Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Šemrl

Igranje mlina z drevesnim preiskovanjem Monte Carlo

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Branko Šter

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Računalniški programi postajajo vedno boljši v igranju iger. V mnogih igrah že so prekosili človeka, celo v šahu. V diplomski nalogi implementirajte igro mlin in agenta za igranje te igre, ki bo uporabljal metodo drevesnega preiskovanja Monte Carlo, ki je ena od priljubljenih metod za implementacijo agentov za igranje iger. Ovrednotite agenta in ocenite, ali se lahko uspešno kosa s človekom.

Zahvaljujem se vsem, ki so mi kakorkoli pomagali pri izdelavi diplomskega dela. Še posebej se zahvaljujem mentorju prof. dr. Branku Šteru, za vso pomoč in nasvete pri izdelavi diplomskega dela.

Zahvaljujem se tudi domačim ter vsem kolegom in sošolcem za vso pomoč in podporo skozi vsa leta študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
MCTS	Monte Carlo tree search	drevesno preiskovanje Monte Carlo
AI	artifical inteligence	umetna inteligenca
UCT	Upper Confidence Bound for Trees	ocena vozlišča z zgornjo mejo zaupanja

Povzetek

Naslov: Igranje mlina z drevesnim preiskovanjem Monte Carlo

Avtor: Miha Šemrl

Pri odločanju veliko vlogo igra naše dosedanje znanje in predvsem razmišljanje o možnostih, ki se nam bodo pojavile v prihodnosti. Za ljudi je dolgoročno planiranje v kompleksnih domenah težavno, saj je med drugim precej zahtevno že za ohranjanje v spominu. Pri odločitvenih problemih, med katere spadajo tudi igre, želimo izvajati take akcije, ki nam bi naj prinesle čimvečjo možnost uspeha. Ena od takšnih metod je drevesno preiskovanje Monte Carlo. Ker je ta način razmišljanja v večini primerov za človeka pretežak, prihaja do vse večje uporabe v računalniškem svetu, saj se dobro izkaže pri igrah z velikim vejitvenim faktorjem.

V diplomski nalogi je metoda implementirana in prikazana njena uporaba v igri mlin, kjer je število potekov igre zelo veliko in je zato raziskovanje v prihodnost še bolj težko.

Ključne besede: umetna inteligenca, drevesno preiskovanje Monte Carlo, igre, mlin.

Abstract

Title: Nine men's morris using Monte Carlo tree search

Author: Miha Šemrl

In decision making our current knowledge plays a big part, especially when thinking of possibilities that could happen in the future. For humans, long-term planning is hard, since it is, among other reasons, hard to remember. In decision-making problems, which include also games, we want to perform such actions that most probably lead to success. One of such methods is Monte Carlo tree search. Because it is too hard for humans to use it on their own, we try to implement it using computers, since it is a method that works well on games with large branching factors.

In this thesis we implement the method and show how to use it in a game called Nine men's morris, which is moderately complex and so it is hard to predict it into the future.

Keywords: artifical inteligence, Monte Carlo tree search, games, Nine men's morris.

Poglavje 1

Uvod

Razvoj umetne inteligence, ki bi bila zmožna konkurirati ljudem, je v porastu, saj ljudje vse bolj iščemo rešitve, kako povečati uporabnost strojev in nam s tem olajšati življenje. Če pa želimo, da nam bo umetna inteligenca res lahko konkurirala, bomo morali še veliko postoriti. Z implemetacijo prave metode razmišljanja pa nas lahko že sedaj premaga v marsičem. Vse več pa imamo tudi umetne inteligence v svetu iger in zabave. Tam nam omogočajo, da se računalnik vse bolj približa človeškemu razmišljanju in nam tako predstavlja vedno večji izziv oziroma delovanje agentov deluje vse bolj podobno razmišljanju človeka. Za izboljšanje razmišljanja in odločanja se uporablja veliko metod, včasih tudi kombinacija raznih algoritmov, izdelanih za pomoč pri odločanju, kateri korak naslednji izbrati. Ena izmed takih metod je tudi drevesno preiskovanje Monte Carlo, ki je s prvo zmago umetne inteligence proti človeku v igri Go pokazala, da ima velik potencial.

Moč same metode preiskovanja se pokaže še posebej pri igrah z veliko akcijami, saj bi nam za pregled celotnega poteka igre to porabilo veliko časa in sredstev. Ravno v tem pa ima drevesno preiskovanje Monte Carlo svojo premoč, saj za pregled dogajanja ne pregleda vseh možnosti, vendar samo nekaj naključnih in s tem pridobi oceno moči posamezne akcije. Na podlagi te ocene moči se nato odloča, katero akcijo izbrati, in tako ciljati na čimvečjo možnost uspeha.

Ker je tudi mlin tak tip igre, kjer imamo vsaj na začetku veliko možnih akcij in posledično veliko možnih zaključkov, se zdi MCTS ena izmed boljših izbir za izgradnjo umetne inteligence igre. Cilj je izdelati agenta, ki s pomočjo uporabe drevesnega preiskovanja Monte Carlo izvaja akcije, ki se izkažejo za najmočnejše in tako predstavlja kar največji izziv za nasprotnika.

Poglavje 2

Preiskovalni algoritmi

Preiskovalni algoritmi so algoritmi, namenjeni iskanju podatkov znotraj podatkovnih struktur ali iskanju najboljše naslednje akcije za doseganje želene rešitve problema. Poznamo več različnih algoritmov, ki se razlikujejo po načinu iskanja in učinkovitosti, na katero pa imajo velik vpliv tudi podatki in način, kako jih uporabimo. V večini primerov nam ti algoritmi vrnejo True ali False kot rezultat, ali je bilo to, kar smo iskali, najdeno, lahko pa jih priredimo in nam vrnejo to, kar smo iskali. Njihova učinkovitost se meri s časovno zahtevnostjo, saj ko dostopamo do podatkov oziroma iščemo rešitev, nam je ponavadi cilj, da podatke dobimo v najkrajšem možnem času.

2.1 Tipi algoritmov

Za različne probleme in različne podatkovne strukture poznamo različne tipe algoritmov, saj vedno stremimo k največji učinkovitosti in najhitrejšemu iskanju podatkov oziroma rešitve. Seveda včasih samo en algoritem ni dovolj, zato lahko različne algoritme tudi združujemo.

1. **Algoritmi za iskanje med podatki:**

    - Linearno iskanje:

Linearno iskanje je najosnovnejši in najpreprostejši način iskanja podatkov. Algoritem poteka tako, da se pomika preko elementov v zbirki in vsakega posebej primerja, če je iskani. Celoten postopek traja dokler ne najde iskanega rezultata in tako zaključi iskanje.

- Binarno iskanje:

  Binarno iskanje se izvaja na podatkih, ki so urejenih po velikosti. Izbere se srednji element zbirke in se ga primerja z iskano vrednostjo, če se ne ujema, se za naslednjo zbirko izbere polovico trenutne zbirke, kjer elementi ustrezajo glede na velikost iskanega elementa. Vse skupaj nato ponovimo na novi zbirki in ponavljamo dokler ne najdemo ustrezen element oziroma je zbirka prazna.

2. **Algoritmi za iskanje rešitev problemov:**

   - Genetski algoritmi

     Kot že ime algoritma nakazuje, so pri razvoju uporabili za osnovo genetski zapis. Algoritem spada v skupino evolucijskih algoritmov, kjer kot osnovo za razvojno idejo uporabimo različne principe evolucije. Postopek najprej naključno zgenerira neko rešitev, nato pa z evolucijskimi tehnikami, kot so dedovanje, mutacije in naravna selekcija, to rešitev spreminja, dokler ne pride do najboljše [1]. Zaradi preprostosti lahko več takih mutacij poteka sočasno in tako pohitri iskanje, hkrati pa se tudi hitro zgodi, da iskanje kvalitetne rešitve lahko traja dolgo, saj nikoli ne vemo, katera mutacija je najboljša, in moramo zato izvesti in testirati veliko možnosti. Poleg velike količine možnih rešitev je tu še problem, da koliko je neka rešitev boljša, je mogoče določiti zgolj s primerjanjem z drugimi rešitvami.

   - Preiskovanje grafov:

     Preiskovanje grafov poteka z algoritmi, kjer za iskanje reštve vsako točko problema predstavimo kot vozlišče, dokler ne sestavimo grafa z vsemi povezavami med vozlišči in tako dobimo pregled vseh poti,

ki potekajo od začetka do rešitve. Vsako pot lahko ocenimo s težo in nato s seštevanjem tež posameznih poti najdemo najustreznejšo pot do naše rešitve. Pri tem moramo paziti, da se pri iskanju reštve ne ujamemo v cikel, kjer bi se ponavljale iste poti.

- Drevesno preiskovanje:

  Drevesno preiskovanje je ena izmed oblik preiskovanja grafov, kjer podatke oziroma podprobleme razdelimo v drevo in nato z izbiro različnih vozlišč pridemo do končne rešitve. Algoritem je uporaben tako za iskanje rešitve problemov, kot tudi za iskanje podatkov. (Če želimo uporabiti algoritem za iskanje podatkov, morajo podatki biti ustreznega tipa, saj jih moramo kot pri binarnem iskanju tudi to najprej urediti, šele nato je iskanje učinkovito in hitro.) Iskanje se konča, ko najdemo pravo rešitev, ali ko nam zmanjka ponujenih možnosti. Paziti moramo, da so podatki ves čas urejeni, saj le tako algoritem deluje pravilno. Zaradi vseh omejitev je drevesno preiskovanje podatkov na prvi pogled zelo podobno binarnemu iskanju, zgolj podatki so urejeni v drevo, kar pa je lahko tudi problem, ko zgrajeno drevo ni uravnoteženo in tako pri iskanju podatkov na eni strani lahko traja dlje kot na drugi. Algoritem je veliko bolj uporaben, pri preiskovanju rešitev drevesa, vendar moramo tudi tu paziti, saj vsak problem ni primeren za iskanje rešitve na ta način.

2.2 Drevesni preiskovalni algoritem

Ko drevesni graf uporabimo za iskanje rešitve, ga izdelamo tako, da začetno vozlišče predstavlja naše začetno stranje, vsak možen korak iz začetnega stanja pa predstavlja podvozlišče. Vsakega od podvozlišč izberemo in pregledamo možne korake iz njega ter jih dodamo kot njegova podvozlišča. Tako počasi z dodajanjem vozlišč izdelamo drevo, ki nakazuje vse možne rešitve našega problema. Drevesa lahko preiskujemo na več načinov:

1. **Neinformirani algoritmi:** Pri neinformiranih algoritmih preiskovanje ne uporablja nobenih dodatnih informacij in tako neusmerjeno in sistematično pregleduje celotno drevo.

   - Gradnja v globino:

     Pri gradnji v globino iz posameznega vozlišča vzamemo prvo podvozlišče, dodamo njegova podvozlišča in korak ponovimo, dokler ne pridemo do lista (končnega vozlišča). Iz lista se vrnemo v zadnje vozlišče in ponovimo postopek za preostala podvozlišča. Tako drevo gradimo najprej po eni poti do končnega stanja, komaj nato se odločimo za drugo pot.

   - Gradnja v širino:

     Pri gradnji drevesa v širino iz vozlišča dodamo vsa podvozlišča enega nivoja, vsakemu iz podvozlišč njegova, pri čemer se vračamo nazaj in pazimo, da čez celotno drevo gradimo vozlišča istočasno, brez da bi katerega predhodno zgradili v globino. Tako drevo namesto po vejah gradimo po nivojih.

   - Iterativno poglabljanje:

     Iterativno poglabljanje je kombinacija obeh vrst, kjer grajenje v globino omejimo na določeno število nivojev. Ko pregledamo vse nivoje in ne najdemo ustrezne rešitve, drevo zavržemo, številu nivojev, ki jih želimo pregledati, prištejemo 1 in postopke ponovimo. Vse skupaj ponavljamo, dokler ne najdemo ustrezne rešitve. Tako se izognemo nepotrebnemu iskanju v globino in hkrati se drevo ne širi tako hitro.

2. **Informirani algoritmi:** Informirano ali hevristično preiskovanje med preiskovanjem uporablja hevristično oceno in se glede na njeno vrednost usmerja v bolj obetavno smer. S tem se gradnja drevesa zmanjša, saj se usmerimo le v obetavne smeri, medtem ko smeri, kjer je hevristična ocena manjša, zanemarjamo. Tako se iskanje rešitve pohitri in izboljša. Nekaj primerov:

- Požrešno preiskovanje:

  Algoritem je zaradi svojega delovanja lahko zelo uspešen ali pa popolnoma neuspešen, saj deluje po principu vzemi najboljše, kar imaš in se ne oziraj na ostalo. Delovanje namreč poteka tako, da v danem vozlišču izberemo, v danem trenutku, najbolj ustrezno podvozlišče in se pomaknemo naprej. Pot, ki jo izbere na koncu, je za algoritem najboljša. Zaradi takega delovanja je algoritem uspešen za zelo specifične probleme, kot je izbira denarja, ki ga je potrebno vrniti in kot sito izbira bankovce oziroma kovance, z največjo ustrezno vrednostjo, glede na ostanek, ki ga še mora vrniti [3].

- A*:

  Algoritem A* svojo pot izbira na podlagi seštevkov ocen poznanih vozlišč. Med grajenjem drevesa se ves čas pregleduje ocene posamezne poti, ter vsakič, ko dodamo novo vozlišče, primerjamo novo pot z ocenami poznanih. Med primerjanjem ugotovimo, katera je najbolj ugodna pot v trenutno poznanem drevesu, in nadaljujemo po njej. Tako zgradimo drevo samo okoli najlažjih poti, preostanek pa zanemarimo. Grajenje takega drevesa ima velikokrat visoko ceno pri porabi pomnilnika, vendar je algoritem vseeno eden izmed bolj priljubljenih.

- IDA*:

  IDA* združuje algoritma A* in iterativno poglabljanje, kar pomeni, da drevo pregledujemo na način, ki ga uporablja A*, vendar pregled omejujemo v globino kot pri iterativnem pregledovanju. Tako se v drevesu še bolj osredotočimo samo na najboljše poti, saj med vsako iteracijo pobrišemo trenutno drevo in tako odstranimo nepotrebna vozlišča ter tako zmanjšamo porabo pomnilnika.

- Minimax:

  Algoritem je namenjen iskanju naboljše akcije pri igrah z dvema

igralcema. Vozlišča predstavljajo naslednje možne akcije, vsakemu vozlišču izračunamo moč, ki predstavlja, kako ugodno je za igralca. V osnovi deluje kot algoritem gradnje v globino, le da se, ko pride v točko odločanja najboljše akcije, odloča na podlagi katerega igralca v točki predstavlja oziroma glede na kateremu nivoju se nahajamo. Če v točki, kjer se nahaja, predstavlja igralca, ki išče akcijo, izbere pot, ki vsebuje najmočnejše vozlišče, ko pa je v točki, kjer predstavlja nasprotnika, pa izbira najšibkejše vozlišče. Iz tega načina izbiranja tudi izhaja ime, saj izbira vozlišča z minimalno oziroma maksimalno vrednostjo. Zaradi prioritiziranja gradnje v globino je lahko gradnja drevesa zelo dolga in posledično potratna. Za izboljšanje tega se velikokrat odločimo za omejitev nivojev, ki jih gradimo, in tako zmanjšamo drevo in pohitrimo izgradjo.

Poglavje 3

Drevesno preiskovanje Monte Carlo

Drevesno preiskovanje Monte Carlo (MCTS) je hevristično drevesno preiskovanje, namenjeno iskanju najboljše odločitve oziroma najboljše naslednje akcije, najpogosteje uporabljen pri igrah z dvema igralcema. Za osnovo se uporablja delovanje algoritma drevesnega priskovanja (v primeru iger z dvema igralcema uporabimo Minimax), ki se ga nadgradi z metodo Monte Carlo, ki naključno generira problem iz znanega stanja do končnega stanja in tako pridobi oceno vozlišča. S pomočjo ocene se nato algoritem odloča, katero akcijo naslednjo izvesti oziroma kaj narediti v naslednjem koraku.

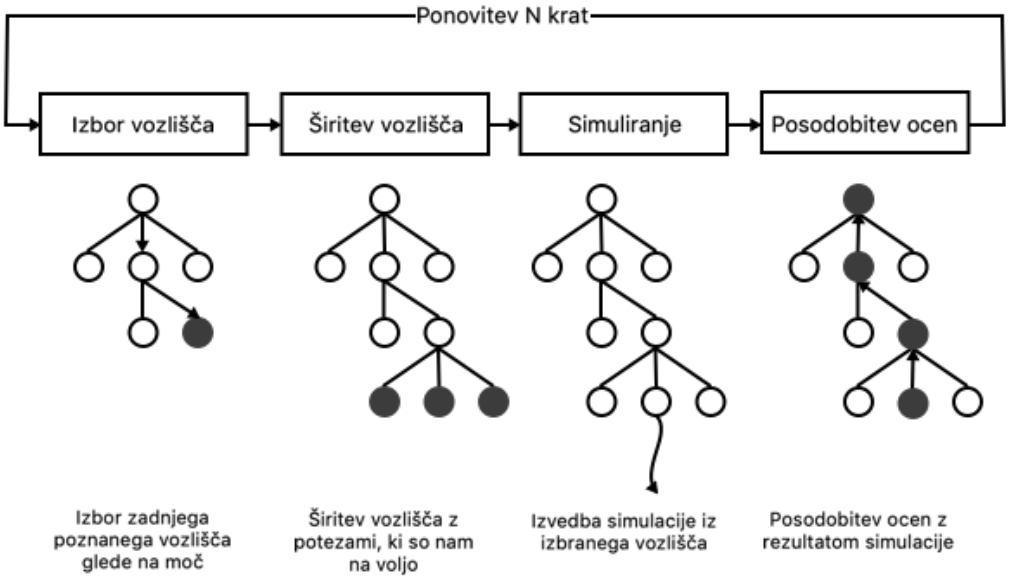
3.1 Metode Monte Carlo

Metode Monte Carlo so metode, ki s pomočjo velikega števila simulacij ali izračunov predvidevajo izid oziroma rezultat problema. V primeru drevesnega algoritma naključno zgenerirajo sestavljanje oziroma potek drevesa od trenutno znanega stanja do končnega stanja in tako pridobijo oceno, koliko možnosti imamo za uspeh, če nadaljujemo po določeni poti. Zaradi naključnega generiranja moramo upoštevati, da se med veliko količino naključno generiranih rešitev najde tudi del rešitev, ki niso ustrezene. Njihova

uporaba je razširjena predvsem v matematiki, fiziki in računalništvu, veliko pa je še področij, kjer se uporaba šele odkriva.

3.2 Delovanje algoritma

Algoritem deluje skozi cikel (Slika 3.1), v katerem izbiramo med znanimi vozlišči, ko pa pridemo do nepoznanega dela, vozlišče razširimo, izvedemo simulacijo in na podlagi rezultata simulacije posodobimo ocene vozlišč.



Slika 3.1: Prikaz delovanja MCTS

1. **Izbor vozlišča:**

   V začetnem vozlišču pričnemo s pregledovanjem podvozlišč, med katerimi izberemo vozlišče z najmočnejšo oceno, ko izbiramo vozlišča za poteze agenta in vozlišče z najslabšo oceno, ko izbiramo vozlišča za poteze nasprotnika. Izbiranje ponavljamo, dokler ne dosežemo vozlišča, ki ali nima ocene, ali nima poznanih podvozlišč. V primeru, da imamo na voljo več vozlišč, med katerimi so tudi vozlišča, ki še nimajo izračunane

ocene, izberemo enega izmed njih, ne glede na oceno ostalih. Tako zagotovimo, da ne bi prišlo do tega, da kako vozlišče ne bi bilo obiskano in s tem zmanjšamo možnost napake. Če pridemo do vozlišča, ki še nima ocene, ga izberemo kot osnovo za simulacijo, v nasprotnem primeru najprej izvedemo širitev vozlišča.

2. **Širitev vozlišča:**

   Širitev vozlišča lahko izvedemo, ko ima vozlišče, v katerem se trenutno nahajamo, že izračunano osnovno moč. V takem primeru pogledamo, katere so možne sledeče akcije in jih kot podvozlišča dodamo zadnjemu vozlišču in enega izmed njih izberemo za osnovo simulacije. V primeru, da širitev ni možna, ker je to končno vozlišče, se namesto simulacije zgodi zgolj posodobitev ocen.

3. **Simulacija:**

   Iz trenutno zadnjega znanega vozlišča poiščemo akcije, ki so nam na voljo in izbermo eno naključno, ali prvo, ki je na voljo. Izbiranje akcij ponavljamo, dokler simulacija ne pride do končnega stanja in tako pridobimo rezultat, ki bi nastal z tem naborom akcij. Po koncu simulacije del dreves, ki je bil zgrajen med simulacijo, zavržemo, ter tako poskrbimo, da nima vpliva na prihodnje izračune oziroma simulacije.

4. **Posodobitev ocen:**

   Po pridobitvi rezultata ga ovrednotimo in rekurzivno posodobimo rezultate vseh vozlišč, izbranih pred simulacijo. Ko posodobimo rezultat in število obiskov v vseh izbranih vozliščih, uporabimo MCTS formulo in vsem znanim vozliščem v drevesu ponovno izračunamo oceno ter tako pridobimo novo stanje.

Celoten postopek ponavljamo, kolikorkrat se za to odločimo, oziroma, dokler ne porabimo vsega časa, namenjenega za izvajanje simulacij. Po koncu vseh simulacij se izračuna moč vozlišč kot razmerje števila obiskov in števila

zmag (moč = zmage/obiski) in se izbere vozlišče z najboljšim razmerjem, kot naslednjo izvedeno akcijo.

3.3 UCT Formula

Za izračun ocene moči posameznega vozlišča se uporabljajo različne formule. Prvo formulo za izračun ocen, imenovano UCT (Upper Confidence Bound for Trees), sta razvila Levente Kocsis in Csaba Szepesvári [2].

Formula (3.1) za računanje ocene moči vozlišč, ki predstavljajo akcijo agenta:

$$UCT = \frac{w_i}{n_i} + c * \sqrt{\frac{\ln N_i}{n_i}} \tag{3.1}$$

Formula (3.2) za računanje ocene moči vozlišč, ki predstavljajo akcijo nasprotnika:

$$UCT = \frac{w_i}{n_i} - c * \sqrt{\frac{\ln N_i}{n_i}} \tag{3.2}$$

- w_i - predstavlja število zmag vozlišča po izvedeni i-ti akciji
- n_i - predstavlja število simulacij z uporabo vozlišča po izvedeni i-ti akciji
- N_i - predstavlja skupno število iteracij po izvedeni i-ti akciji oziroma število iteracij na starševskem vozlišču
- c - predstavlja konstanto preiskovanja, s katero nadzorujemo, kako na oceno drevesa vpliva neobiskanost vozlišča. Višja kot je konstanta, večji del ocene predstavlja neobiskanost in tako poskrbimo, da so vozlišča obiskana enakomerno. Največkrat uporabljena vrednost konstante je $\sqrt{2}$, vendar se lahko za njeno velikost odločimo poljubno, primerno glede na posamezen problem.

  Prvi del formule ($\frac{w_i}{n_i}$) predstavlja moč vozlišča, glede na trenutno stanje po simulacijah, drugi del ($c * \sqrt{\frac{\ln N_i}{n_i}}$), pa predstavlja moč raziskovanja oziroma vpliv neraziskanosti vozlišča. Manj kot je vozlišče raziskano,

večji je rezultat drugega dela in tako poveča celotno oceno vozlišča ter spodbudi k raziskovanju.

Poglavje 4

Igra mlin

4.1 Opis igre

Mlin je namizna strateška igra (Slika 4.1), ki so jo igrali že stari Rimljani. Pri igri poskuša igralec z izvajanjem akcij s svojimi žetoni sestaviti mlin (trije žetoni v vrsti) in tako nasprotniku odvzeti njegove. Zmaga igralec, ki prvi nasprotniku odstrani 7 žetonov in mu tako onemogoči, da bi sestavil mlin.

Slika 4.1: Namizna igra mlin

4.1.1 Pravila igre

Plošča je sestavljena iz štirindvajsetih stičišč povezav oziroma polj, kamor lahko postavimo žetone. Vsak izmed igralcev ima na voljo 9 žetonov, s katerimi z izvajanje različnih potez skozi igro poskušata doseči mlin.

1. **Polaganje žetonov:** Igra se začne s polaganjem žetonov na ploščo. Igralca izmenično polagata žetone na prazna polja. Če med polaganjem kateri izmed igralcev sestavi mlin, lahko odstrani enega izmed nasprotnikovih žetonov.

2. **Premikanje žetonov:** Ko igralca položita vse žetone, se igra premakne v drugo fazo, kjer igralci premikajo svoje žetone in ponovno poskušajo doseči mlin. Posamezen žeton lahko igralec premakne le na sosednje prazno polje. Kot sosednja polja štejejo vsa, do katerih ima trenutno polje žetona izdelano povezavo na plošči.

3. **Skakanje:** Ko igralcu ostanejo samo še trije žetoni, lahko z njimi začne skakati. Skok pomeni premik žetona na poljubno prazno polje, ne glede na povezave, torej tudi, če to polje ni sosednje.

4. **Odstranjevanje:** Če igralec sestavi mlin, lahko izbere enega od nasprotnikovih žetonov in ga odstrani iz plošče. Pri tem mora paziti, saj žetonov, ki sestavljajo nasprotnikov mlin, ne sme odstaniti. Žetone, ki sestavljajo mlin, lahko odstranjuje le v primeru, da ni na voljo nobenega žetona, ki ne sestavlja mlina.

Poleg zmage z odvzemanjem žetonov lahko uporabnik zmaga tudi tako, da nasprotniku onemogoči nadaljnje akcije in ga tako prisili v predajo. V primeru, da se stanje na plošči ponovi, se razglasi remi in igra se konča.

4.1.2 Variacije igre

Poleg verzije igre, ki je uporabljena v nalogi, obstajajo tudi različice z drugačnimi števili žetonov oziroma z drugačno velikostjo plošče.

- Različica s tremi žetoni (Slika 4.2):

  Igra se igra na plošči z devetimi pozicijami (podobno kot Tri v vrsto), kjer igralec poskuša doseči mlin in zmagati. Po koncu polaganja žetonov lahko uporabnik premika žetone kamorkoli želi. Obstaja pa tudi različica, kjer lahko uporabnik žetone premika le na sosednja polja brez preskakovanj [4].

- Različica s šestimi žetoni:

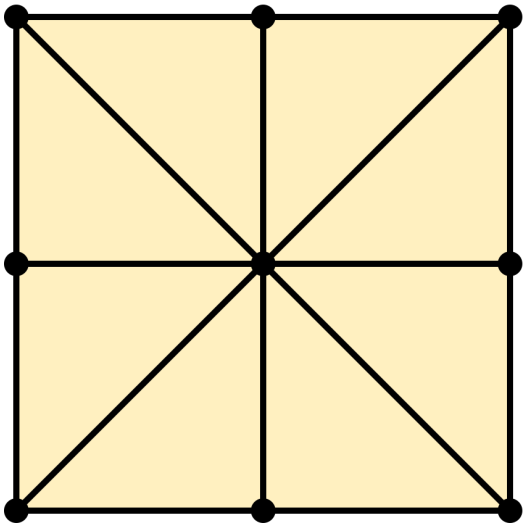
  Za igro s šestimi žetoni se uporabla ploščo z dvema obročema (Slika 4.3), brez diagonalnih povezav. Igra te velikosti je bila najbolj popularna v srednjem veku, vendar je s časom izgubila na popularnosti. Na isti plošči se igra tudi igro s petimi žetoni, če pa dodamo še križ v sredini, se lahko igra igra s sedmimi žetoni [5].

- Različica z dvanajstimi žetoni (Slika 4.4):

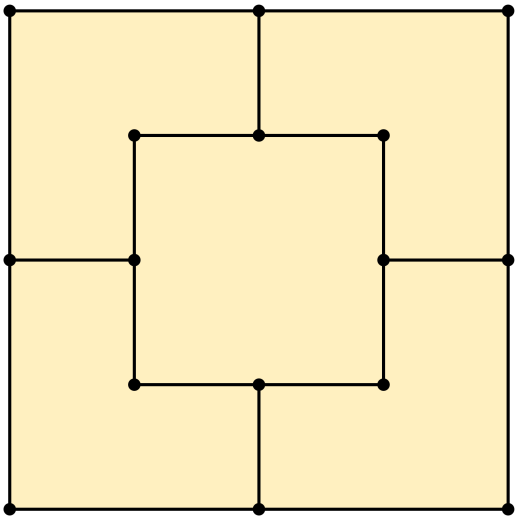
  Največja izmed vseh je različica z dvanajstimi žetoni na igralca. Ker je še vedno na voljo samo 24 pozicij, lahko igralca med polaganjem žetonov popolnoma zapolneta ploščo, brez da bi kdo izgubil kak žeton. V takem primeru je v igri prišlo do izenačenja, saj nobeden od njiju ne more narediti nobenega premika. Na sami plošči pa so poleg sredinske povezave med obroči tudi diagonalne povezave med koti obročev. Ta različica je popularna v Južnoafriški republiki, kjer je poznana pod imenom Morabaraba [6] in je uveljavljena kot šport.

- Različica z desetimi žetoni:

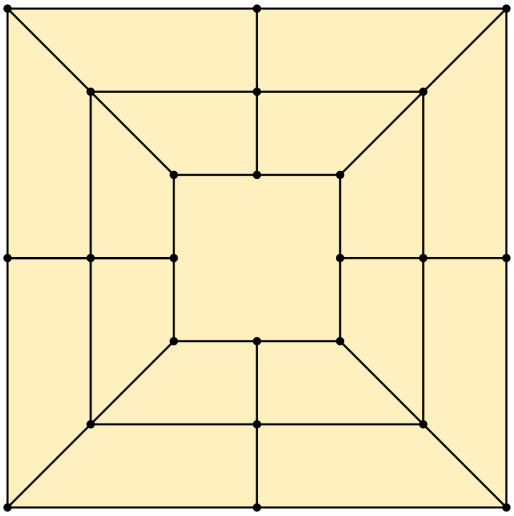
  Poleg osnovnih različic obstaja tudi različica z desetimi žetoni ali Lasker morris [7], avtorja šahista Emanuela Laskerja. V njegovi različici, ki temelji na igri mlin z devetimi žetoni, so premiki dovoljeni že med prvo fazo igre, torej se lahko igralec odloči, ali bo položil nov žeton, ali zgolj premaknil trenutne.



Slika 4.2: Plošča različice s tremi žetoni



Slika 4.3: Plošča različice s šestimi žetoni



Slika 4.4: Plošča različice z dvanajstimi žetoni

4.2 Moja igra

Igra omogoča igranje proti agentu, ki za svoje razmišljanje med izbiranjem naslednje poteze uporablja MCTS. V namen testiranja sem izdelal dve različici igre: različico s tremi žetoni in različico z devetimi. Pri manjši različici sem za iskanje naslednje poteze uporabil samo MCTS, pri večji različici pa se zaradi boljšega delovanja število možnih potez omeji z izbiro ustreznejših.

4.2.1 Opis delovanja

Delovanje programa se deli na tri dele: prvi del skrbi za igranje nasprotnika, drugi za delovanje posameznega žetona in tretji za delovanje agenta oziroma simuliranje igre med iskanjem naslednje poteze.

1. **Nasprotnikov del:**

   Med samo igro moramo paziti, kakšno je stanje na plošči, ter v kateri stopnji je igra. V ta namen sledimo številu žetonov posamezne barve na plošči ter kateri igralec je trenutno na vrsti. Po izvedbi vsakega premika

se igralec zamenja, stanje na plošči se posodobi in, če je naslednji igralec agent, pokličemo del igre, ki skrbi za to. Paziti moramo, ali so premiki igralca dovoljeni, ali je uporabnik sestavil linijo treh žetonov in ali ima uporabnik na voljo še kakšno potezo. Po vsakem premiku, še posebej po odstranitvi žetonov, preverimo ali ima nasprotnik še dovolj žetonov oziroma ali ima na voljo še kak premik in tako preverimo, ali je konec igre.

2. **Delovanje žetona:**

   V tem delu zaznavamo klike na posamezen žeton ter pravilno spreminjamo njegovo stanje glede na stanje igre. Ves čas pa je potrebno paziti, da je klik na žeton dovoljen, saj bi drugače lahko igralec med premikanjem kliknil na mesto, kjer je žeton nasprotnika ali pa na prazno mesto oziroma svoj žeton, med odstranjevanjem.

3. **Agent:**

   Ko izdelujemo agenta, moramo upoštevati, da je to različica igralca, katerega delovanje je potrebno nadzorovati, brez da bi to uporabnik opazil, torej brez vpliva na del, ki skrbi za prikaz na zaslon. Zato sem v ta namen izdelal igro tako, da lahko del, ki nadzoruje agenta, odstranim in igra še vedno ostane delujoča, vendar sta za igranje potrebna dva človeška igralca.

   Celotna enota se deli na več delov, potrebnih za simulacijo, ki skupaj poskrbijo, da lahko agent igra povsem svojo namišljeno igro brez človeškega igralca.

   - Simuliacija:

     Najpomembnejši del predstavlja izvajanje simulacij, ker se za izbiro naslednje poteze izvaja MCTS, za kar potrebujemo dve manjši metodi. Najprej je pomembno, da pravilno izračunavamo moč vozlišča po koncu simulacije.

```
private void CalculatePower (Node node)
 {
  node.power = 0;
  foreach (Node child in node.children) {
   CalculatePower (child);
  }

  if (node.usageNumber > 0) {
   if (node.parent != null) {
    node.power = (node.victories / node.usageNumber)
        + ((int)node.pieceState * (c * (Mathf.Sqrt (
        Mathf.Log (node.parent.usageNumber) / node.
        usageNumber))));
   }
  }
 }
```

Ker je to igra, kjer se igralca izmenjujeta, se v formuli upošteva Minimax algoritem in se tako izmenjujoče prišteva oziroma odšteva del formule, kjer se upošteva moč preiskovanja manj poznanih vozlišč. V ta namen je uporabljen igralec v tem vozlišču, ki je shranjen kot stanje žetona in ima vrednost -1 za človeškega igralca in 1 za vozlišče, ki predstavlja agenta. Izračun moči se izvede nad vsemi vozlišči v drevesu po koncu vsake simulacije in tako skrbi za pravilno izbiro naslednje poteze.

Drugi pomemben del simulacije je izbira pravilnega vozlišča, kjer moramo ravno tako upoštevati, da je osnova algoritem Minimax in se tako izbira najmočnejše vozlišče za AI igralca in najslabše vozlišče za človeškega igralca.

```
public Node SelectStrongest (System.Collections.
   Generic.List<Node> children)
 {
  System.Collections.Generic.List<Node> strongest =
```

```
    new System.Collections.Generic.List<Node> ();
 if (children [0].pieceState == PieceStates.aiPlayer
    ) {
  double maxPower = children [0].power;
  foreach (Node child in children) {
   if (child.power > maxPower) {
    strongest.Clear ();
    maxPower = child.power;
    strongest.Add (child);
   } else {
    if (child.power == maxPower) {
     strongest.Add (child);
    }
   }
  }
 } else {
  double minPower = children [0].power;
  foreach (Node child in children) {
   if (child.power < minPower) {
    strongest.Clear ();
    minPower = child.power;
    strongest.Add (child);
   } else {
    if (child.power == minPower) {
     strongest.Add (child);
    }
   }
  }
 }
 Node strongestNode = strongest [Random.Range (0,
    strongest.Count - 1)];
 return strongestNode;
}
```

Tako z uporabo MCTS iščemo poteze, ki so nam najbolj ustrezne in se na njih osredotočamo. V primeru, ko imamo na izbiro več enakovrednih potez, pa se odločimo z naključnim izborom ene izmed potez v izboru najboljših.

- Iskanje naslednje poteze:

  Delovanje metode za iskanje poteze je razdeljeno na štiri različne dele, odvisno v kateri fazi je igra.

  (a) **Polaganje žetonov:**
  Najprej žetone polagamo na prazna mesta na ploščo, glede na ustreznejše pozicije in poskušamo že med tem sestavljati mlin oziroma ga preprečiti nasprotniku. Ko položimo vse žetone, igra preide v fazo premikanja.

  (b) **Začetek premika žetona:**
  Vsak premik se začne z izbiro vseh žetonov, ki imajo omogočen kakšen premik in izbiro najustreznejšega. Ko si izberemo najustreznejši žeton, si ga moramo zapomniti, saj le tako lahko premik opravimo pravilno.

  (c) **Konec premika žetona:**
  Ko imamo izbran žeton, ki ga bomo premaknili, pregledamo, kakšne so možnosti premika in eno izmed njih izberemo. Pri tem poskušamo sestaviti mlin ali zapreti nasprotnika.

  (d) **Odstranjevanje žetonov:**
  Če igralcu uspe sestaviti mlin, moramo izbrati vse žetone, ki jih nasprotnik lahko odstrani. Pri tem moramo paziti, da ne odstranjujemo žetonov, ki sestavljajo nasprotnikove mline, razen če so to edini žetoni na voljo. Zaradi preprostosti te stopnje te poteze ne ocenjujemo, tako so vse poteze izenačene in se odločitev popolnoma prepusti MCTS.

- Ocenjevanje potez:

  Za zmanjšanje števila možnih potez se med izibiranjem ustre-

znih potez opravi pregled in ocenitev posamezne poteze, ter tako zmanjša nabor ustreznih akcij na nekaj najmočnejših. Npr. med polaganjem žetonov, se kot najboljše poteze oceni poteze, ki bi sestavile mlin, sledijo poteze, ki preprečijo nasprotnikov mlin, ko pa nobena taka poteza ni na voljo, so vse poteze enakovredne. Tako izmed vseh možnih potez izluščimo najmočnejše in se osredotočimo zgolj na njih.

- Premikanje:

  Pri premikanju je potrebno paziti, da je premik dovoljen, saj se skozi igro pravila premikanja spreminjajo. Za pravilen začetek premika je najprej potrebno paziti, da izberemo žeton, ki se lahko premakne. Vsak premik moramo shraniti in tako navidezno igrati igro, saj le tako lahko ustvarimo realno simulacijo in preverimo rezultat naključno generirane igre. Paziti moramo pa tudi, da po koncu vsake simulacije stanje igre povrnemo v začetno stanje in nato med pomikom po drevesu izvajamo poteze, ki so v posameznem vozlišču.

- Preverjanje, ali še obstaja kakšna poteza:

  Pred vsako menjavo igralcev pa moramo preveriti tudi, ali ima nasprotnik še na voljo kako potezo, saj je v nasprotnem primeru igre konec.

4.2.2 Največji problemi

- Določanje, na katere linije vpliva žeton:

  Eden večjih problemov oziroma stvari, na katere moramo paziti, da vsak žeton vpliva na oziroma nanj vplivata dve liniji na plošči. Tako je potrebno najprej določiti, kje se žeton nahaja oziroma katere linije moramo pregledati, ali se v njihovi smeri dogaja, kar nas zanima. Zato je žetone najlažje deliti na te, ki se nahajajo v kotu obročev, ter tiste, ki se nahajajo na linijah, ki povezujejo obroče.

- Zaznavanje mlina:

  Zaznavanje, ali je žeton, ko ga položimo na ploščo oziroma po koncu premika v mlinu, je zelo pomemben del igre, saj je to nekako cilj vseh premikov. Zato je to potrebno izvajati pravilno, pri tem posebno paziti, saj različni žetoni lahko sestavljajo različne mline. Žetoni, ki se nahajajo v kotih, lahko sestavljajo mlin na linijah desno od sebe in linijah levo od sebe, žetoni na povezavah, pa lahko sestavljajo mlin na povezavi med obroči, ter stranici obroča, na kateri se nahajajo.

- Določanje naslednje poteze:

  Iskanje naslednje poteze je zahtevno, ker moramo paziti, kaj se je zgodilo pred njo, kajti če se odločimo za en nepravilen korak, se igra popolnoma spremeni in igra se lahko konča popolnoma drugače, kot bi se morala. Paziti moramo tudi, da izberemo le žetone, ki lahko izvedejo naslednjo potezo.

- Ocenjevanje potez:

  Za zmanjšanje števila možnih potez se ocenjujejo, glede na vrednosti, ki jo predstavljajo igralcu. Tako dobimo poteze, ki so za nasprotnika bolj vredne in se osredotočimo nanje, ter tako odstranimo tiste, ki so „nesmiselne“. Različne poteze imajo različne vrednosti, vendar se v večini primerov uporabi priotiziranje potez, ki sestavijo mlin, sledijo poteze, ki preprečijo nasprotniku sestavo mlina in šele nato sledijo preostale.

Poglavje 5

Pregled delovanja

Za testiranje je bil uporabljen človeški igralec, ki je za igranje uporabljal različne strategije.

Strategija 1:

Naključni izbor poteze, ne glede na stanje na plošči, brez usmeritve.

Strategija 2:

Prioriteta igralca je sestavljanje lastnega mlina, ne glede na postavitev nasprotnika. Igralec ves čas poskuša postavljati žetone poleg svojih oziroma premikati žetone bližje prostoru, kjer bi lahko sestavil mlin.

Strategija 3:

Igralec poskuša preprečiti, da bi agent sestavil lasten mlin. Tako ves čas postavlja žetone poleg nasprotnikovih oziroma v primeru, da agent nima možnosti sestaviti mlina, sestavlja svojega in prednostno odstranjuje nasprotnikove žetone, ki imajo večjo možnost sestave mlina.

Strategija 4:

Mešani izbor potez, izbere se potezo, ki v danem trenutku izgleda najboljša.

Vse strategije so testirane na obeh različicah igre, ter tako v igri, kjer kot prvi začne človeški igralec, ter v igri, kjer prvi začne agent. Vsaka strategija je bila odigrana desetkrat in se preverilo razmerje zmag med igralci.

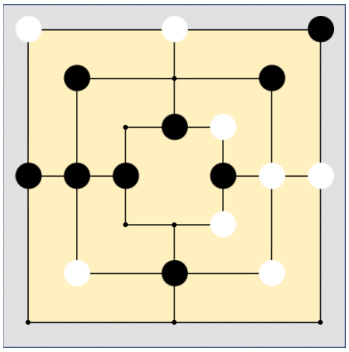
5.1 Različica z devetimi žetoni

Začetni igralec	Strategija 1	Strategija 2	Strategije 3	Strategija 4
Človek	Človek: 0 Agent: 10	Človek: 7 Agent: 3	Človek: 10 Agent: 0	Človek: 5 Agent: 5
Agent	Človek: 0 Agent: 10	Človek: 6 Agent: 4	Človek: 10 Agent: 0	Človek: 10 Agent: 0

Tabela 5.1: Rezultati različice z devetimi žetoni

Pri različici z 9 žetoni (Slika 5.1) ima velik vpliv na igro agenta prioritiziranje potez, saj tako agent hitreje išče ustreznejše poteze in tako otežuje igro nasprotniku. V primeru, da nasprotnik naslednjo potezo izbira naključno, nima možnosti za uspeh, saj prioritizacija potez v kombinaciji z MCTS hitro sestavlja mline. Tako AI že med polaganjem žetonov sestavlja mline in uporabniku odstranjuje njegove žetone in tako hitro doseže zmago.

Ko pa proti AI igra igralec, ki ima določeno strategijo, se rezultati razlikujejo glede na strategijo. Kot vidimo po rezultatih v Tabeli 5.1, se AI boljše obnese kadar se brani oziroma zapira človeškega igralca. Velik vpliv na ta rezultat ima algoritem, ki ocenjuje moč potez, saj je v začetku usmerjen v preprečevanju zmage človeškega igralca in ga tako, v primeru, ko algoritem napada, lahko hitro oslabimo z zapiranjem njegovih potez.



Slika 5.1: Igra z devetimi žetoni

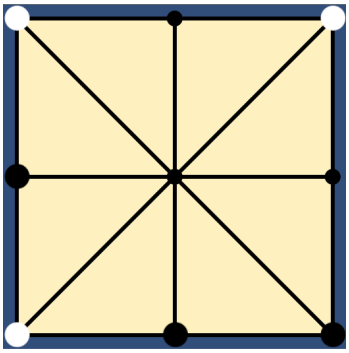
5.2 Različica s tremi žetoni

Delovanje manjše različice (Slika 5.2) je drugačno kot delovanje večje različice. Za delovanje agent uporablja zgolj MCTS brez ocenjevanja in prioritiziranja potez. Vseeno pa je za boljše delovanje potrebno izdelati manjše prilagoditve, saj med igro spreminjam paramtere, ki sestavljajo formulo UCT. Problem nastane, ker agent po normalnem delovanju sili k sestavljanju svojega mlina, zato je potrebno poskrbeti, da najprej zapira nasprotnika, po koncu polaganja pa preide k sestavljanju svojega mlina. Žal je, zaradi majhnosti plošče, težko prepreči igralcu sestaviti mlin oziroma tako postaviti žetone, da bi lahko agent sam sestavil mlin pred nasprotnikom.

Začetni igralec	Strategija 1	Strategija 2	Strategije 3	Strategija 4
Človek	Človek: 1 Agent: 9	Človek: 5 Agent: 5	Človek: 9 Agent: 1	Človek: 9 Agent: 1
Agent	Človek: 0 Agent: 10	Človek: 3 Agent: 7	Človek: 8 Agent: 2	Človek: 9 Agent: 1

Tabela 5.2: Rezultati različice s tremi žetoni

Po rezultatih v Tabeli 5.2 vidimo, da tudi manjša različica, kljub drugačnemu algoritmu, deluje podobno kot večja. Tudi pri tej opazimo, da kadar človeški igralec napada in pri tem ne uporablja obrambnih potez, je agent bolj uspešen, sej ravno tako prioritizira zapiranje nasprotnika. Kljub temu, da po položitvi vseh žetonov algoritem spremeni svoje delovanje v bolj napadalno, je v veliki nemoči kadar nasprotnik napada, saj ga zaradi majhnosti plošče težko zaustavi. Še posebej težko pride do zmage, kadar nasprotnik hkrati izvaja poteze, ki skrbijo za sestavljanje mlina in sočasno zapirajo agenta.



Slika 5.2: Igra s tremi žetoni

5.3 Opažanja

Pri delovanju agenta lahko opazimo, da je potrebno nekako prioritizirati poteze, saj se drugače prehitro osredotoči na napačno potezo. Na to ima najbrž manjši vpliv tudi naključno izbiranje izenačenih potez, saj tako hitro lahko pride do izbora poteze, ki z vidika človeškega igralca ni najboljša poteza. Velik vpliv ima tudi velikost plošče, saj se pri manjših potezah hitro lahko zgodi, da si agent med simulacijo zgenerira igro, po kateri kljub slabi začetni potezi na koncu z naključno generacijo pride do zmage.

Drugi kriterij, ki vpliva na iskanje uspešne poteze, predstavlja ocenjevanje zmage oziroma poraza, saj se v primeru, da je zmaga predobro ocenjena, MCTS težje osredotoči na slabše raziskana vozlišča. Poleg teh spremenljivk pa je potrebno paziti tudi na število simulacij, saj lahko premajhno število simulacij pri igrah z velikim številom potez, kot je večja različica mlina, povzroči, da simulacije ne raziščejo dovolj in tako slabo oceni moč vozlišča. Paziti pa moramo tudi, da število simulacij ni preveliko, saj se tako, v primeru, da je bilo izbrano slabo vozlišče, ki je vseeno uspelo, preveč osredotočimo na samo eno vozlišče in mu tako povečujemo moč.

Iz rezultatov hitro opazimo, da ima velik vpliv na delovanje tudi usmerjanje izbora potez agenta. Vidimo, da je bolj uspešen v primerih, ko nasprotnik napada, saj je algoritem bolj usmerjen v obrambne poteze oziroma v poteze, ki preprečujejo človeškemu igralcu sestavo mlina. Tudi v primeru, ko igra

uporablja zgolj MCTS za delovanje, pride do usmerjanja preko vrednosti v formuli za računanje moči oziroma različno vrednotenje rezultatov igre, glede na stopnjo, v kateri se igra nahaja.

Poglavje 6

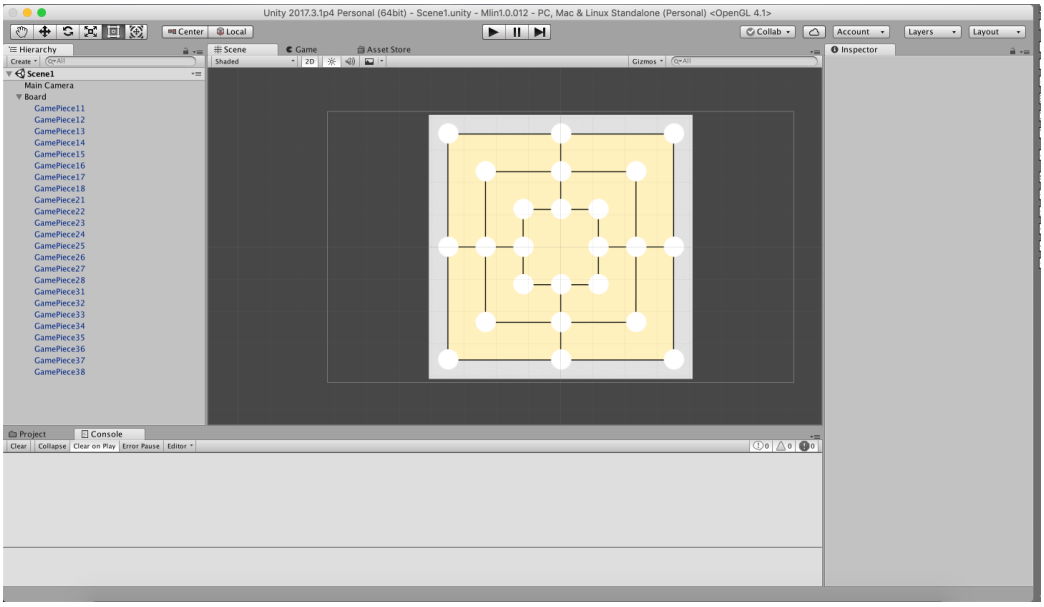
Tehnologije in orodja

6.1 Unity

Unity3D je igralni pogon, s podporo za več različnih operacijskih sistemov, ki ga je razvilo podjetje Unity Technologies. V začetku je bil ta pogon namenjen predvsem 3D igram, vendar so v zadnjih letih naredili veliko napredka tudi v podpori 2D, saj se predvsem trg mobilnih iger zelo širi, tu pa so bolj uporabne 2D igre. Zaradi vsesplošne uporabnosti je uporaba pogona zelo razširjena in posledično ni težko najti pomoči oziroma odgovorov na vprašanja v povezavi s kako kaj narediti.

6.1.1 Programsko okolje Unity

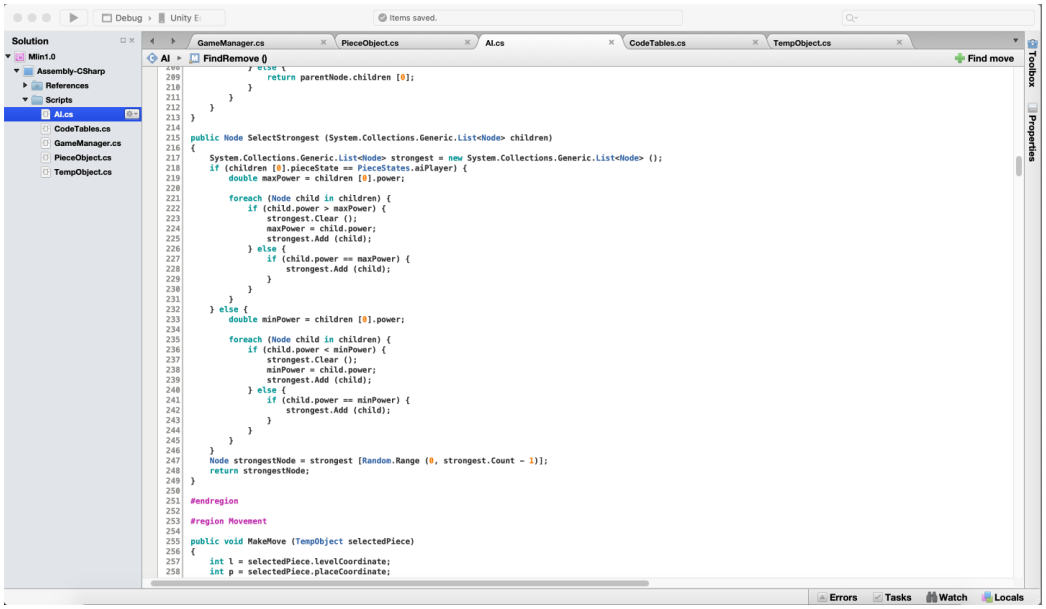
Za lažje razvijanje scene in objektov so pri Unity razvili svoj programski vmesnik (Slika 6.1), preko katerega z lahkoto sestavimo zaslone igre in jih povežemo s kodo. Tako si lažje predstavljamo, kako bodo stvari izgledale, brez da bi morali zagnati igro. V njem si izdelamo vse objekte od 2D do 3D in jih uredimo v začetno stanje na zaslonu, jim nastavimo vse potrebne lastnosti, ter dodamo potebne komponente, kot so zvok, točke za zaznavnje trkov in ostale. S tem si olajšamo razvoj, saj za velik del delovanja naše igre poskrbi že sam pogon.



Slika 6.1: Uporabniški vmesnik Unity

6.1.2 MonoDevelop

Poleg samega izgleda zaslona pa je potrebno poskrbeti tudi za pravilno delovanje in premikanje stvari, ki se na njej nahajajo. V ta namen imamo na voljo Microsoft Visual Studio ali MonoDevelop (Slika 6.2). MonoDevelop je program za pisanje kode, namenjene izvajanju v okolju Unity. Kljub temu, da je program namenjen splošnemu programiranju in se v njem lahko razvija v večih jezikih, se zaradi odločitve razvijalcev pogona Unity za razvoj iger z njihovim pogonom uporablja zgolj še C#. Ko ustvarjamo skripte za posamezne objekte, jih povežemo z objektom na zaslonu in tako poskrbimo, da se vse dogajanje z objektom izvaja tudi na skripti, če definiramo vse potrebne metode, ter po potrebi skozi igro spreminjamo lastnosti objekta oziroma njegovo delovanje.



Slika 6.2: Uporabniški vmesnik MonoDevelop

6.2 Programski jezik C#

C ali „C sharp“ je objektno orientiran jezik, ki ga je razvil Microsoft, za potrebe razvoja v okolju .NET in kot konkurenca jeziku Java. Cilj razvoja je bil sodoben objekto orientiran jezik, na osnovi jezika C. Danes je uporaba razširjena, od razvoja programov, do internetnih skript, z razvojem raznih programov, kot je Unity ali Xamarin, pa lahko z njim razvijamo tudi aplikacije za mobilne naprave in igre.

Kot ime so na začetku uporabljali Cool (C-Like Object Oriented Language), vendar so ga spremenili v C#, da bi se izognili potencialnim problemom z imenom znamke [8]. Za ime so raje izbrali C#, kjer znak # predstavlja glasbeno lastnost zvišanja tona (višaj), podobno kot C++, kjer ++ predstavlja povečanje za 1. Ker # lahko predstavlja tudi 4 znake + v mreži, so s tem hoteli nakazati, da je jezik nadgradnja jezika C++ [9].

Ker jezik še danes aktivno razvijajo in izboljšujejo, s tem poskrbijo za novosti ter za to, da je jezik v koraku s časom in poskrbijo za vse večjo

uporabnost. Zaradi svoje sintakse in oblike, je koda napisana v C#, lahko berljiva in jezik je lahek za učenje, hkrati pa nam ponuja tudi dovolj zahtevnejših funkcij, s katerimi lahko izdelamo kompleksne in napredne izdelke.

Poglavje 7

Sklepne ugotovitve

Po celotnem pregledu in opazovanju lahko vidimo, da je MCTS pri igrah, kljub svoji preprostosti, treba izvesti previdno in premišljeno, saj lahko napačna ali preveč preprosta uporaba hitro pripelje do nepravilnega delovanja. Kljub svoji moči je to algoritem, za katerega, če hočemo doseči pravilno delovanje, moramo poskrbeti, da nima na voljo preveč vozlišč. Na žalost se v primeru, ko ima na voljo veliko število vozlišč, hitro zgodi, da se osredotoči na napačno vozlišče in nam tako ponudi napačne rezultate. Seveda je vse odvisno, kakšne rezultate pričakujemo oziroma iščemo. Ko ga uporabimo za iskanje rešitev manjših problemov, pa lahko uporabimo zgolj MCTS, saj je dovolj uspešen, da sam poišče pravo rešitev.

Kadar ga uporabljamo, moramo vedno paziti, da pravilno ocenimo, kaj želimo iskati, ter velikost raziskovanja, s pravilnim številom simulacij in pravo velikost konstante za preiskovanje manj raziskanih vozlišč. Preveliko število simulacij tudi lahko povzroči, da se samo simuliranje dogaja predolgo oziroma je pomnilniško preveč zahtevno. Torej za pričakovano delovanje se moramo pravilno odločiti za pravilno velikost večih parametrov, ki imajo vsak posamezno majhen vpliv, skupno pa lahko povzročijo velike razlike.

Za pravilno delovanje pri večjih igrah moramo torej poskrbeti, da se MCTS odloča samo med potezami, ki so v dani situaciji ustreznejše. Saj se v nasprotnem primeru lahko zgodi, da se osredotoči na napačne poteze in

tako ne deluje pravilno. Za boljše rezultate lahko formulo za računanje moči tudi priredimo, vendar moramo paziti, da se s tem ne osredotočimo preveč na vozlišča, ki so ustreznejša za razvijalca oziroma vozlišča, ki po naši oceni predstavljajo pravilno delovanje.

MCTS je torej algoritem, ki je samostojno dovolj močan zgolj za manjše probleme. Ko pa ga uporabimo pri reševanju večjih problemov, moramo uporabiti hevristične metode za izboljšanje njegovega delovanja oziroma je sam algoritem hevristična metoda za izboljšanje delovanja algoritmov, ki iščejo rešitve. Seveda moram pri tem opozoriti, da sem za svojo igro uporabil zgolj osnovni MCTS, saj obstaja veliko izboljšav oziroma predvsem veliko sprememb na formuli za računanje moči, vendar je potrebno paziti, saj je to velikokrat dopolnitev formule, ki izboljša rezultate na problemu, kjer je uporabljena, drugje pa bi lahko bila popolnoma zgrešena.

Literatura

[1] Genetski algoritem. Dosegljivo:
http://wiki.fmf.uni-lj.si/wiki/Genetski_algoritem. [Dostopano 6.1.2019].

[2] Zgodovina formule. Dosegljivo:
https://en.wikipedia.org/wiki/Monte_Carlo_tree_search#Exploration_and_exploitation.
[Dostopano 14.1.2019].

[3] Požrešen algoritem. Dosegljivo:
https://en.wikipedia.org/wiki/Greedy_algorithm#/media/File:Greedy_algorithm_36_cen
[Dostopano 10.1.2019].

[4] Three men's morris. Dosegljivo:
https://en.wikipedia.org/wiki/Three_men%27s_morris#Rules. [Dostopano 31.12.2018].

[5] Six men's morris. Dosegljivo:
https://en.wikipedia.org/wiki/Nine_men%27s_morris#Six_men's_morris.
[Dostopano 31.12.2018].

[6] Morabaraba. Dosegljivo:
https://en.wikipedia.org/wiki/Morabaraba. [Dostopano 31.12.2018].

[7] Lasker morris. Dosegljivo:
https://en.wikipedia.org/wiki/Nine_men%27s_morris#Lasker_morris.
[Dostopano 31.12.2018].

[8] Cool. Dosegljivo: https://www.techopedia.com/definition/26272/c-sharp. [Dostopano 7.1.2019].

[9] C#. Dosegljivo: https://en.wikipedia.org/wiki/C_Sharp_(programming_language)#Name. [Dostopano 7.1.2019].