Универзитет у Београду

Електротехнички факултет



ДИПЛОМСКИ РАД

**ИНТЕЛИГЕНТНА ВИЗУЕЛНА СИМУЛАЦИЈА ВИРТУЕЛНОГ ИГРАЧА ШАХА**

Ментор

проф. др Бошко Николић

Студент

Предраг Митровић 2015/0608

Београд, септембар 2019.

Садржај

1. Увод

Рачунари су способни да играју шах. Први покушај да се аутоматизује игра шаха се догодио 1914. године помоћу машине коју је конструисао Торес Кеведо која је била способна да изведе мат помоћу две фигуре, краља и топа, против противничког краља.

У другој половини 20. века, људи су почели да се баве програмирањем рачунарских играча за шах са циљем да направе играча који би био непобедив. Прва машина која је успела да победи у шаховском мечу против човека који је светски шампион био је рачунар *Deep Blue* развијен од стране америчке компаније *IBM*. Био је састављен од 50 рачунара и направљен је специјално за ову намену. Победио је Гарија Каспарова, велемајстора шаха и једног од најбољих играча свих времена. Било је потребно конструисати машину која евалуира 200 милиона потеза у секунди да би се то догодило.

Током времена, заједница људи која се интересује за развој сличних решења се проширила, хардвер је напредовао, али без обзира на то и даље не постоји непобедива машина из разлога што је стабло игре исувише комплексно и садржи приближно 10120 чворова.

Један од најпопуларнијих статистичких метода за мерење вештине ботова за шах се ослања на турнире где бот игра против различитих ботова огроман број мечева са истим временским ограничењима. Закључак о перформансама се изводи на основу броја победа, нерешених исхода и пораза. Потенцијални проблем је што у овом случају коначна оцена зависи од снаге противника против којих су играни мечеви. Оцена се изражава у поенима и након тога се формира коначан пласман.

Софтверски пакет за персонални рачунар који је настао као резултат овог дипломског рада се састоји из две целине. Прву целину представља виртуелни играч који анализира стања шаховске табле, генерише могуће потезе и врши одабир потеза који је евалуирао као најбољи. Поред тога, реализован је и графички кориснички интерфејс који омогућава кориснику да игра против рачунара, да посматра игру између два бота и да игра против другог човека. Програм на тај начин даје кориснику прилику да напредује као играч.

Решење је имплементирано у програмском језику C# уз коришћење мултиплатформске машине за игру под називом *Unity.*

Неки од разлога који чине програмирање виртуелних играча интересантним су коришћење хеш функција, алгоритама сортирања, рекурзивних функција, генерисање случајних бројева и разних других метода при имплементацији решења. Током рада се продубљује знање програмирања у изабраном програмском језику јер се тежи писању ефикасног и „чистог” кода.

У поглављу 2 биће размотрени неки од приступа које су програмери у прошлости користили при решавању проблема који се обрађује. Поглавље 3 ће бити корисничко упутство за коришћење апликације која је резултат дипломског рада. У поглављу 4 ће бити описани детаљи имплементације решења. У поглављу 5 ће бити размотрени неки од недостатака решења и биће приказани предлози за надоградњу система.

1. Преглед постојећих решења

Шах је игра „савршене информације” из разлога што су оба играча свесни целокупног стања игре у сваком тренутку. Посматрајући таблу, они могу закључити које фигуре су у игри и где се налазе. Игре као што су дама, тавла и го спадају у исту категорију, док рецимо покер не спада из ралога што није могуће видети које карте у руци држи противник.

Већина техника које су коришћене за решавање проблема се могу применити на различите игре „савршене информације”, иако детаљи могу варирати од игре до игре. Исти алгоритам претраживања се може применити на било коју од њих, док генерација потеза и евалуација стања зависи од правила игре.

Да би бот успешно играо шах, неопходно је реализовати различите софтверске компоненте од којих су неке:

* Начин репрезентације шаховске табле у меморији како би у сваком тренутку знао које је стање у ком се налази.
* Правила генерације легалних потеза како би поштовао правила игре и уверио се да противник не покушава да прекрши неко од правила.
* Интелигентан начин избора наредног потеза.
* Кориснички интерфејс који омогућава комуникацију са противником.

Програмери који су зачетници развоја виртуелних играча за шах су имали велико ограничење у пројектовању решења зато што је у то време меморија била изузетно лимитирана. Неки од програма су користили само 8 *kB* меморије*.* Због тога су најједноставније репрезентације табле биле најефикасније. Типична табла је имплементирана као низ од 64 елемента где је сваки елемент представљао једно поље на табли. Сваки елемент је био величине 1 *B* и празном пољу је додељивана вредност 0, белом пешаку вредност 1 итд. У данашње време већина машина су 64-битне па су постале актуелне репрезентације табле под називом *Bitboards* где једна 64-битна реч садржи информације о једном аспекту стања игре као што је рецимо распоред пешака на табли где сваки од 64 бита показује да ли је поље заузето. Овакве репрезентације су ефикасне зато што је потребно често вршити логичке операције трају само један процесорски циклус.

Правила игре одређују које потезе може да повуче играч који је на реду. У неким играма је једноставно одредити легалне потезе. Рецимо, у игри икс-окс празно поље означава легалан потез. У шаху, правила су компликованија. Свака фигура има своја правила кретања, није дозвољено да након потеза једног од играча његов краљ остане у шаху, начин промоције фигура, специфични потези као што су рокада или *En Passant*.

Потребно је омогућити рачунару да на што бољи начин врши одабир потеза. Најбољи начин избора између два потеза јесте да се сагледају њихове последице (да се претраже потези који следе иза њих и да се резултати упореде). Како би се што мање смањила могућност грешке, бот претпоставља да ће противник изабрати најбољи потез по себе, односно најлошији по бота. Ово је основни принцип *Minimax* алгоритма претраге који је основа великог броја постојећих решења.

Нажалост, *Minimax* алгоритам има поприлично велику сложеност О(bn) где је b фактор гранања, односно број легалих потеза у просеку, а n је дубина претраге, односно број потеза који се посматрају унапред. Неки од најуспешнијих алгоритама који представљају побољшања *Minimax*-а су *AlphaBeta*, *NegaScout* и *MTD(f)* који покушавају да минимизују број чворова које је потребно обићи у стаблу игре.

Још један велики проблем при претрази јесте *„horizon effect”* који је првобитно описао Ханс Берлинер. Претпоставимо да програм претражи све чворове до дубине 4 и закључи да на дубини 4 постоји потез где ће изгубити краљицу. Програм у том случају може одлучити да брани краљицу другим фигурама и одложи губитак краљице још неколико потеза да би је на крају свакако изгубио. Са тачке гледишта бота краљица је спашена јер није довољно дубоко претражио потезе да би видео да ће она свакако бити изгубљена. На крају изгубио је и краљицу и остале фигуре којом ју је бранио и сада се налази у још лошијој позицији. Постоје разне технике које су развијене како би се избегао овај ефекат од којих је најпознатија *Quiescence* претрага.

Програм мора бити у стању да евалуира стања игре, односно да буде у стању да закључи да ли је позиција повољна или неповољна по њега. Осмишљање добре функције евалуације може бити изузетно тежак посао. Грешке при евалуацији могу довести до губитка партије из разлога што врхунски играчи могу да искористе минималну предност као што је рецимо један пешак више у односу на противника.

1. Корисничко упутство