УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Александра Ж. Никшић

АЛГОРИТМИ ЗА УПАРИВАЊЕ КОРИСНИКА У ВИДЕО ИГРАМА

мастер рад

Београд, 2023.

Ментор:

др Александар Картељ, доцент
Универзитет у Београду, Математички факултет

Чланови комисије:

др Владимир Филиповић, редовни професор
Универзитет у Београду, Математички факултет

др Милан Банковић, доцент
Универзитет у Београду, Математички факултет

Датум одбране:

Наслов мастер рада: Алгоритми за упаривање корисника у видео играма

Резиме: Начин упаривања корисника у видео играма на интернету има значајне последице на задовољство корисника. Тренутне стратегије за упаривање имају једну кључну идеју: креирање равноправних партија при сваком упаривању. Под претпоставком да равноправна партија пружа највеће задовољство играчу, ови системи упарују играче са приближно истим способностима. Показало се да ова претпоставка не мора увек да буде тачна, јер су ангажованост корисника и време чекања на упаривање такође важни. Циљ овог рада је конструкција различитих стратегија за упаривање играча и демонстрација њиховог рада. Неке од стратегија које ће бити размотрене су *FIFO* (енг. first-in-first-out), стратегије засноване на нумеричким оценама и карактеристикама играча. Симулација упаривања ће бити тестирана за различите комбинације контролних параметара. Упоређивање стратегија ће се вршити на основу групне оцене задовољства свих играча који су учествовали у свим одиграним партијама. Задовољство ће у овом случају бити посматрано кроз тенденцију играча да остану ангажовани у игри. У оквиру рада биће имплементирана апликација за симулацију упаривања чији су саставни делови: база података, серверска апликација видео игре и клијентска компонента путем које ће корисник приступати апликацији кроз веб прегледач.

Кључне речи: упаривање, редови, графови, вештина, оптимизација

Садржај

Глава 1

Увод

1.1 Опис проблема упаривања играча у видео играма

Упаривање (енг. *matchmaking*) је процес повезивања играча у видео играма на интернету ради формирања мечева. Тај процес може користити различите алгоритме и методе како би идентификовао одговарајуће играче и омогућио им жељено повезивање. Циљ упаривања је стварање равноправних мечева који пружају изазовно искуство свим учесницима.

Да би се остварило успешно упаривање, обично се узимају у обзир фактори као што су вештине, ниво искуства или ранг играча. Алгоритми упаривања анализирају ове факторе како би пронашли најбоље могуће подударање између играча. Циљ је створити мечеве у којима ће сви учесници имати сличне шансе за победу, што доводи до равноправности и задовољства свих играча.

Упаривање такође може узети у обзир и друге факторе, попут географске локације играча, језика или опредељења према стиловима играња. Ово помаже у стварању парова са сличним интересима, олакшава комуникацију и унапређује игру.

Важно је напоменути да свака игра може имати свој јединствени систем упаривања, прилагођен специфичним потребама и циљевима те игре.

Упаривање корисника има неколико важних мотива, укључујући проналажење играча сличних вештина и интереса, повећање укупног квалитета игре и уживања играча. Принцип упаривања корисника обично се базира на систему оцењивања или рангирања играча. Сваки корисник има свој профил

који садржи податке о његовом нивоу вештина, његовим постигнућима или другим релевантним информацијама. Алгоритми упаривања користе те информације како би пронашли одговарајуће партнере за играче у истом или сличном распону вештина.

Један од изазова код упаривања корисника представља проналажење равнотеже између чекања на упаривање и проналажења играча с одговарајућим нивоом вештина. Пребрзо упаривање може резултирати неравноправним утакмицама, док предуго чекање може фрустрирати играче.

Упаривање корисника може користити параметре као што су географска близина, језичке преференције или преференције играча за одређене стилове играња. На пример, у неким играма преферира се упаривање играча који користе сличне тактике или уживају у истим врстама игара [28].

У неким случајевима, налажење одговарајућих супарника може бити посебно важно, као на пример у електронском спорту (енг. *e-Sports*), у ком се играчи рангирају према својим постигнућима. У тим ситуацијама би требало користити алгоритме који би осигурали што већу равнотежу између тимова или појединаца, како би се обезбедио фер и изазован меч.

О значају ове теме говоре бројне популарне и широко распрострањене игре као што су *Hearthstone, League of Legends, Counter Strike* и *Fortnite*.

У овом раду ће фокус бити на мечевима *играч против играча* (енг. *PvP match*). Играч против играча је режим видео игре у којем више играча директно учествује у такмичењу или борби. Оне обухватају многе популарне жанрове, као што су *MOBA* (енг. *multiplayer online battle arena*) или пуцачке игре (енг. *first-person shooting – FPS*) [28].

1.2 Модели вештина за упаривање играча

Модел вештина у упаривању играча (енг. *skill-based matchmaking model*) је алгоритам који се користи у играма како би се играчи упарили на основу њихове вештине или способности. Модел користи различите факторе за процену вештине, као што су статистике, резултати претходних утакмица, ранг листе или друге метрике. На основу ових података, алгоритам процењује ниво вештине сваког играча и врши упаривање тако да играчи са сличним нивоом вештине играју заједно.

Рејтинг (eng. *rating)* је мерило које се користи за евалуацију и рангирање резултата неке особе. У контексту игара, рејтинг се обично односи на оцену или рангирање играча на основу њихових постигнућа или успеха остварених приликом играња. Рејтинг системи омогућавају компаративну анализу између учесника и пружају меру њихове способности. Мотивација за оцену вештина долази из потребе да се играчи рангирају и потребе да се обезбеди равноправност при упаривању. До те оцене се може доћи на различите начине, примера ради Бредли-Теријевим моделом или Ело системом.

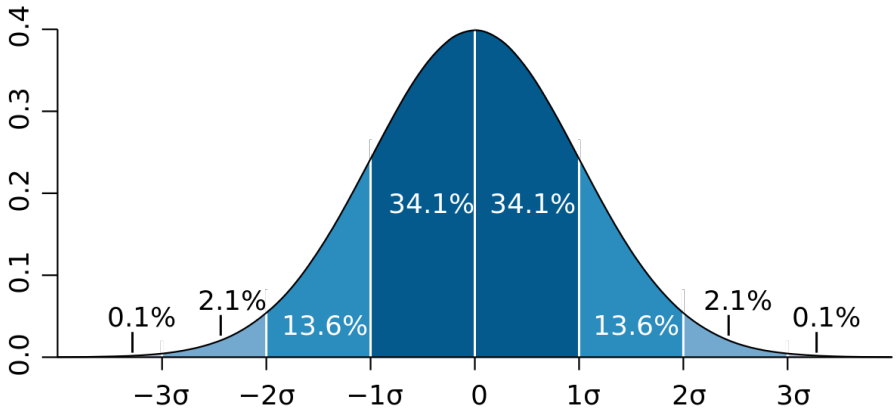
Брадли-Теријев модел за процену индивидуалних вештина на основу поређења парова широко се примењује у многим областима, као што су статистика, спорт и машинско учење [25]. Сложеност Бредли-Теријевог модела је $\mathcal{O}(n^2)$, где је n број елемената који се рангирају. Ова сложеност произилази из потребе за поређењем сваког пара елемената како би се израчунала њихова међусобна вероватноћа победе. Будући да има $\frac{n\cdot(n-1)}{2}$ могућих парова за поређење, сложеност модела расте квадратно са бројем елемената. Бредли-Теријев модел сваком играчу i додељује фиксни скалар γ_i који представља његове способности, примера ради тај скалар може бити једнак рангу играча. Вероватноћа да победи играча j се моделује као однос способности играча i и суме способности оба играча [20].

$$P(\text{играч } i \text{ побеђује играча } j) = \frac{\gamma_i}{\gamma_i + \gamma_j}$$

Овај модел оцењује вештине играча тек након што прође кроз све могуће парове и за сваког играча одреди вероватноћу да он победи било ког другог играча.

У својој књизи [13] физичар и шаховски мајстор Арпад Ело предлаже систем за оцену вештина који одређује релативне оцене играча на основу вероватносног модела. У такмичарским активностима као што је шах, турнирски резултати пружају привремена рангирања, али због варијација у појединачним наступима, ранг листа заснована на једном догађају није увек потпуно поуздана. Зато систем оцењивања тежи да прикаже све наступе појединца или тима на некој врсти скале, тако да се у сваком тренутку такмичари могу представити претпостављеним редоследом њихове способности. Додатно, адекватан систем оцењивања треба да иде корак даље од самог рангирања и треба да пружи процену релативне вештине такмичара. Релативна оцена је мера која се користи за рангирање или поређење елемената на основу њихових међусобних вештина или способности. Уместо да се користе конкретне вредности које представљају апсолутне вештине елемената, релативна оцена вештина се базира на односима између елемената. Ово значи да се не бави самим вредностима вештина, већ се фокусира на упоређивање елемената и на то како се међусобно надмећу. На основу резултата поређења, добија се информација о томе који је елемент јачи или има већу вероватноћу да победи у односу на друге елементе. Елов систем оцењивања је бројчани систем у којем се разлике у рејтингу играча могу претворити у вероватноћу победе. И обрнуто, проценти освојених поена могу се претворити у разлику у рејтингу играча. То је научни приступ у процени шаховских партија [13].

Модел оцењује перформансе p_i играча i преко једнодимензионе Гаусове расподеле, са средњом вредношћу r_i.



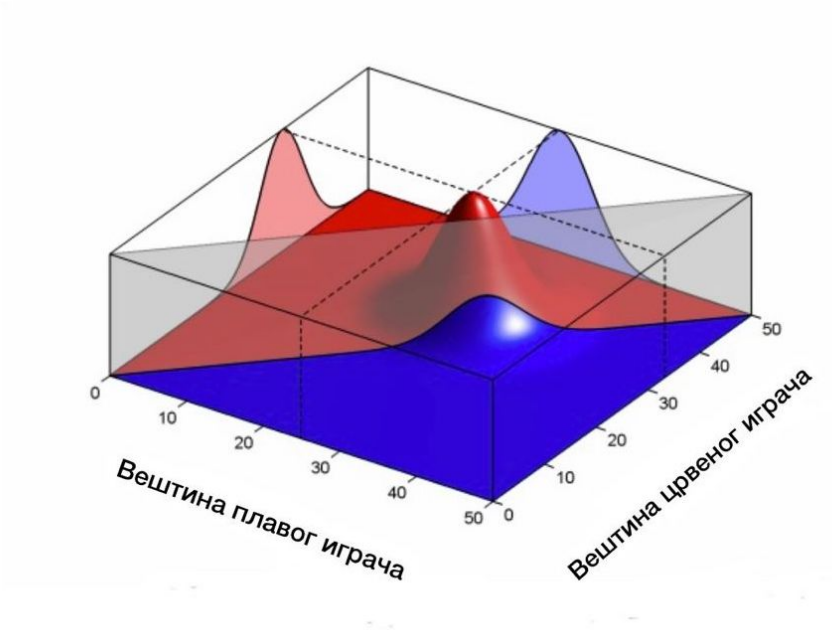
Слика 1.1: График стандардне девијације [23]

На слици 1.1 може се видети визуелни приказ овог концепта. Средина криве представља просечни резултат који играч остварује. За довољан број резултата играча (енг. *player scores*), функција добија облик глатке криве познате под називом функција густине нормалне расподеле. Ова крива је симетрична у односу на средњу вредност, и приближно две трећине свих резултата упадају у опсег од једног стандардног одступања σ са обе стране средње вредности. То значи да ће играч углавном постизати резултате који су у околини средње вредности. Преостала трећина резултата налази се изван овог опсега, и подједнако су распоређени на крајевима криве.

За игру са два играча, вероватноће победе и пораза моделују се помоћу Гаусових функција расподеле. Слика 1.2 илуструје поређење вештина два играча (црвеног и плавог). Примера ради, плави играч има вештину X на одређеној скали, а црвени играч има вештину Y (модови њихових Гаусових функција расподеле означени су испрекиданом линијом). Перформансе оба играча ће варирати из игре у игру око њихових средњих вредности вештина, као што је приказано црвеним и плавим функцијама расподеле перформанси на бочним страницама квадра на слици 1.2. Уколико вештина плавог играча премашује вештину црвеног играча, предвиђа се да ће плави играч победити и обрнуто [19]. Вероватноћа ових догађаја је пропорционална црвеној и плавој запремини испод дводимензионалне звонасте криве на слици 1.2.

У Ело систему, r_i се ажурира на основу односа између очекиваних и стварних резултата. Ело систем подразумева да играчи добијају или губе бодове на основу резултата својих мечева. Сваки играч почиње са одређеним бројем бодова. Тренутни број бодова које играч има је његов *Ело рејтинг*. Када играчи играју једни против других, резултат меча се користи за израчунавање промене *Ело рејтинга*. Ако играч победи противника који има виши *Ело рејтинг*, он добија више бодова. Ако играч изгуби од противника са нижим *Ело рејтингом*, он губи више бодова. Обрнуто, ако играч победи противника са нижим рејтингом, он добија мање бодова. За разлику од Бредли-Теријевог модела, r_i се може ажурирати одмах након сваке игре играча j. Ако γ_i и γ_j представљају редом способности играча i и j, вероватноћа да играч i победи играча j изражена је следећом формулом [26]:

$$P(\text{играч } i \text{ побеђује играча } j) = \frac{1}{1 + 10^{(\gamma_j - \gamma_i)/400}} \tag{1.1}$$



Слика 1.2: Графички приказ поређења вештина два играча [19]

Глико систем рејтинга је побољшана верзија Ело система. Док Ело систем само користи победу, пораз или нерешен резултат, Глико систем узима у обзир и квалитет игре, варијацију резултата и број одиграних мечева. Глико систем узима у обзир не само резултат меча, већ и сигурност или поузданост резултата [18].

Глава 2

Алгоритми упаривања

У овом поглављу представљено је неколико врста алгоритама за упаривања играча. Ови алгоритми играју битну улогу у правилном повезивању играча према њиховим карактеристикама, вештинама или претходним резултатима како би се обезбедило квалитетно играчко искуство. Фокус ће бити на неколико често коришћених алгоритама упаривања у играма. Они нуде различите приступе и стратегије за формирање парова играча. Наведени алгоритми су подељени у две категорије: алгоритми засновани на редовима и алгоритми засновани на графовима. Ови алгоритми приступају проблему упаривања на више начина, користећи различите структуре података и технике за постизање жељених резултата.

2.1 Стратегије засноване на редовима

Стратегије засноване на редовима пружају приступ за упаривање који је једноставан за имплементацију и разумевање. Ови алгоритми користе редове, односно листе играча, како би их упарили према одређеним критеријумима. Стратегије засноване на редовима које се могу користити у упаривању играча су *FIFO* алгоритам и алгоритми засновани на вештинама (енг. *skill-based matchmaking*).

FIFO алгоритам

FIFO (енг. *first-in-first-out*) алгоритам за упаривање играча у видео играма представља један од најједноставнијих приступа који осигурава непристрасно

упаривање међу играчима. Овај алгоритам функционише на основу концепта реда.

Принцип *FIFO* алгоритма је једноставан: играчи се упарују према редоследу у коме су се придружили игри, односно затражили упаривање. Први играч који је доступан за упаривање бива упарен са следећим доступним играчем у реду. Овај процес се наставља док се сви играчи не упаре или док не буде задовољен неки други услов упаривања, као што је временско ограничење или граница на број направљених упаривања.

Једна од предности *FIFO* алгоритма је његова једноставност и брзина упаривања. Он обезбеђује брзо упаривање играча без комплексног анализирања специфичних параметара играча. *FIFO* алгоритам је непристрасан, јер не доноси одлуке о упаривању на основу карактеристика играча, већ сви играчи бивају упарени према њиховом редоследу пријаве.

Ипак, *FIFO* алгоритам може имати нека ограничења. Овај алгоритам не узима у обзир специфичну способност, степен вештине или ранг играча. Такав стил упаривања лако може креирати небаланисаране партије. Играчи могу бити незадовољни када су упарени са играчима који поседују веома лошије или веома боље вештине, из разлога што овакво упаривање може резултирати неуравнотеженом партијом. Искуснији играчи могу доминирати и лако победити мање веште противнике, што може довести до неинтересантног искуства за играче укључене у меч. Недостатак изазова и праве конкуренције може смањити задовољство игре за све учеснике.

Као унапређење, *FIFO* алгоритам може бити проширен да узима у обзир додатне факторе, као што су ранг играча, бодови или статистике, како би се подигла равноправност и квалитет упаривања. Један алгоритам тог типа је описан у следећем одељку.

Алгоритам са сортирањем по вештинама

У поглављу 1.2 уведен је модел вештина. Он је значајан у погледу ове врсте упаривања јер пружа слику о способностима играча и даје метрику за њихово поређење. За разлику од *FIFO* алгорима који не узима у обзир никакву врсту поређења сем редоследа уласка у игру, овај алгоритам то ради.

Након што су познате карактеристике играча, овај алгоритам прво врши сортирање играча на основу одабраних параметара, као што су ранг, способ-

ности или удео победа у свим одиграним партијама. Затим, упарује играче у редоследу у коме су сортирани.

Алгоритам са сортирањем по вештинама има предност у томе што обезбеђује упаривање играча на основу њихових карактеристика. Ово може довести до бољег баланса међу играчима у игри. Алгоритам омогућава да играчи са сличним карактеристикама играју заједно, што може побољшати искуство играња. Овакав стил упаривања познат је као упаривање на основу једнаких вештина (енг. *equal-skill based matchmaking*).

2.2 Стратегије засноване на графовима

Упаривање играча може бити засновано и на теорији графова (енг. *graph matching model*). Претходно разматрана стратегија упаривања се ослања на претпоставку да спајање играча са сличним вештинама обично за последицу има компетитивне игре које играчи желе [19]. Међутим, треба размотрити да ли је овај интуитивни систем упаривања увек користан за играче. Спајање играча са супарницима сличног нивоа вештина може имати различит утицај на њих због разлике у очекивањима које они могу имати за следеће мечеве. Вештина и успех играча у претходним мечевима може утицати на њихова очекивања:

- ако је играч недавно постигао добре резултате и има добар низ победа, његово очекивање може бити да ће наставити побеђивањем;
- ако је играч имао неуспешну серију пораза, може имати нижа очекивања и бити мало песимистичан у вези са наредним мечом.

Други фактор који утиче на очекивања играча је њихов статус или циљ у игри. Неки играчи могу бити веома такмичарски настројени и амбициозни, што значи да им је битно да побољшају свој пласман и освоје што више победа. Ови играчи потенцијално могу очекивати да ће бити спарени са играчима који су на њиховом нивоу или су чак мало бољи, како би им они пружили изазов и стимулисали их да покажу своје најбоље вештине.

С друге стране, постоје играчи који су опрезни и брину се о свом рангу или репутацији у оквиру игре. Они можда више воле да буду спарени са играчима сличног или нижег нивоа, како би имали што већу шансу за победу

и тиме заштитили свој ранг. Ови играчи могу очекивати да ће бити упарени са играчима који су на сличном нивоу вештине.

Укратко, очекивања играча могу варирати због њихових претходних перформанси, амбиција, циљева у игри и преференције за упаривање са одређеним типом играча. Ово наглашава важност пружања индивидуално прилагођеног упаривања које ће задовољити потребе различитих играча и пружити им задовољство и ангажовање у игри [28].

Ризик од одсуства играња (енг. *churn risk*), тј. ризик од неангажованости, представља вероватноћу или опасност да играч има период одсуства након одиграног меча. Насупрот овом појму, **ангажованост играча** представља његову непрекидну активност у игри. Ризик од одсуства играња, односно, престанка са игром у одређеном временском периоду (у овој табели 7 дана) након меча приказан је у табели 2.1. У табели се јасно може видети пораст ризика од неангажовања играча у зависности од три претходно одиграна меча. Исходи мечева могу бити победа (П), губитак (Г) и нерешено (Н). Ризик одсуства играња у стању након три губитка је 5,1%, што је скоро два пута веће него код стања када је играч у последњој партији победио након изгубљених или нерешених партија (2,6%-2,7%).

Из ових података може се закључити да стратегије упаривања треба да зависе од динамичких и индивидуалних стања играча [28].

Теоријске основе алгоритама заснованих на графовима

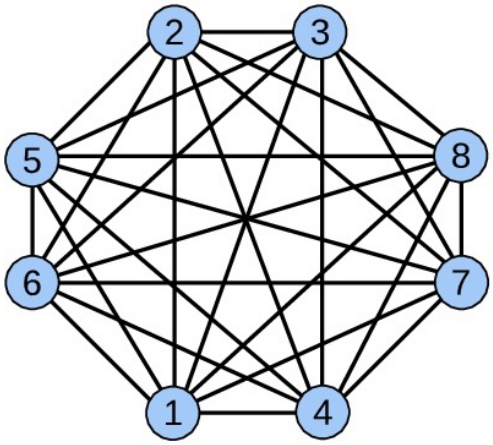
Упаривање корисника у видео играма се може формално дефинисати као оптимизациони проблем и свести на проблем упаривања чворова у неусмереном тежинском графу тако да се максимизује укупно ангажовање играча,

Табела 2.1: Табела ризика од одсуства играња направљена од стране ЕА (енг. *Electronic Arts*)

исход последња 3 меча	ризик
НГП | ГГП | ГНП | ННН	2.6% - 2.7%
...	...
ППП	3.7%
...	...
НГГ | ГПГ | ГНГ	4.6% - 4.7%
ППГ	4.9%
ГГГ	5.1%

или еквивалентно, минимизује укупно неангажовање играча.

Граф $G = (V, E)$ састоји се од скупа V чворова и скупа E грана. За задати неусмерени граф $G = (V, E)$ упаривање је скуп дисјунктних грана (грана без заједничких чворова). Назив упаривање долази од чињенице да се гране могу интерпретирати као парови чворова. Тежински граф је граф чијим гранама су придружени реални бројеви (тежине, цене, дужине) [30]. Чвор који није суседан ниједној грани из упаривања је неупарен чвор, каже се такође да такав чвор не припада упаривању. Савршено упаривање је упаривање у коме су сви чворови упарени, односно у коме је сваки чвор из графа инцидентан са тачно једном граном у упаривању. Максимално упаривање је пак упаривање које се не може проширити додавањем нових грана [31]. Комплетан граф је граф у ком сваки његов чвор има степен $n\text{-}1$, где је n укупан број чворова, што се може видети на слици 2.1.



Слика 2.1: Комплетан граф G [1]

У неусмереном тежинском графу $G = (V, E)$, упаривање минималне тежине (енг. *minimum weight matching – MWM*) је подскуп M непреклапајућих грана из скупа E са најмањим збиром тежина.

Савршено упаривање минималне тежине (енг. *minimum-weight perfect matching – MWPM*) је савршено упаривање са најмањим збиром тежина грана у коме су сви чворови упарени [28]. Циљна функција у MWPM алгоритму се обично дефинише као укупна тежина или цена упаривања. Циљ је пронаћи савршено упаривање (упаривање у коме је сваки чвор упарен) са минималном

$$\begin{bmatrix} & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 \\ P_1 & 0 & 5 & 1 & 15 & 20 & 23 & 3 & 4 \\ P_2 & 5 & 0 & 10 & 23 & 5 & 21 & 1 & 20 \\ P_3 & 1 & 10 & 0 & 33 & 7 & 13 & 8 & 50 \\ P_4 & 15 & 23 & 33 & 0 & 27 & 44 & 10 & 1 \\ P_5 & 20 & 5 & 7 & 27 & 0 & 1 & 22 & 2 \\ P_6 & 23 & 21 & 13 & 44 & 1 & 0 & 18 & 11 \\ P_7 & 3 & 1 & 8 & 10 & 22 & 18 & 0 & 24 \\ P_8 & 4 & 20 & 50 & 1 & 2 & 11 & 24 & 0 \end{bmatrix}$$

Слика 2.2: Матрица удаљености која представља тежину грана графа

(максималном) укупном тежином.

Матрица удаљености је квадратна матрица чија је величина једнака броју чворова графа. Матрица удаљености на слици 2.2 представља приказ односа повезаности између чворова у графу, у овом конкретном графу представља тежине грана између свака два чвора. С обзиром да је граф неусмерен, матрица повезаности је симетрична, а вредности елемената на дијагонали су нуле (у контексту упаривања играч не може бити повезан са собом). Елемент матрице на позицији *(i, j)* једнак је тежини гране између два чвора.

За овакву матрицу удаљености се може уочити савршено упаривање најмање тежине. Том упаривању припадају гране:

$$\{P_1, P_3\}, \{P_2, P_7\}, \{P_4, P_8\}, \{P_5, P_6\}$$

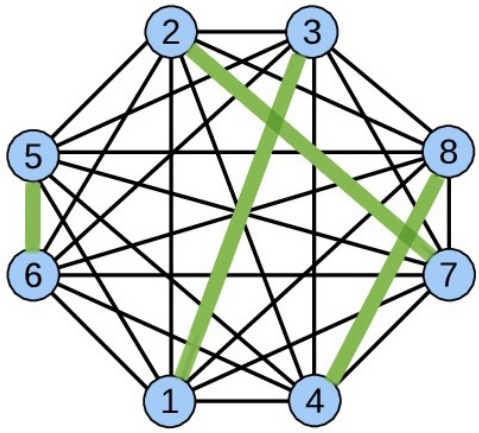
То су гране упарених чворова које имају најмањи збир тежина од свих могућих упарења и износи 1+1+1+1=4. У овом случају, број чворова је паран, међутим у општем случају не мора бити тако.

На слици 2.3 илустровано је како граф изгледа након упаривања. Зеленом бојом су означене гране које припадају упаривању.

MWM и *MWPM* имају широку примену у другим областима, укључујући формирање парова пратећи специфична правила на шаховским турнирима [29] и пренос слика преко мрежа [11].

Минимизација ризика

Игра се може моделирати тако да сви играчи који чекају у бази играча представљају чворове комплетног графа. Дакле, између свака два играча



Слика 2.3: Упарени чворови комплетног графа G

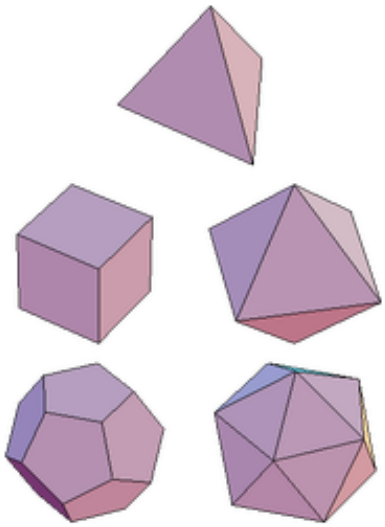
постоји грана, што даље значи да могу бити упарени. У том графу треба пронаћи гране које представљају савршено упаривање у тежинском графу. Тежина гране изражава збир ризика за оба играча уколико би били упарени. Тај ризик је њихово индивидуално стање у тренутку упаривања.

Упаривање се сада може дефинисати као оптимизациони проблем који на основу ангажовања налази најмањи збир тежина ивица у комплетном графу или скраћено *EOMM* (енг. *Engagement Optimized Matchmaking*) [28]. Налажењем најмањег збира долази се до најмање ризичних упаривања.

Овај проблем је специјални случај проблема налажења савршеног упаривања најмање тежине (*MWPM*), где су тежине грана дефинисане преко ангажовања игача. Такво упаривање је скуп непреклапајућих парова чворова који имају минималну тежину грана у комплетном тежинском графу [8].

За разлику од претходних метода упаривања који хеуристички спарују играче сличних вештина, *EOMM* има за циљ да спарује играче на оптималан начин који максимизује укупну ангажованост играча.

Први покушај решавања *MWPM* проблема у полиномском времену био је алгоритам цветања (енг. *blossom algorithm*) који је формулисао амерички математичар Џек Едмонс. У раду [12] се налази опис алгоритма који проналази максимално упаривање у тежинском графу који је представљен конвексним полиедром. Конвексни полиедар је геометријски објекат у тродимензионалном простору који се састоји од равних површи и рогљева тако да свака дуж



Слика 2.4: Конвексни полиедри [6]

која повезује две тачке унутар полиедра лежи потпуно унутар полиедра, а њихова илустрација се налази на слици 2.4. Конвексни полиедар одговара упаривањима у графу, при чему темена полиедра представљају чворове који се упарују, а ивице поледра гране. Алгоритам описан у раду ефикасно налази упаривање у полиедру са максималном збирном тежином, где свака страница полиедра носи одређену тежину. У раду се помињу дела других истраживача у области геометрије полиедра и истиче њен значај у решавању комбинаторних проблема коришћењем техника линеарног програмирања [12].

Већина система за упаривање претпоставља да су игре са балансираним вештинама добре за ангажовање и зато користе алгоритме за оцењивање вештина како би идентификовали противнике сличних способности [19].

Ангажованост играча може се сматрати објективном мером корисничког искуства у играма [5]. Ризик од неангажованости дефинише се као пропорција укупног броја играча који престају да играју игру у току одређеног временског периода. Предвиђање ризика од неангажовања (енг. *churn prediction model*) се примењује у различитим дисциплинама већ деценијама, као што су телекомуникације [27], претплатнички сервиси [21] и осигурање возила [4].

У наставку је функција циља дефинисана преко ризика од неангажованости.

Нека је P скуп свих играча који чекају на упаривање.

Стање играча, s_i, је скуп индивидуалних карактеристика које поседује

играч, укључујући вештине, учесталост играња, успех у одиграним партијама итд.

Ризик од неангажованости, $c_{i,j}$, играча $p_i \in P$ након упаривања са играчем $p_j \in P$ је функција стања оба играча:

$$c_{i,j} = P(p_i \ churns| \ s_i, \ s_j) = \mathrm{c}(s_i, \ s_j).$$

Треба напоменути да је:

$$c_{i,j} \neq c_{j,i}$$

зато што упаривање може различито утицати на играче с обзиром да је задовољство упаривањем индивидуално.

Резулатат упаривања биће представљен списком парова играча:

$$M = \{p_i, p_j\}$$

у којим су сви играчи у P спарени тачно једном.

Укупан ризик од неангажованости играча је збир свих појединачних ризика од неангажованости, из чега произилази да је циљ пронаћи упаривање које минимизује тај збир.

$$M^* = \min_M \sum_{\{p_i,p_j\}\in M} [c(s_i, s_j) + c(s_j, s_i)] \tag{2.1}$$

За моделовање игре конструише се граф G. Сваки играч p_i је чвор графа, који има стање играча s_i пре упаривања. Грана између два играча p_i и p_j има тежину:

$$c_{i,j} + c_{j,i}$$

која представља очекивану суму ризика од неангажованости ако би играчи p_i и p_j били упарени.

Као што је већ напоменуто, G је потпуни граф, па било која два играча могу бити упарена. Када су сви ризици од неангажованости $c_{i,j}$ израчунати (све тежине грана познате), тражење M^* у формули 2.1 се своди на проблем минималног тежинског савршеног упаривања, тј. на налажење парова са минималном збирном тежином грана на графу G.

Поједностављено предвиђање ризика од неангажованости $c_{i,j}$ је да се он базира на:

- стању s_i играча p_i;
- исходу меча $o_{i,j}$ из перспективе играча p_i.

Ово функционише зато што стање противника s_j, као што је вештина, историја игре и стил, нема директну интеракцију са ризиком одласка $c_{i,j}$ играча p_i. Међутим, то утиче на исход будућег меча, који директно утиче на играча p_i и стога утиче на његов ризик одласка на одређено време. Када је исход меча $o_{i,j}$ познат, $c_{i,j}$ постаје условно независан од стања противника s_j. Формално, ово својство је представљено као:

$$c(s_i, s_j, o_{i,j}) = c(s_i, o_{i,j}) \tag{2.2}$$

Могући исходи игре припадају коначном скупу $O = \{\text{Победа, Губитак, Нерешено}\}$. На пример:

$o_{i,j} = \text{П}$ значи да p_i побеђује p_j из перспективе p_i
$o_{j,i} = \text{Г}$ представља исти исход из перспективе p_j

За приближно предвиђање вероватноће исхода игре могу се користити стандардни модели вештине [13, 20].

Репрезентација вештине играча p_i означена као r_i је примера ради: Ело рејтинг [13], Глико просек [18] или једноставно ранг додељен играчу p_i. Дакле, r_i је део стања s_i играча p_i.

Као резултат, може се рећи да је:

$$P(o_{i,j}|s_i, s_j) \approx P(o_{i,j}|r_i, r_j) \tag{2.3}$$

Дакле, вероватноћа исхода $o_{i,j}$ под условом да су стања играча p_i и p_j редом s_i, s_j је приближно једанака вероватноћи исхода $o_{i,j}$ под условом да су вештине играча p_i и p_j редом r_i, r_j.

Спајањем свих делова, може се ефикасно предвидети ризик одласка спарених играча из формуле 2.1:

$$c(s_i, s_j) + c(s_j, s_i) = \sum_{o_{i,j} \in O} P(o_{i,j}|s_i, s_j)(c(s_i, s_j, o_{i,j}) + c(s_j, s_i, o_{j,i})) \approx \sum_{o_{i,j} \in O} P(o_{i,j}|r_i, r_j)(c(s_i, o_{i,j}) + c(s_j, o_{j,i}))$$

где је прва једнакост маргинализација на исходу игре $o_{i,j}$. У апроксимативној једнакости користи се условна независност $c_{i,j}$ од s_j, дати исход $o_{i,j}$ (једначина 2.2) и предвиђање исхода игре (једначина 2.3).

Сада се $c(s_i, o_{i,j})$ може посматрати као стандардни проблем предвиђања одустајања играча на одређени период. Битна информација је ажурирано стање играча након што је познат исход меча, односно:

$$s_i' \leftarrow s_i \text{ и } o_{i,j}$$

Ако је o_i^K низ исхода претходних K мечева (нпр. за K=4, o_i^K = ГПНП, што је низ који чине Губитак, Победа, Нерешено, Победа), $\hat{s}_i$ скуп осталих способности играча, стање играча се може формулисати на следећи начин:

$$s_i = [o_i^K, \hat{s}_i]$$

Ажурирање стања играча:

$$s_i' = [o_i^K, \hat{s}_i] \text{ и } o_{i,j}$$
$$s_i' = [o_i^{K+1}, \hat{s_i'}]$$

Ознака $\hat{s_i'}$ служи да покаже да се одиграним мечом ажурирају и друге каратеристике играча, рецимо број одиграних партија.

Када је информација о предвиђеним ризицима од одласка за сваки пар играча позната, односно позната је тежина сваке гране у графу G, алгоритам упаривања се своди на проблем минималног тежинског савршеног упаривања. Циљ је наћи парове M^* на потпуном графу G тако да имају најмању збирну тежину грана.

За граф са N чворова, један од начина јесте исцрпно поређење (енг. *brute force*) за добијање свих могућих упаривања које има сложеност $\binom{N}{N/2}/2^{N/2}$, али временска сложеност је превише висока да би била изводљива у практичним системима. Међутим, постоје алгоритми полиномске временске сложености за проблем *MWPM*. Неколико алгоритама може решити проблем у најгорој временској сложености $\mathcal{O}(N^3)$, као на пример [16].

Ако би мере ангажованости биле цели бројеви, постоји мало бржи алгоритам [15] са временском сложеношћу $\mathcal{O}(N^{11/4} \cdot logK)$, где је K највећа апсолутна вредност тежине гране.

Постоје и похлепни алгоритми, као што је [10], са бржим временским извршавањем за проналажење субоптималних решења.

Осим поменутих стратегија, *MWPM* се може решити паралелно како је предложено у [24].

Глава 3

Имплементација алгоритама и резултати

У оквиру овог рада, имплементирани су и симулирани претходно дефинисани алгоритми за упаривање. Сваки од ових алгоритама је тестиран кроз одређени број симулација и детаљно је праћено понашање сваког од њих.

Након спроведених симулација, приказани су резултати упаривања добијених применом ових алгоритама. Они ће приказати ефикасност сваког алгоритма за упаривање и идентификовати најбољи алгоритам који резултује највећом ангажованошћу играча. Приказани резултати нам дају увид у перформансе и предности сваког алгоритма, што може бити од користи приликом доношења одлука о имплементацији одговарајућег упаривања у практичним сценаријима.

Алгоритми који ће бити тестирани су следећи:

- алгоритми засновани на редовима:
  - *FIFO* алгоритам;
  - алгоритам заснован на рангу;
  - алгоритам заснован на уделу победа у одиграним партијама;
- алгоритми засновани на графовима:
  - алгоритам за максимизацију ангажованости (минимизацију ризика од неагажованости);
  - алгоритам за минимизацију ангажованости (максимизацију ризика од неагажованости).

3.1 Опис података

За симулације алгоритама за упаривање креира се скуп поља која описују перформансе играча. Неке од перформанси биће генерисане на случајан начин, док ће неке бити иницијализоване на нулу. Особине ирача чија је почетна вредност нула ће се у току одигравања партија развијати, побољшавати или погоршавати у складу са исходима партија.

Међутим, инхерентна снага (интелигенција, брзина рефлекса) играча може бити унапред постављена на одређену вредност, што одређује њихове специфичне способности на самом почетку. Инхерентна снага се односи на природну или урођену снагу, својство или квалитет који је увек присутан код некога. Када се користи у контексту играча, инхерентна снага представља њихов таленат, природну снагу која је својствена њима и није резултат специфичног тренинга или развоја.

За креирање поља и вредности перформанси играча користи се скрипта направљена у програмском језику *Python*. Подаци се складиште у датотеку формата *csv*.

Играче описују следећа поља: корисничко име играча (енг. *username*), снага (енг. *strength*), ранг као мерило вештине играча (енг. *rank*), број победа (енг. *wins*), број пораза (енг. *losses*), број изједначених партија (енг. *draws*), исход претпоследњег меча (енг. *penultimate outcome*) и исход последњег меча (енг. *last outcome*).

За генерисање насумичних корисничких имена (енг. *username*) употребљена је библиотека *Faker* [7]. При креирању података осигурано је да имена имају јединствене вредности. Поље које означава инхерентну снагу играча генерисано је помоћу библиотеке *random*. Важно је да вредности које описују снагу буду насумичне како би покриле што више различитих стања у којима се могу наћи играчи. То омогућава да се упаривање обавља између различитих профила играча, што доприноси разноврсности и занимљивости игре. Као што је случај у реалном свету, играчи имају различите урођене предности и особине које утичу на њихову игру.

Ранг играча је цео број који је на почетку игре постављен на нулу. Број победа, пораза и нерешених партија је такође нула приликом првог покретања игре. Исход меча може бити један од три стања: победа, пораз и нерешено, па на основу тога поља која представљају исходе последња два меча могу имати

Табела 3.1: Пример садржаја *csv* датотеке са подацима о играчима

username	strength	rank	wins	losses	draws	penultimateOut	lastOut
jose77	2	455	75	64	10	0	-1
emoody	510	86	87	92	5	-1	1
fford	700	20	84	43	25	-1	0
terri18	123	100	6	42	56	0	1
hanry3	89	5	96	56	48	1	1

вредности из скупа {1, -1, 0}. При покретању скрипте формирају се подаци за 300 различитих играча. У табели 3.1 приказан је исечак из *csv* датотеке са подацима о корисницима након извесног броја одиграних партија.

3.2 Опис имплементације

Имплементација алгоритама реализована је у програмском језику *Python*. Комплетан код је доступан у *GitHub* репозоторијуму на следећој адреси: `https://github.com/aleksandraniksic/masters_thesis`. Стратегије које су имплементиране у раду представљене су псеудокодовима.

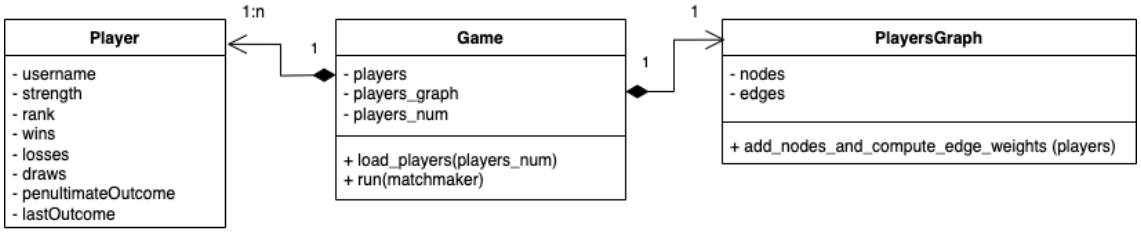
Имплементација општих аспеката симулације

Компоненте приказане на дијаграму 3.1 које се користе за симулацију су:

- *Player* – класа која представља играча;
- *PlayersGraph* – класа која представља неусмерен граф у ком су играчи чворови, а гране потенцијална упаривања међу њима;
- *Game* – класа која представља симулацију упаривања и одигравања игара.

Класа *Player* има атрибуте који осликавају карактеристике играча. Ови атрибути се учитавају из датотеке са подацима, за сваки ред у датотеци креира се објекат класе *Player*.

Класа *Game* представља симулацију једне игре, она упарује играче на основу различитих стратегија упаривања и рачуна просечан број задржаних играча. У конструктору ове класе се најпре извршава метода *load_players()*



Слика 3.1: Класни дијаграм

кроз коју се учитавају играчи из *csv* датотеке и креира листа објеката играча. Затим се израчунавају тежине грана између играча и прави се граф играча.

Метода *run()* је одговорна за покретање игре. Након утврђивања који алгоритам упаривања је изабран, позива се одговарајућа метода упаривања (узимајући у обзир или граф играча или листу играча у зависности од алгоритма). Након упаривања, израчунава се очекивани проценат играча који ће бити задржан на основу тежина грана у графу.

Када је упаривање завршено позива се метода *update_player_data()* унутар које се одигравају партије и на основу исхода мечева ажурирају карактеристике играча. Овај корак је битан због великог броја симулација упаривања над истим скупом играча, како би се пратио њихов напредак кроз рунде и повећала разноврсност података.

Партија се одиграва у методи *play_the_match()* која као улазни параметар има два упарена играча смештена у променљиву *pair*. Функција се користи за одређивање победника између та два играча.

Прво се дефинишу тежине за сваки фактор који се узима у обзир при одређивању резултата меча. Тежине су дефинисане следећим вредностима:

- *strength_weight*: тежина снаге играча – 0.4;
- *streak_weight*: тежина низа победа – 0.3;
- *random_weight*: тежина случајног фактора – 0.3.

Низ победа се рачуна користећи функцију која сабира вредности исхода последњих партија. С обзиром да исходи припадају скупу {1, -1, 0} збир је добар показатељ успешности у одиграним партијама.

Затим се рачуна резултат за оба играча. Резултат се рачуна као збир производа тежине и одговарајућег фактора за сваког играча. Фактори који се узимају у обзир су:

- *strength* – јачина играча;
- *streak* – низ победа играча;
- случајни фактор генерисан помоћу *random.random()* функције.

Коначно, функција упоређује резултате играча *player1_ score* и *player2_ score*, а затим враћа или победника или обавештење да је резултат партије нерешен (*draw*).

Ако је резултат *player1_ score* већи од *player2_ score*, повратна вредност фукције је играч *player1*. Ако је резултат *player2_ score* већи од *player1_ score*, враћа се играч *player2*.

Ранг играча је атрибут који има посебну логику ажурирања вредности. Посматрајући из перспективе играча p_i резултат одигране партије на следећи начин мења његову вредност:

- ако је p_i победио играча p_j:
  - ранг играча p_i се повећава за четвртину разлике између рангова играча, уколико је победио јачег играча;
  - ранг играча p_j се повећава за осмину разлике између рангова играча, уколико је победио слабијег играча;
- ако је p_i изгубио од играча p_j:
  - ранг играча p_i се смањује за четвртину разлике између рангова играча, уколико је изгубио од слабијег играча;
  - ранг играча p_j се смањује за осмину разлике између рангова играча, уколико је изгубио од јачег играча;
- ако је резултат нерешен, рангови оба игача остају исти.

Формално, уколико партија није завршена нерешеним резултатом, израчунава се разлика у вредности ранга између победника и губитника и смешта у променљиву *rank_ diff*. Ако је разлика већа од нуле, то значи да је играч победио играча који има нижи ранг. У овом случају, играч који је победио добија одређени број поена који зависи од разлике у рангу. С обзиром да је победио у једноставном мечу, фактор којим се множи разлика рангова је једна осмина, и формула за добитак у погледу ранга је:

$$rank\_gain = 0.125 \cdot rank\_diff$$

Уколико је разлика у рангу мања од нуле, то значи да је играч победио играча који има виши ранг. У овом случају, играч који је победио добија одређени број поена који такође зависи од разлике у рангу, али се умножава већим фактором (једна четвртина) јер је победио у изазовном мечу. Формула за добитак у погледу ранга је:

$$rank\_gain = 0.25 \cdot rank\_diff$$

Када је вредност поена *rank_ gain* позната, ранг победника се повећава за израчунати број поена, док се ранг губитника смањује за израчунати број поена.

Мотивација за ову врсту ажурирања долази из Ело система у ком играч који победи противника са већим Ело рејтингом добија више бодова, а играч који изгуби од противника са мањим Ело рејтингом губи више бодова. Обрнуто, ако играч победи противника са мањим рејтингом добија мање бодова, док губитак против таквог противника резултује већим губитком бодова. Користи се формула 1.1.

Као повратна вредност метода *run()* враћа се проценат задржаних играча.

Поређење различитих метода упаривања играча у оквиру симулиране игре обављено је у модулу *comparison.py*. Псеудокод је приказан у наставку:

```
strategije_uparivanja = [MAX_EMM, SB_R, SB_WR, FIFO, MAX_DMM]
broj_simulacija = 500
broj_igraca = [50, 100, 150, 200, 250, 300]
simulacija_uparivanja():
    za svaki element iz liste broj_igraca:
        pomerajuci_prosek_zadrzanih_igraca = [0, 0, 0, 0, 0]
        za svaku strategiju iz strategije_uparivanja:
            igra = Game(broj_igraca)
            za i u opsegu od 0 do broj_simulacija:
                pokreni igru sa trenutnom strategijom
                azuriraj pomerajuci prosek za trenutnu strategiju
                upisi rezultat u datoteku
```

Кроз овај модул се инстанцирају објекти класа које представљају различите стратегије за спајање играча у игри (*matchmakers*). Свака стратегија је имплементирана као засебна класа која је имплеменрирана у модулу *algorithms.mm_strategies*.

Параметри за симулацију игре су број рунди које ће бити одигране над одређеним скупом играча (*games_to_play*) и различите кардиналности скупа играча (*player_nums*) над којим ће се извршити поређење метода.

У оквиру сваке рунде креира се објекат игре (*Game*) са одређеним бројем играча као улазним параметром. Тај број одређује колико ће се играча учитати из датотеке са подацима. При креирању инстанце класе *Game*, у њој се креира и граф играча и израчунавају се тежине грана између сваког пара играча. Затим се за сваку стратегију упаривања у листи *matchmakers* извршава функција *game.run(matchmaker)* која покреће игру користећи одговарајућу стратегију упаривања играча. Током извршавања игара, прати се број задржаних играча *player_retention_rate* за сваку стратегију упаривања. Ова вредност се ажурира у свакој итерацији игре како би се израчунала просечна стопа задржавања играча током свих одиграних партија. За ажурирање се користи формула за израчунавање померајућег просека (енг. *moving average*) [14]. Та формула се користи из потребе да се рачуна средња вредност са променљивим бројем узорака. Ова формула се често користи за праћење трендова и израчунавање просека вредности на основу нових информација које долазе током времена.

Симулације су покретане за кардиналности из скупа {50, 100, 200, 250, 300}. У свакој рунди, креира се скуп за одређени број играча.

Примера ради, ако је тренутни број играча над којим се тестирају алгоритми сто, процес се обавља на следећи начин:

- алгоритми се примењују на сто играча и израчунава се број задржаних;
- ажурира се померајући просек;
- сви парови одигравају мечеве и након тога се ажурирају њихове карактеристике у складу са исходом;
- поступак се понавља у петсто рунди.

Ако су алгоритми упаривања означени као *matchmakers*, а број играча над којим се тренутно одигравају игре *player_nums*, резултат овог поступка

симулације, поновљеног петсто пута, је просечан број задржаних играча за сваки од алгоритама *matchmakers*, над свим вредностима које може имати променљива *player_nums*.

На почетку игре, играчи се учитавају из *csv* датотеке где су подаци о њима насумично генерисани. После сваке рунде, играчи добијају нове вредности на основу резултата партије у којој су учествовали. Ажурирање се обавља тако што сви парови одиграју мечеве и у складу са исходом мењају одређене карактеристике што је детаљно објашњено на почетку овог поглавља.

Алгоритми засновани на редовима

Први и најједноставнији алгоритам заснован на концепту реда је *FIFO*.

```
uparivanje_FIFO(niz_igraca):
    broj_parova = duzina(niz_igraca)/2
    lista_parova = []
    za i u opsegu 0 do broj_parova:
        par = uzmi dva igraca iz niz_igraca
        dodaj par u lista_parova
    vrati lista_parova
```

Овај алгоритам не узима у обзир карактеристике играча, већ само редом упарује играче у редоследу у ком су затражили упаривање. За прослеђену листу играча формира листу парова играча. Најпре се израчунава број парова тако што се дужина низа играча дели са два. Иницијализује се празна листа која ће садржати формиране парове. Затим се у петљи формира пар тако што се издвоје два играча са почетка низа. Петља има итерација колико и парова. Формирани пар се додаје у листу играча. Након завршетка петље, попуњена листа парова се враћа као излазна вредност функције.

Наредна два алгоритма раде по сличном принципу, са једним додатком. Пре него што се отпочне упаривање играчи се сортирају по неком критеријуму. У овом случају, један од критеријума је ранг играча, а други удео победа у одиграним партијама. Оба алгоритма су представљена наредним псеудокодом.

```
uparivanje_sa_sortiranjem(niz_igraca):
    niz_igraca = sortiraj igrace po izabranom kriterijumu
    broj_parova = duzina(niz_igraca)/2
    lista_parova = []
    za i u opsegu 0 do broj_parova:
        par = uzmi dva igraca iz niz_igraca
        dodaj par u lista_parova
    vrati lista_parova
```

На самом почетку алгоритма играчи из прослеђене листе се сортирају по жељеном критеријуму. Након тога, механизам упаривања функционише на исти начин као претходни *FIFO* алгоритам. Сложеност је нешто већа због корака сортирања и износи барем колико и само сортирање, што је у општем случају $\mathcal{O}(n \log n)$. Оно што овај стил упаривања доноси јесте креирање партија са равноправним играчима.

Алгоритми засновани на графовима

Када су у питању алгоритми засновани на графовима, користи се другачија структура података. Листа играча се трансформише у неусмерен граф.

Библиотека *networkx* [9] је популарни *Python* пакет за анализу и манипулацију графова и мрежа. Класа *PlayersGraph* наслеђује класу *nx.Graph*. Класа *nx.Graph* је реализација неусмереног графа у библиотеци *networkx*. Она пружа многе функционалности за рад са графовима, као што су додавање и брисање чворова и грана, проверу постојања чворова и грана, приступ информацијама о графу и много других операција. Поред наслеђених функционалности, класа *PlayersGraph* има и своје методе за додавање играча из прослеђене листе и израчунавање тежине грана, о којима ће бити више речи у секцији 3.2.

Имплементиране су две варијанте алгоритама базираних на графовима. Једна верзија максимизује број ангажованих играча, дакле таквим графовима је тежина гране број који представља предвиђање да играч остане и након одиграног меча. Друга верзија упаривања, која је направљена ради поређења, има за циљ да минимизује ангажованост, па за тежину грана узима ризик да играч престане да игра – и њега максимизује.

```
uparivanje_graf(igraci):
    ukoliko igraci nije graf:
        kreiraj graf G u kom su igraci cvorovi,
        a tezine grana verovatnoca da ostanu/odu
    vrati maksimalno tezinsko uparivanje grafa G
```

Овој методи се може проследити или листа играча или граф играча. Она ће проверити да ли је прослеђена вредност инстанца графа и уколико није, направиће граф у којима су чворови управо играчи из прослеђене листе.

Функција *max_ weight_ matching()* у библиотеци *networkx* налази максимално упаривање највеће тежине у графу. Параметри које је портребно проследити методи су: неусмерен граф G, логичка вредност која индикује максималну кардиналност упаривања *maxcardinality* и одговарајући атрибут гране који се односи на њену тежину *weight*. Када се методи проследи вредност *maxcardinality=True*, онда се захтева да максимално упаривање буде највеће кардиналности, што значи да је циљ наћи упаривање које обухвата што више чворова у графу.

По извршењу, метода *max_ weight_ matching()* ће вратити резултат у облику речника (енг. *dictionary*) који садржи чворове који су упарени, где су чворови означени њиховим идентификационим бројевима или именима. Резултат представља упаривање које максимизује збир тежина грана у упаривању, при чему гране немају заједничке чворове. Максимално најтеже упаривање је упаривање где је збир тежина грана у спаривању највећи могући.

Ова метода се базира на *blossom* методи за налажење алтернирајућих путева и примално–дуалнох методи (енг. *primal-dual*) за налажење упаривања највеће тежине. Обе методе је изумео амерички научник Џек Едмонс, а детљан опис алгоритма се може наћи у раду [17]. У њему су представљени ефикасни алгоритми за налажење максималног упаривања у графовима у четири варијанте: (1)упаривање највеће кардиналности у бипартитним графовима, (2)упаривање највеће кардиналности у општим графовима, (3)упаривање највеће тежине у бипартитним графовима и (4)упаривање највеће тежине у општим графовима. Први полиномски алгоритам за проблем 3 креирао је Кун, а за проблеме 2 и 4 Едмонс. Заједнички концепт који сва четири алгоритма користе је повећавајући пут (енг. *augmenting path*). Сви проблеми се решавају у етапама, у свакој етапи постоји упаривање M. Чвор i је упарен ако постоји грана *(i,j)* у M, а неупарен иначе. Грана је упарена ако та грана

постоји у M, а неупарена иначе. Алтернирајући пут (у односу на M) је прост пут такав да је свака друга грана упарена. Повећавајући пут (у односу на M) је алтернирајући пут непарне дужине који повезује два неупарена чвора, а у случају бипартитних графова крајеви грана морају припадати различитим скуповима.

На почетку алгоритма M је празан скуп, а сви чворови су неупарени. Затим се врши следећи поступак:

- изабере се један чвор из скупа неупарених чворова;
- врши се претрага у дубину (енг. *Depth First Search – DFS*) како би се пронашао повећавајући пут (он мора бити алтернирајући – гране пута наизменично припадају и не припадају упаривању M);
- ако није пронађен повећавајући пут – изабрани чвор се означава као спољни и прелази се на следећи;
- ако је пронађен повећавајући пут ажурира се упаривање M преко проширења повећавајућег пута P – то подразумева обртање грана у P, гране које су у M постају гране које нису у упарењу а гране које нису у M постају део M;
- поступак се понавља до тренутка када више нема повећавајућег пута, а тренутно упаривање је максимално упаривање у графу.

Теорема: Упаривање M има највећу кардиналност ако и само ако не постоји повећавајући пут у односу на M.

Проналажење повећавајућег пута није једноставан проблем када су у питању општи графови. У те сврхе користи се примално–дуална метода која представља оптимизациону технику која се често користи за решавање проблема линеарног програмирања. У овом приступу се истовремено прате два проблема: примални проблем (који се фокусира на проналажење оптималног упаривања) и дуални проблем (који се фокусира на одређивање оптималних тежина грана), док се промењљиве оба проблема итеративно прилагођавају како би се конвергирало ка оптималном решењу. За примални проблем се дефинише фукција циља (у случају упаривања минимизација или максимизација укупне тежине грана у упаривању). Дуални проблем се изводи из прималног проблема тако да се сваком ограничењу у прималном проблему

додељује одговарајућа дуална променљива. Циљ дуалног проблема је минимизација (у случају да је примални проблем био максимизација) или максимизација (у случају да је примални проблем био минимизација) у складу са дуалним променљивима.

У случају прималног проблема у сваком кораку се тражи побољшање у упаривању путем одређивања повећавајућег пута. Паралелно са решавањем прималног проблема, прати се дуални проблем. Свака грана је повезана са одговарајућом дуалном променљивом. Ове дуалне променљиве одражавају колико кошта додати или уклонити одређену грану из тренутног упаривања у складу са тежинама које су додељене гранама. То омогућава алгоритму да изабере најповољније гране за формирање оптималног упаривања.

Одређивање тежина грана

Одређивање тежина грана обавља се при иницијализацији игре (при инстанцирању објекта класе *Game*), када се позива метода *load_players()* која учитава играче из *csv* датотеке, а затим додаје чворове у граф и рачуна тежине грана између свака два играча.

Тежина гране се одређује у методи *predict_total_churn()* на основу предвиђања ризика да та два чвора буду упарена. Укупни ризик играча се израчунава комбинујући различите факторе као што су вероватноћа да играч има одређени исход (победу, губитак, нерешено) као и вредност индивидуалних ризика у случају да играчи одиграју тај меч. Најпре се предвиђа вероватноћа победе, пораза или нерешеног резултата користећи формулу за вероватноћу победе у Ело систему, поменутом у поглављу 1.2.

Индивидуални ризик се израчунава на основу претходна два резултата играча и потенцијалног наредног исхода. С обзиром да су могуће вредности исхода меча Победа (1), Губитак (-1) и Нерешено (0), могу се крерати комбинације ових вредности и свакој од комбинација исхода доделити одређени скалар. Тај скалар биће баш ризик који за играча доноси наредни меч.

Имплементација методе за рачунање укупног ризика се може видети у следећем псеудокоду:

```
predvidi_ukupni_rizik(i1, i2):
    rizik_i1_pobeda  = predvidi_individualni_rizik(i1, ako pobedi)
    rizik_i1_gubitak = predvidi_individualni_rizik(i1, ako izgubi)
```

```
rizik_i2_pobeda  = predvidi_individualni_rizik(i2, ako pobedi)
rizik_i2_gubitak = predvidi_individualni_rizik(i2, ako izgubi)

p_pobede_i1, p_poraza_i1 = predvidi_pobedu_Elo(i1, i2)

ukupan_rizik = p_pobede_i1·(rizik_i1_pobeda + rizik_i2_gubitak)
             + p_poraza_i1·(rizik_i1_gubitak + rizik_i2_pobeda)

vrati ukupan_rizik
```

Вероватноће добијене Ело системом се користе у следећем израчунавању ризика:

- за победу играча p_i, ризик се рачуна као вероватноћа победе p_i помножена са збиром ризика да p_i победи и ризика да p_j изгуби;
- за изједначен резултат, ризик се поставља на нулу;
- за пораз играча p_i, ризик се рачуна као вероватноћа пораза p_i помножена са збиром ризика да p_i изгуби и ризика да p_j победи.

Као коначан резултат, метода *predict_ total_ churn()*, враћа укупни ризик од неангажованости између играча p_i и p_j.

У табели 3.2 представљене су повратне вредности методе за предвиђање индивидуалног ризика (*predict_ individual_ churn()*) за прослеђене претходне исходе играча и потенцијални наредни меч. Различите комбинације исхода имају додељене адекватне ризике.

Када се израчуна вредност промељиве *churn_ risk* (ризик од неангажованости) у граф се додаје грана која као атрибуте има: играче p_i и p_j, тежину гране која представља ризик од неангажованости (*churn_ weight*) и тежину гране која представља вероватноћу да играчи остану ангажовани (*retain_ weight*). Пошто вредност укупног ризика припада интервалу *[0,2]*, атрибуту гране *churn_ weight* се додељује израчуната вредност ризика од неангажованости *churn_ risk*, а атрибуту (*retain_ weight*) се додељује вредност *2 - churn_ risk*.

Табела 3.2: Индивидуални ризик од неангажованости базиран на претходним исходима мечева

претходни исходи	**ризик од неангажованости**
[1, 1, 1]	0.37
[1, 1, -1]	0.49
[1, -1, 1]	0.46
[-1, 1, 1]	0.43
[-1, 1, -1]	0.37
[-1, -1, 1]	0.27
[1, -1, -1]	0.56
[-1, -1, -1]	0.61
подразумевани ризик: 0.5	

3.3 Опис резултата

У поглављу 3.2 је описан ток извршавања симулације приликом које се добијају резултати алгоритама, док је у овом поглављу приказан начин за интерпретацију добијених резултата.

Мере успешности алгоритма

Мера успешности алгоритма за упаривање омогућава квантификовање и оцењивање његових резултата. Циљ је испитати перформансе предложених алгоритама и утврдити њихову ефикасност у решавању проблема упаривања. Ова мера пружа објективан начин за поређење и анализу алгоритама на основу њихових резултата.

Упоређивање стратегија врши се на основу различитих критеријума, као што су просечно задовољство свих играча који су учествовали у свим одиграним партијама или удео играча који ће остати ангажовани и након меча. У овом раду су алгоритми поређени на основу оба принципа, међутим принцип ангажованости је пружио значајније резултате.

Задовољство корисника

Мера успешности алгоритма се може добити посматрањем задовољства корисника (енг. *satisfaction*), што одражава колико је корисницима алгоритам испунио њихова очекивања и потребе. Упоређивање стратегија је најпре из-

вршено на основу групне оцене задовољства свих играча који су учествовали у свим одиграним партијама.

С обзиром да су неки од алгоритама креирани да уравнотеже партије или претпоставе мечеве којима би играчи били задовољни, циљ је видети колико су били добри у томе. Када се за критеријум успешности одабере степен задовољства корисника, полази се од следећих претпоставки:

- играч је задовољан ако је играо изазовну партију и победио је, тј. имао је јачег противника;
- играч није превише незадовољан ако је играо изазовну партију и изгубио је, тј. имао је јачег противника;
- играч није превише незадовољан ако је играо неизазовну партију и победио је, тј. имао је слабијег противника;
- играч је незадовољан ако је играо неизазовну партију и изгубио је, тј. имао је слабијег противника.

За потребе рачунања индивидуалног задовољства при одиграној партији направљена је функција *calculate_satisfaction()* која врши израчунавање на основу неколико фактора. Као улазне вредности метода има инидикатор победе *is_winner*, ранг играча за ког се рачуна сатисфакција *player_rank*, ранг његовог противника у одиграној партији *opponent_rank* и тренутна максимална разлика у рангу између играча у одиграним партијама *max_rank_diff*. Псеудокод приказан у наставку представља методу за рачунање индивидуалног задовољства играча, а на слици 3.2 се могу видети неки од репрезентативних резутата позване методе.

```
izracunaj_zadovoljstvo(pobednik, rang_igraca,
                       rang_protivnika, maksimalna_razlika_rangova):

    maksimalno_zadovoljstvo = 1000
    razlika_rangova = rang_igraca - rang_protivnika
    tezina_meca = razlika_rangova / maksimalna_razlika_rangova
    neutralno_zadovoljstvo = maksimalno_zadovoljstvo / 2

    ako je igrac za kog se racuna zadovoljstvo pobednik:
        ako je tezina manja od nule:
            zadovoljstvo = neutralno_zadovoljstvo·(1 + |tezina|)
        inace:
            zadovoljstvo = neutralno_zadovoljstvo·(1 - |tezina|)
    inace:
        ako je tezina veca od nule:
            zadovoljstvo = neutralno_zadovoljstvo·(1 - |tezina|)
        inace:
            zadovoljstvo = neutralno_zadovoljstvo·(1 + |tezina|)

    vrati zadovoljstvo
```

Слика 3.2: Индивидуална задовољства играча

Максимално задовољство је постављено на хиљаду, а минимално је нула. Дакле, задовољство корисника може варирати у интервалу *[0,1000]*. Затим се израчунава разлика у рангу између играча и противника. Ова разлика се чува у променљивој *rank_ diff*. Следећи корак је израчунавање тежине (*difficulty*) на основу разлике у рангу. Ова вредност се рачуна као количник разлике у рангу између тренутног пара играча и максималне до сада забележене разлике у рангу између два играча. Тежина се користи како би се одредила релативна тежина меча и може имати вредности у интервалу *[0,1]*.

Након тога следи провера да ли је играч победник или губитник. Ако је вредност *is_ winner True*, то значи да је играч за ког се рачуна сатисфакција победник, а у супротном, играч је изгубио меч.

У зависности од улоге (победник или губитник) и тежине меча, израчунава се задовољство корисника. Вредност сатисфакције се рачуна као производ средине интервала и фактора који се користи да ојача или ослаби задовољство на основу тежине меча. Тај фактор је $1 \pm$ тежина меча *difficulty*, при чему се у формули користи апсолутна вредност *difficulty* како би се осигурао позитиван фактор у рачуну.

На слици 3.2 могу се видети неки од резултата имплементиране методе, примера ради о екстремном незадовољству играча који је као доста јачи победио слабијег играча или пак о екстремном задовољству играча који је као доста слабији победио јачег играча. Такође се може видети да играч није много незадовољан ако је изгубио од јачег, или победио слабијег.

Ова мера се у почетним итерацијама алгоритма показала као приступ који даје интересантне податке о појединачном задовољству корисника различтим алгоритмима упаривања, међутим у погледу укупног просечног задовољства даје готово исте реултате за све стратегије упаривања.

Ангажованост

Мера успешности алгоритма се може изразити кроз број задржаних играча. Тај број је удео укупног броја играча који су упарени а након одигране партије имају тенденцију да наставе да играју, тј. неће имати период одсуства.

У свакој одиграној партији за одређени алгоритам упаривања рачуна се проценат задржаних играча. Када се над скупом играча позове одређени алгоритам за упаривање, резултат тог позива је листа парова играча. Квалитет добијеног резултата може се закључити из укупне тежине свих грана у упаривању. Из комплетног графа играча који је креиран при инстанцирању класе *Game* налазе се информације о тежинама грана између свака два чвора у графу.

Пошто су познате тежине грана између свака два играча (атрибути гране *retain_ weight* и *churn_ weight*), итерира се кроз скуп парова који су резултат алгоритма. У свакој итерацији на акумулативну суму се додаје атрибут гране који представља вероватноћу да ће играчи остати ангажовани *retain_ weight*.

Мера успешности алгоритма је управо та акумулативна сума, а у наредној секцији су приказани конкретни резултати за покренуту симулацију.

Резултати

У табели 3.3 су приказани резултати извршавања пет предложених алгоритама. За изабране контролне параметре, алгоритам и број играча, израчунат је проценат задржаних играча. На основу процената задржаних играча може се оценити успешност алгоритма. Број покретања алгоритма са задатим бројем играча је петсто. У свакој рунди, покреће се сваки од пет алгоритама над скупом играча кардиналности n. Вредност $n \in \{50, 100, 150, 200, 250, 300\}$. Алгоритми чије перформансе су испитане: алгоритам који упарује у редоследу доласка играча (*FIFO*), алгоритам који упарује на основу ранга играча (*SB_R*), алгоритам који упарује на основу ранга играча (*SB_WR*), алгоритам који који максимизује неангажованост играча (*MAX_DMM*) и алгоритам који који максимизује ангажовање играча (*MAX_EMM*).

Табела 3.3: Поређење алгоритама упаривања према броју задржаних играча

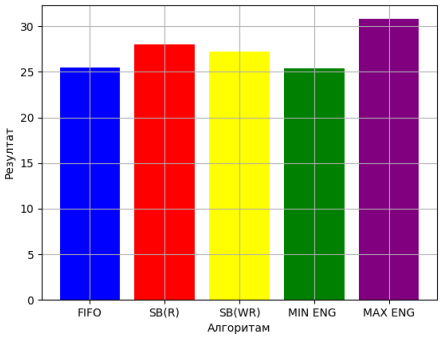
	FIFO	***SB_R***	***SB_WR***	***MAX_DMM***	***MAX_EMM***
n=50	0.51	0.5606	0.5438	0.5076	**0.616**
n=100	0.51	0.5526	0.5481	0.5094	**0.618**
n=150	0.51	0.5494	0.5309	0.5066	**0.618**
n=200	0.5095	0.5629	0.5384	0.508	**0.6171**
n=250	0.51	0.5588	0.5375	0.507	**0.6177**
n=300	0.5066	0.573	0.5366	0.5036	**0.6145**

У почетним итерацијама извршавања алгоритама, може се приметити драстична разлика у успешности стратегија. Како се број покретања повећава, вредности се све више упросечавају.

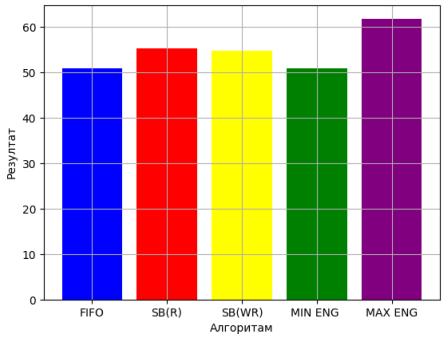
На дијаграму 3.3 може се видети визуелни приказ резултата алгоритама извршених у петсто итерација над скуповима играча различите кардиналности. Зелени стуб, који представља алгоритам који максимизује неангажовање има најнижи степен задржаних играча, заједно са *FIFO* алгоритмом који даје приближно исте резултате. Најуспешнији, чак и на малом узорку играча, је алгоритам који максимизује ангажовање, који над скупом од триста играча задржи у просеку чак тридесет три играча више него претходно поменути алгоритам. Међу алгоритмима заснованим на редовима, најбоље се показао

алгоритам који сортира играче по вештинама (означен црвеном бојом на дијаграму). Независно од броја играча који се упарују, алгоритам *MAX_EMM* задржава приближно 11% више играча него *MAX_DMM*.

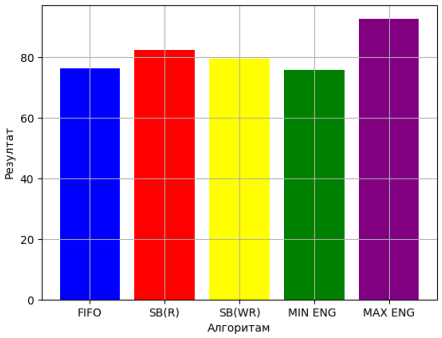
На представљеним дијаграмима може се уочити тренд који прате алгоритми. Без обзира на кардиналност скупа играча на којим се позива упаривање, успешност алгоритама има исти поредак, и то у следећем редоследу почевши од најбољег: *MAX_EMM*, *SB_R*, *SB_WR*, *FIFO*, *MAX_DMM*.



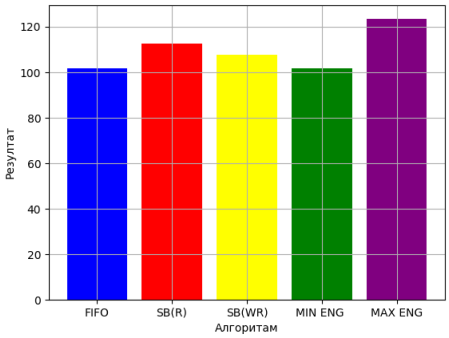
(a) Поређење над скупом од 50 играча



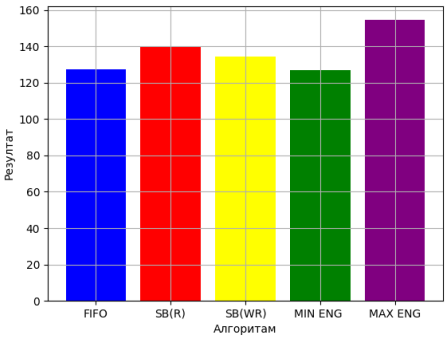
(b) Поређење над скупом од 100 играча



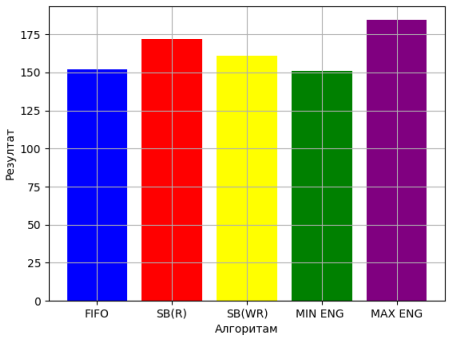
(c) Поређење над скупом од 150 играча



(d) Поређење над скупом од 200 играча



(e) Поређење над скупом од 250 играча



(f) Поређење над скупом од 300 играча

Слика 3.3: Дијаграми поређења просечног рада алгоритама над скуповима играча кардиналности X, за $X \in \{50, 100, 150, 200, 250, 300\}$

Глава 4

Апликација за симулирање упаривања

У овом поглављу биће описана употреба алгоритама за упаривање на клијент-сервер веб апликацији. Апликација симулира упаривање играча и одигравање партија. Играчи приступају игри на веб страни, логују се и траже противника. Када се упаривање обави на серверској страни играч бива преусмерен на страну у којој се може покренути партија.

4.1 Преглед коришћених технологија

За потребе развоја веб апликације коришћени су:

- *Python* програмски језик;
- *Django framework* [2] – веб развојни оквир отвореног кода написан у програмском језику *Python*;
- *SQLite* [3] – база података написана у програмском језику *C*.

Django је бесплатни развојни оквир отвореног кода који прати архитектуру *MVC* (енг. *Model-View-Controller*). Ова архитектура, састоји се од модела (енг. *model*) који представља базу података, погледа (енг. *view*) који представља кориснички интерфејс и контролера (енг. *controller*) који управља логиком апликације. *Django* нуди многе уграђене функционалности као што су аутентификација корисника, управљање сесијама и подршка за рад са базама података. Такође подржава концепт *URL* (енг. *uniform resource*

locator) рутирања, што олакшава дефинисање рута и њихово мапирање на одговарајуће функционалности унутар логике апликације [2]. Садржи објектно релационо мапирање (енг. *Object Relational Mapping – ORM*) које посредује између *Python* класа (модела) и базе података.

У оквиру *Django* апликације модели се дефинишу унутар датотеке *models.py*. Коришћењем система миграција (енг. *migrations*) и *ORM* механизма на основу описаних модела биће креиране релационе табеле унутар базе података. Миграције представљају начин пропагирања промена модела (додавање поља, брисање модела, креирање модела, итд.) у шему базе података. Миграције су дизајниране да буду углавном аутоматске, али потребно је и експлицитно извршити неку од миграција.

Контролери се дефинишу као методе које прихватају *HTTP* захтеве и налазе се унутар датотеке под називом *views.py* што представља неконзистентност због самог имена датотеке, али сами контролери као повратну вредност имају погледе. Коришћењем технике рутирања одговарајући контролери ће бити позвани када корисник пошаље *HTTP* захтев коришћењем унапред дефинисаних *URL* рута, које се дефинишу унутар датотеке *urls.py* [2]. Сврха контролера је да на основу одређене логике врате кориснику одговарајући поглед.

Контролери као повратну вредност имају погледе, а сам поглед описан је коришћењем шаблона (енг. *template*). Шаблон је датотека у којој се описује изглед веб странице, која је најчешће у *.html* формату. Веб страницама се могу прослеђивати одређени параметри које треба динамички приказати као садржај саме странице.

Када се користе заједно, *Python* и *Django* чине алат за развој веб апликација. *Django* пружа оквир и конвенције за структуирање апликације, док *Python* пружа флексибилност и језик за имплементацију логике и манипулацију подацима. *Python* се користи за писање *Django* апликација, где се дефинишу модели, погледи, рутирање, логика и остале компоненте апликације. *Django* оквир затим користи *Python* код како би извршио све потребне операције за генерисање *HTML*-а, управљање базом података, обраду корисничких захтева и приказ резултата корисницима.

SQLite база података је посебно погодна за мање апликације, јер не захтева посебан сервер и може се интегрисати директно у апликацију. То значи да је база података једноставна за постављање и употребу, што олакшава и

убрзава развој апликације. Кроз коришћење *SQLite* базе података, веб апликација може ефикасно чувати информације о играчима, њиховим профилима, историјату мечева, статистикама и другим релевантним подацима. Када се креира *Django* пројекат, добија се празна *SQLite* база података под називом *db.sqlite3*. Као што је већ поменуто, дефинисани модели у *Django* апликацији биће креирани као табеле унутар ове базе података. Када се опише модел у датотеци *models.py*, креирање табеле у бази података се врши следећим командама [3, 2]:

- *python manage.py makemigrations* - креира миграционе датотеке у којима је описано на који начин ће бити креиране табеле на основу дефинисаних модела. Миграционе датотеке прате промене самих модела и омогућавају лако креирање и ажурирање табела.
- *python manage.py migrate* - када су миграционе датотеке креиране, потребно је описане кораке унутар тих датотека применити, чему и служи наведена команда.

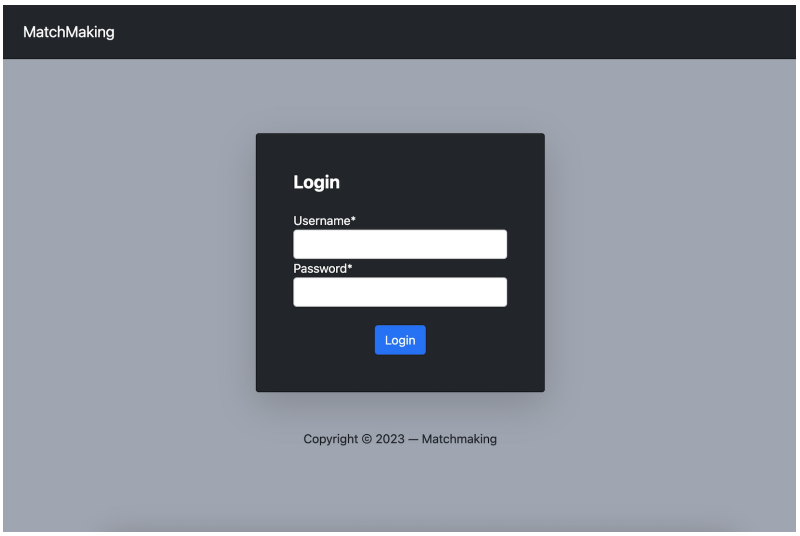
Основни кораци кроз које пролази *Django* веб апликација су следећи:

- претраживач захтева *URL*;
- *Django* прима *URL*;
- проверава датотеку *urls.py*;
- позива поглед који одговара *URL*-у;
- поглед, који се налази у датотеци *views.py*, проверава релевантне моделе;
- модели се учитавају из датотеке *models.py*;
- поглед шаље податке одређеном шаблону;
- шаблон садржи *HTML* и заједно са подацима враћа готов *HTML* садржај назад претраживачу.

4.2 Опис и демонстрација апликације

Када корисник приступи *URL*-у веб апликације, *Django* приказује одговарајући поглед, у овом случају форму за пријаву. Приказ ове форме се може

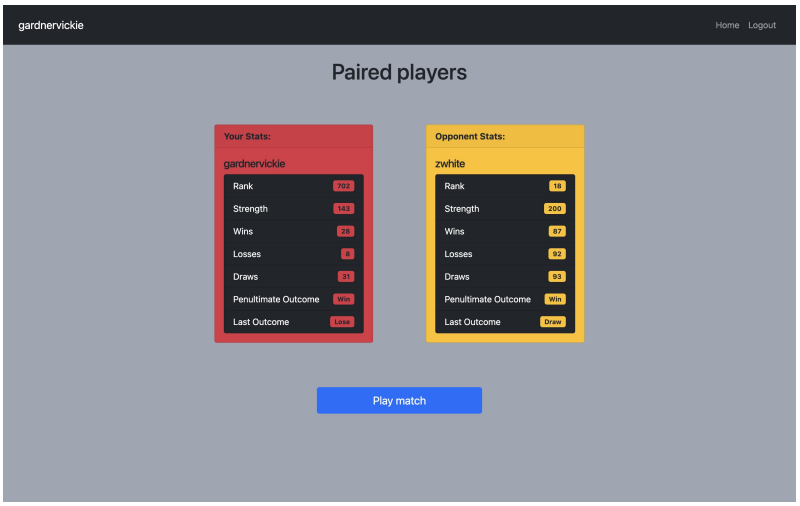
видети на слици 4.1. Корисник уноси своје корисничко име и лозинку, а затим кликом на дугме *Login* серверу се шаље *POST* захтев. Сервер прима захтев и аутентификује корисника путем корисничког имена и лозинке. Уколико се аутентификација успешно обави, корисник се преусмерава на страницу на којој може затражити упаривање са противником кликом на дугме *find_match*. Тим кликом шаље се *POST* захтев ка серверу, где се играч додаје у *pool* играча који чекају на упаривање. Сваки захтев носи информацију о пошиљаоцу, па стога сервер има податке о играчу ког треба да дода у *pool*. Корисник се преусмерава на страницу *match_making* на којој се налази анимација која је визуелни приказ да је упаривање у току и да ће ускоро бити упарен.



Слика 4.1: Страница за пријаву

За имплементацију скупа играча који чекају на упаривање (енг. *player pool*) користи се образац за пројектовање (енг. *design pattern*) Уникат (енг. *singleton*) [22]. Тај образац је погодан јер обезбеђује да класа има само један примерак, с обзиром да је *pool* нека врста управљачког регистра. Он садржи информацију о томе да ли се тренутно одвија упривање, као и методе за дохватање играча који чекају упаривање, додавање и уклањање играча из *pool*-а.

Упаривање се врши над *pool*-ом играча на сваких 10 секунди, али уколико је упаривање и даље у току, захтев за упаривање ће бити игнорисан. Упаривање се врши позивањем алгоритма имплементираног у претходном поглављу. Када се оствари упаривање, веб страница ће пружити приказ индивидуалних



Слика 4.2: Приказ упарених играча спремних да започну меч

карактеристика играча који је затражио упаривање, као и карактеристике противничког играча. Уз то, корисник ће имати могућност да покрене игру кликом на дугме *Play match*, што се може видети на слици 4.2.

Када два играча одиграју меч, информације о мечу се бележе у табелу у бази података.

Глава 5

Закључак и правци даљег развоја

У оквиру овог рада истражене су и имплементиране методе за упаривање играча у играма на Интернету. Стратегије за приступ проблему су користиле две структуре података: редове и графове. Неки од алгоритама су узимали у обзир карактеристике играча, а неке нису. Стратегије са графовима су узеле у обзир и очекивања играча како би максимизовале ангажовање.

Алгоритми за упаривање су имлементирани и тестирани за различите контролне параметре (број играча и алгоритам) и израчунат је проценат задржаних играча који у просеку има тенденцију да остане ангажован. Као мера успешности алгоритама узет је тај проценат. Резултати потврђују да је алгоритам за максимизацију ангажовања применљив као оптимизатор ангажованости. То указује на чињеницу да је ефикасан у побољшању ангажованости играча. Независно од броја играча који се упарују, тај алгоритам задржава приближно 11% више играча него алгоритми који су се најлошије показали. На основу резултата тестирања, може се закључити да су алгоритми показали успешност у следећем редоследу: графовски алгоритам који максимизује ангажовање је постигао најбоље резултате, следи га алгоритам који играче упарује по сличном нивоу вештина, затим алгоритам који упарује по сличном уделу победа у одиграним партијама, и на крају најмање успеха су показали *FIFO* и алгоритам који максимизује неангажованост играча.

Ови резултати сугеришу да графовски алгоритам који максимизује ангажовање има највећи потенцијал за решавање задатка упаривања игача у видео играма, док *FIFO* који редом упарује по редоследу пристизања и графовски

алгоритми који максимизују неангажованост показују најмању успешност у поређењу са осталим алгоритмима. Ови закључци могу бити од значаја за примену алгоритама у пракси и даље истраживање у области стратегија упаривања.

Правци даљег развоја

Додатна мера успешности алгоритама би могла бити базирана на директној повратној информацији од самих играча, тј. њихово стварно задовољство игром. Ово би се могло остварити путем анкета, оцена или коментара које играчи остављају након игре. Ова информација може пружити увид у то како играчи заиста доживљавају упаривање и да ли су задовољни искуством. На основу ових информација, алгоритми могу бити прилагођени како би се побољшала тачност упаривања и повећало задовољство играча.

Процес упаривања би такође могао бити побољшан узимањем у обзир статистике из правих игара. Уместо да се ослањају само на унапред дефинисане вештине или оцене, алгоритми могу користити стварне податке о перформансама играча током игре. Ове статистике могу обухватити резултате партија, постигнуте поене, ефикасност, стил игре и слично. Интеграција ових података у алгоритме може довести до прецизнијег упаривања играча, што резултира већом успешношћу алгоритма.

Библиографија

[1] Complete graph. on-line at: https://www.researchgate.net/figure/Complete-graph-K8-By-exhaustive-inspection-details-omitted-it-turns-out-that-fig1_282182010.

[2] Django Tutorial. https://www.w3schools.com/django/index.php.

[3] SQLite. https://www.codecademy.com/article/what-is-sqlite.

[4] Yan Chen a; Lei Zhang a; Yulu Zhao b; Bing Xu. Implementation of penalized survival models in churn prediction of vehicle insurance. *Journal of Business Research*, 153, 2022.

[5] Regina Bernhaupt. Concepts and methods. In *Evaluating User Experience in Games*, 2010.

[6] Daniel Theuke Christopher Quadflieg. Poliedar. https://matematika.fandom.com/bs/wiki/Poliedar.

[7] Daniel Theuke Christopher Quadflieg. Faker library GitHub, 2022. on-line at: https://fakerjs.dev/guide/.

[8] William Cook and Andre Rohe. Computing Minimum-Weight Perfect Matchings. *INFORMS Journal on Computing*, 11, 1999.

[9] ©Copyright NetworkX Developers. NetworkX - Network Analysis in Python - documentation, 2004-2023. on-line at: https://networkx.org/documentation/stable/tutorial.html.

[10] Stefan Hougardy Doratha E. Drake. A simple approximation algorithm for the weighted matching problem. *Humboldt University of Berlin*, 5, 2002.

[11] L E Atlas E A Riskin; R Ladner; R Y Wang. Index assignment for progressive transmission of full-search vector quantization. IEEE Trans Image Process. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 6, 1994.

[12] Jack Edmonds. Maximum Matching and a Polyhedron With O,1-Vertices1. *JOURNAL OF RESEARCH of the Na tional Bureau of Standards-B. Mathematics and Mathematical Physics*, 6, 1964.

[13] Arpad E. Elo. Second edition. In *The rating of chessplayers, past and present*, 1978.

[14] JASON FERNANDO. TECHNICAL ANALYSIS > TECHNICAL ANALYSIS BASIC EDUCATION: Moving Average (MA): Purpose, Uses, Formula, and Examples, 2023. on-line at: `https://www.investopedia.com/terms/m/movingaverage.asp`.

[15] Harold Gabow. A scaling algorithm for weighted matching on general graphs. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, 1985.

[16] HAROLD NEIL GABOW. IMPLEMENTATION OF ALGORITHMS FOR MAXIMUM MATCHING ON NONBIPARTITE GRAPHS. *Stanford University ProQuest Dissertations Publishing*, 24, 1974.

[17] ZVI GALIL. Efficient Algorithms for Finding Maximum Matching in Graphs. *Department of Computer Science, Columbia University, New York, N. Y., 10027 and Tel-Aviv University, Tel-Aviv, Israel*, 16, 1986.

[18] Mark E. Glickman. Parameter estimation in large dynamic paired comparison experiments. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 48, 1999.

[19] Thore Graepel and Ralf Herbrich. Ranking and Matchmaking - How to rate players' skills for fun and competitive gaming. *Game Developer Magazine, Microsoft Research Cambridge*, 20, 2006.

[20] DAVID R. HUNTER. MM ALGORITHMS FOR GENERALIZED BRADLEY–TERRY MODELS. *Institute of Mathematical Statistics, Pennsylvania State University*, 23, 2004.

[21] Dirk Van den Poel Kristof Coussement. Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques. *Expert Systems with Applications*, 34, 2008.

[22] Sasa Malkov. OOP - C++ kroz primere, 2007. on-line at: `http://poincare.matf.bg.ac.rs/~sasa.malkov/files/res/C++%20kroz%20primere%20-%20Sasa%20Malkov%20-%202007.pdf`.

[23] National. Standard deviation, 2023. on-line at: `https://en.wikipedia.org/wiki/Standard_deviation`.

[24] C.N.K. Osiakwan and S.G. Akl. The maximum weight perfect matching problem for complete weighted graphs is in pc. In *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing 1990*, 1990.

[25] Chih-Jen Lin Tzu-Kuo Huang, Ruby C. Weng. Generalized Bradley-Terry Models and Multi-Class Probability Estimates. *Journal of Machine Learning Research 7*, 31, 2006.

[26] PAUL C. H. ALBERS HAN DE VRIES. A simple approximation algorithm for the weighted matching problem. *Ethology and Socio-ecology Group, Utrecht University*, 7, 2002.

[27] Ning Lu; Hua Lin; Jie Lu; Guangquan Zhang. A Customer Churn Prediction Model in Telecom Industry Using Boosting. *IEEE Transactions on Industrial Informatics*, 10, 2014.

[28] Chen Zhengxing. EOMM: An Engagement Optimized Matchmaking Framework. *International World Wide Web Conference Committee*, 16, 2017.

[29] Snjólfur Ólafsson. Weighted Matching in Chess Tournaments. *The Journal of the Operational Research Society*, 8, 1990.

[30] Весна Маринковић и Миодраг Живковић. Конструкција и анализа алгиритама 1, 2020. on-line at: `http://poincare.matf.bg.ac.rs/~vesna.marinkovic/kaa/kaa.pdf`.

[31] Весна Маринковић и Миодраг Живковић. Конструкција и анализа алгиритама 2, 2020. on-line at: `http://poincare.matf.bg.ac.rs/~vesna.marinkovic/kaa2/kaa2.pdf`.

Биографија аутора

Александра Никшић рођена је 6. марта 1997. године у Чачку, где је завршила Основну школу „Вук Караџић”. Гимназију у Чачку завршила је 2016. године, на природно-математичком смеру. Математички факултет у Београду уписује исте године. Дипломирала је као студент на смеру Информатика 2020. године, након чега уписује мастер студије на истом смеру. Године 2021. обавља праксу у компанији *Mad Head Games*, а 2022. се запошљава у компанији *Telesign* као софтверски инжењер.