместо тоа, Minimax алгоритмот, со својата способност да ги истражува сите можни потези и да ги избере оние со најдобар исход, е идеален за дама.

Во проектот беа имплементирани и хевристики за подобро оценување на состојбите на таблата. Овие хевристики ги земаа предвид факторите како бројот на фигури, нивната позиција, можноста за напад и одбрана, како и важноста на кралевите. Ова му овозможи на Minimax алгоритмот да донесува поинформирани и стратешки подобри одлуки.

Како заклучок, Minimax алгоритмот, заедно со добро дизајнираните хевристики, се покажа како најефикасно решение за играта дама. Expectimax, иако моќен, не е соодветен за оваа игра поради отсуството на несигурност и случајност. Овој проект ја истакна важноста на изборот на соодветен алгоритам и хевристики за решавање на специфични проблеми во вештачката интелигенција.

Референци:

Извадок од книга на Расел и Норвиг

Github Checkers Game Example youtube video one

Expectimax Algoritam Code Implementation youtube video two

Data for minimax Data for expectimax

simulations of the checkers game Data graphic and statistic in Colab