# Дипломна работа

на тема

**Аркаден стил Шах игра с изкуствен интелект(AI)**

Дипломант: **Мартин Неделинов Петров**

ученик от 12В клас в

ЧПГДН "СофтУни Светлина"

Научен ръководител: **Георги Кирилов Величков**

Рецензент: **XXX XXX XXX**

Дата: **12 май 2023**

Сесия: май-юни 2023 г.

# Съдържание

[Дипломна работа](#_30j0zll) **1**

[Съдържание](#_1fob9te) **2**

[Увод](#_3znysh7) **5**

[Проблем](#_2et92p0) 5

[Цели на дипломния проект](#_tyjcwt) 5

[Глава 1. Главни принципи на (AI) и езици за разработка](#_1t3h5sf) **6**

[Главни методологии за разработка на изкуствен интелект(AI)](#_p1568eh2a7gs) 6

[1. Системи, базирани на правила (Rule-Based systems)](#_d015qfcygq4l) 6

[2. Експертни системи (Expert Systems)](#_ajg7ysc2or1l) 7

[3. Размита логика (Fuzzy Logic)](#_p6cjjq5k1v2l) 7

[4. Невронни мрежи (Neural Networks)](#_fsifwyavlie3) 7

[5. Еволюционни алгоритми (Evolutionary Algorithms)](#_9w4cu2t70xpv) 8

[6. Генетични алгоритми (Genetic Algorithms)](#_azg7a5dzhdph) 8

[7. Интелигентност на рояка (Swarm Intelligence)](#_jwkhb4o74f09) 8

[8. Обучение с подсилване (Reinforcement Learning)](#_9ewnhkj25r2y) 9

[9. Обработка на естествен език (NLP - Natural Language Processing)](#_ldnd73vzkv5j) 9

[10. Дълбоко обучение (Deep Learning)](#_k6qt2ind74rj) 9

[Популярни езици за разработка на изкуствен интелект(AI)](#_wm9vhrvct3xy) 10

[1. Python:](#_ooeq4u2y3vuz) 10

[2. R:](#_r35yso4ttmq9) 11

[3. Java:](#_2dbdypv696mq) 11

[4. C++:](#_6m2lbotema15) 11

[5. Julia:](#_k4y5yswf1d9a) 11

[6. Lisp:](#_6w3gx3vyzl6b) 12

[7. Prolog:](#_ifbdq5ejoxzb) 12

[8. MATLAB:](#_24lk2vibui5h) 12

[9. JavaScript:](#_nqs3bh86km58) 12

[10. Swift:](#_ifxk1syjsi95) 13

[фиг.1](#_a4qb0y28kpzw) 13

[Глава 2. Аркаден стил Шах с изкуствен интелект](#_4d34og8) **14**

[1. Защо Python?](#_4dpxscsj9zum) 14

[Python е лесен за научаване и използване:](#_cphk35o4w8pc) 14

[Python има силна подкрепа от общността:](#_b4ntcs8fvov5) 15

[Python е съвместим с различни платформи:](#_hssbpt320jlw) 15

[Python осигурява ефективно управление на паметта:](#_6t718wlv7f8f) 15

[Python поддържа обектно-ориентирано програмиране:](#_jjsskahit79f) 15

[Pygame е отличен избор за създаване на игри:](#_v2ggtfpj2yux) 16

[Python предоставя множество парадигми за програмиране:](#_bwi3vismmy6v) 16

[2. Къде живее кода ни?](#_cjjtwduno48q) 16

[Първо,](#_8vvrlv50ydfi) 16

[Второ,](#_yw7ltnbs8f8w) 16

[Трето,](#_nmmg9xkj9c9o) 17

[Четвърто,](#_94j760brpyoy) 17

[3. Компоненти и тяхното съдаржание](#_nny59b3oij50) 17

[ChessEngine.py:](#_9vlt8sybo1sb) 17

[Main.py](#_ma3l8f7nchaq) 18

[PlayerVsComputer.py](#_2x397nnt8g14) 19

[PlayerVsPlayer.py](#_tcfhdv7dafon) 19

[SmartMoveFinder.py](#_fce3vxgru2fm) 20

[Глава 3. Главен алгоритъм на програмата и структура на играта](#_2s8eyo1) **20**

[Архитектура на системата](#_17dp8vu) 20

[Имплементация на системата](#_3rdcrjn) 20

[1. piece\_score dictionary:](#_hiuzjlm5l62r) 23

[2. Константи CHECKMATE и STALEMATE:](#_f558bvqiv7o0) 23

[3. Константа DEPTH:](#_4wv0xna8z2pj) 23

[4. Функция find\_best\_move\_minmax():](#_5bwn1mxa9ped) 23

[5. Функция find\_move\_minmax():](#_o4vl40v9k64s) 24

[6. Функция score\_board():](#_q9kt4q7088z1) 24

[7. Функция score\_material(board):](#_yzogrg89dlsl) 25

[(Castling) и (En Passant ).](#_bhnqdxerloiq) 25

[Проверка за валиден ход](#_7r9qqta9p75y) 26

[Ако има само един шах,](#_dllwtz8h15x) 28

[Ако има две проверки,](#_8oxoak870y6m) 28

[Ако играчът не е в шах,](#_uhc0op8ledeu) 29

[highlight\_squares](#_hw0eqjog06eg) 29

[Внедряване (deployment)](#_j20zvrmjvu8s) 30

[Глава 4. Ръководство за стартиране на програмата](#_35nkun2) **31**

[**Заключение**](#_qgu7xxax7kqm) **31**

[Информационни източници](#_2jxsxqh) **32**

[Рецензия на дипломен проект](#_3j2qqm3) **33**

# Увод

Добре дошли в света на шаха и изкуствения интелект в Python!

Този проект ще ви преведе през разработването и внедряването на игра на шах с AI в Python. Шахът е стратегическа настолна игра за двама играчи, която изисква критично мислене, планиране и вземане на решения. В този проект ще изградим игра на шах, задвижвана от AI, която може да се играе от един играч срещу компютъра. Нашата шахматна игра ще бъде проектирана с помощта на език за програмиране Python, който е популярен език за AI и приложения за машинно обучение. Ще използваме библиотеката Pygame, която ни предоставя необходимите инструменти за създаване на графичен потребителски интерфейс за нашата игра на шах. AI компонентът на нашата игра на шах ще използва алгоритъма minimax с алфа-бета подрязване, за да взема интелигентни решения. Този алгоритъм позволява на компютъра да търси във всички възможни ходове и да оценява най-добрия ход въз основа на текущото състояние на играта. В допълнение към геймплея, ние също ще внедрим редица функции, които подобряват потребителското изживяване, като звукови ефекти, отмяна и повторение на ходове и подчертаване на валидни ходове за избраната фигура.

## Проблем

Във все така бързо напредващия свят на технологията и програмирането е лесно за човек да се изгуби. AI(Artificial Intelligence) е една от по новите и непознати сфери в света на технологията и програмирането, но също така е и една от най-важните. Главния проблем е, че повечето хора си нямат изобщо на представа какво означава една машина да мисли сама и да взема оптимални решения за дадена ситуация без помощта на човек . С този проект се старая да запозная читателя със най-основните части на изкуствения интелект и

## Цели на дипломния проект

Целта на настоящата дипломна работа е да покаже как една машина мисли сама и как стига до дадени решения. Да демонстрира как една машина може да мисли на различни нива .Да покаже, че изкуствения интелект (AI) е най-бързия път към един по-добре оптимизиран и развит свят. Да запознае обикновения човек със света на изкуствения интелект.

Във връзка с поставената цел за разработван са поставени следните задачи:

* Извършване на подробно проучване и подготовка на обзор на проблемната област
* Проучване на инструментите и технологиите за работа с изкуствен интелект(AI)
* Проектиране на програма която е лесна за разбиране от хора на всякаква възраст и показва на обикновения човек как един компютър мисли сам
* Сравняване на различните среди за разработка на изкуствен интелект(AI) и представяне на миносвете и плюсовете на всяка една от популярните методологии за развитие на машинния интелект.

# Глава 1. Главни принципи на (AI) и езици за разработка

Изкуственият интелект (AI) е усъвършенствана технология, която е разработена, за да имитира човешкия интелект, вземането на решения и когнитивните способности. Това е бързо развиваща се област на компютърните науки, която включва създаването на интелигентни машини, които могат да изпълняват задачи, които обикновено изискват човешка намеса. AI системите могат да бъдат обучени да се учат и да подобряват ефективността си с течение на времето, което ги прави все по-ефективни при изпълнението на сложни задачи.

AI може да бъде разработен по няколко различни начина, всеки със своите предимства и недостатъци. В тази статия ще разгледаме десет различни подхода към разработването на (AI), вариращи от системи, базирани на правила, до задълбочено обучение на алгоритми(Deep Learning Algorithms) и други.

## Главни методологии за разработка на изкуствен интелект(AI)

### Системи, базирани на правила (Rule-Based systems)

Системите, базирани на правила, са една от най-ранните форми на развитие на изкуствен интелект(AI). Те се състоят от набор от оператори if-then, които позволяват на (AI) система да взема решения въз основа на набор от предварително определени правила. Например, система, базирана на правила, може да се използва за откриване на измами чрез анализиране на набор от финансови транзакции и маркиране на всички, които отговарят на определени критерии.

Предимството на базираните на правила системи е, че те са относително лесни за разработване и могат лесно да бъдат разбрани от хората. Те обаче могат да бъдат ограничени в способността си да се справят със сложни ситуации и може да изискват чести актуализации, когато се добавят нови правила.

### Експертни системи (Expert Systems)

Експертните системи са вид система, базирана на правила, която е предназначена да имитира способностите за вземане на решения на човешки експерти в определена област. Тези системи се обучават с помощта на комбинация от знания и опит от човешки експерти и могат да се използват за предоставяне на съвети или вземане на решения в широк диапазон от области.

Предимството на експертните системи е, че те могат да предоставят точни и последователни съвети, дори в сложни ситуации. Те обаче могат да бъдат скъпи за разработване и може да изискват текуща поддръжка, за да бъдат в крак с промените в домейна.

### Размита логика (Fuzzy Logic)

Размитата логика е математически подход към разработването на AI, който позволява представянето на несигурни или двусмислени данни. За разлика от традиционната булева логика, която е ограничена до двоични верни/неверни твърдения, размитата логика може да представлява степени на истина.

Размитата логика може да се използва в широк спектър от приложения, от контролиране на температурата в сграда до прогнозиране на цените на акциите. Предимството на размитата логика е, че може да обработва сложни и несигурни данни, което я прави идеална за приложения в реалния свят.

### Невронни мрежи (Neural Networks)

Невронните мрежи са вид разработка на изкуствен интелект(AI), която се моделира според структурата и функцията на човешкия мозък. Тези системи се състоят от взаимосвързани възли или неврони, които могат да обработват информация и да взимат решения въз основа на тази информация.

Невронните мрежи могат да се използват в широк спектър от приложения, от разпознаване на изображения до обработка на естествен език. Предимството на невронните мрежи е, че те могат да се учат и подобряват с течение на времето, което ги прави все по-ефективни при справянето със сложни задачи.

### Еволюционни алгоритми (Evolutionary Algorithms)

Еволюционните алгоритми са вид структура за изкуствен интелект(AI), което се моделира след процеса на естествен подбор. Тези системи използват процес на проба и грешка, за да генерират решения на сложни проблеми.

Еволюционните алгоритми могат да се използват в широк спектър от приложения, от проектиране на нови продукти от нулата до оптимизиране на веригите за доставки. Предимството на еволюционните алгоритми е, че те могат да генерират силно оптимизирани решения, дори и в сложни ситуации.

### Генетични алгоритми (Genetic Algorithms)

Генетичните алгоритми са вид еволюционен алгоритъм, който е моделиран след процеса на генетично наследяване. Тези системи използват процес на мутация и селекция, за да генерират решения на сложни проблеми.

Генетичните алгоритми могат да се използват в широк спектър от приложения, от оптимизиране на продуктов дизайн до създаване на сложни математически модели. Предимството на генетичните алгоритми е, че те могат да генерират силно оптимизирани решения, дори и в ситуации, в които традиционните методи може да се провалят.

### Интелигентност на рояка (Swarm Intelligence)

Интелигентността на рояка(Swarm Intelligence) е вид развитие на изкуствен интелект(AI), което е моделирано след поведението на социални насекоми, като мравки и пчели. Тези системи използват процес на самоорганизация за решаване на сложни проблеми.

Интелигентността на рояка(Swarm Intelligence) може да се използва в широк спектър от приложения, от оптимизиране на трафика на данните до оптимизиране на веригите за доставки. Предимството на (Swarm Intelligence) е, че може да генерира силно оптимизирани решения, дори в динамична и променяща се среда.

### Обучение с подсилване (Reinforcement Learning)

Обучението с подсилване е вид развитие на изкуствен интелект(AI), което се основава на концепцията за проба и грешка. Тези системи използват процес на учене от обратна връзка, за да подобрят ефективността си с течение на времето.

Обучението с подсилване може да се използва в широк спектър от приложения, от роботика до игра на игри. Предимството на обучението с подсилване е, че може да генерира силно оптимизирани решения, дори в сложни и променящи се среди.

### Обработка на естествен език (NLP - Natural Language Processing)

Обработката на естествен език (NLP) е вид разработка на (AI), която се фокусира върху взаимодействието между компютри и хора на естествен език. Тези системи използват комбинация от машинно обучение и лингвистика, за да разберат и генерират човешки език.

(NLP) може да се използва в широк спектър от приложения, от чатботове до автоматизиран превод. Предимството на (NLP) е, че може да подобри ефективността и точността на комуникацията между хора и машини.

### Дълбоко обучение (Deep Learning)

Дълбокото обучение е вид разработка на изкуствен интелект(AI), която се основава на невронни мрежи с множество слоеве. Тези системи могат да анализират и обработват огромни количества данни, което ги прави идеални за сложни задачи като разпознаване на изображения и разпознаване на реч.

Предимството на дълбокото обучение (Deep Learning) е, че може да генерира много точни резултати, дори в ситуации, в които традиционните методи за машинно обучение могат да се провалят. Системите за дълбоко обучение обаче изискват големи количества данни и изчислителна мощност, което ги прави скъпи и отнема много време за разработване.

Изкуственият интелект(AI) е бързо развиваща се област на компютърните науки, която има потенциала да революционизира начина, по който живеем и работим. Има много различни начини за разработване на AI системи, всеки със своите предимства и недостатъци. Системите, базирани на правила, и експертните системи са сравнително лесни за разработване, но могат да бъдат ограничени в способността си да се справят със сложни ситуации. Размитата логика, невронните мрежи и еволюционните алгоритми са по-усъвършенствани методи, които могат да обработват сложни и несигурни данни. Интелигентността на рояка, обучението с подсилване, обработката на естествения език и дълбокото обучение са още по-усъвършенствани методи, които могат да генерират силно оптимизирани решения за сложни задачи.

Знаейки основните принципи и методологии на изкуствения интелект(AI) следва да се запознаем как да ги приложим в практика и кой програмен език е най-подходящ за дадената ни задача. В следващите страници ще се запознаем и сравним най-популярните програмни езици за разработка на изкуствен интелект(AI).

## Популярни езици за разработка на изкуствен интелект(AI)

### Python:

Python е най-популярният програмен език за разработка на AI, тъй като осигурява прост и лесен за разбиране синтаксис, четливост и абстракция на високо ниво. Той има обширна библиотека за изкуствен интелект(AI), като Tensorflow, Keras, Scikit-Learn, PyTorch и много други. Python се използва широко за разработване на приложения за обработка на естествен език, машинно обучение, задълбочено обучение и приложения за компютърно зрение(computer vision applications).

### R:

R е език за програмиране, който се използва широко за статистически изчисления и анализ на данни. Той предоставя широк набор от пакети за машинно обучение, анализ на данни и визуализация на данни. R е много подходящ за разработване на алгоритми и модели за изкуствен интелект(AI) и се използва широко в академични изследвания и проекти за наука за данни.

### Java:

Java е популярен език за програмиране за разработване на широкомащабни (AI) системи, като роботика, интелигентни агенти и експертни системи. Той предоставя широк набор от инструменти и рамки, като WEKA, Java-ML, Deeplearning4j и други, които са много подходящи за разработване на (AI) приложения.

### C++:

C++ е високопроизводителен език за програмиране, който се използва широко за разработване на (AI) приложения, особено в компютърното зрение и обработката на изображения. Той осигурява достъп до управление на паметта на ниско ниво и поддържа многопоточност, което го прави много подходящ за разработване на (AI) приложения в реално време.

### Julia:

Julia е сравнително нов език за програмиране, който е предназначен за числени изчисления и научни изчисления. Той осигурява високо ниво на абстракция и производителност, което го прави много подходящ за разработване на (AI) приложения, особено за числен анализ, оптимизация и машинно обучение.

### Lisp:

Lisp е функционален език за програмиране, който се използва широко за разработване на експертни системи, обработка на естествен език и приложения за машинно обучение. Той предоставя широк набор от функции, като например динамично въвеждане, събиране на боклук(garbage collection) и макроси, които са много подходящи за разработване на (AI) приложения.

### Prolog:

Prolog е декларативен език за програмиране, който се използва широко за разработване на експертни системи и системи базирани на правила. Той осигурява мощен двигател за изводи и поддържа обратно проследяване, което го прави много подходящ за разработване на (AI) приложения, които изискват разсъждения и вземане на решения.

### MATLAB:

MATLAB е език за програмиране, който се използва широко за числени изчисления и анализ на данни. Той предоставя широк набор от инструменти за машинно обучение, задълбочено обучение(Deep Learning) и компютърно зрение(computer vision), което го прави много подходящ за разработване на (AI) приложения.

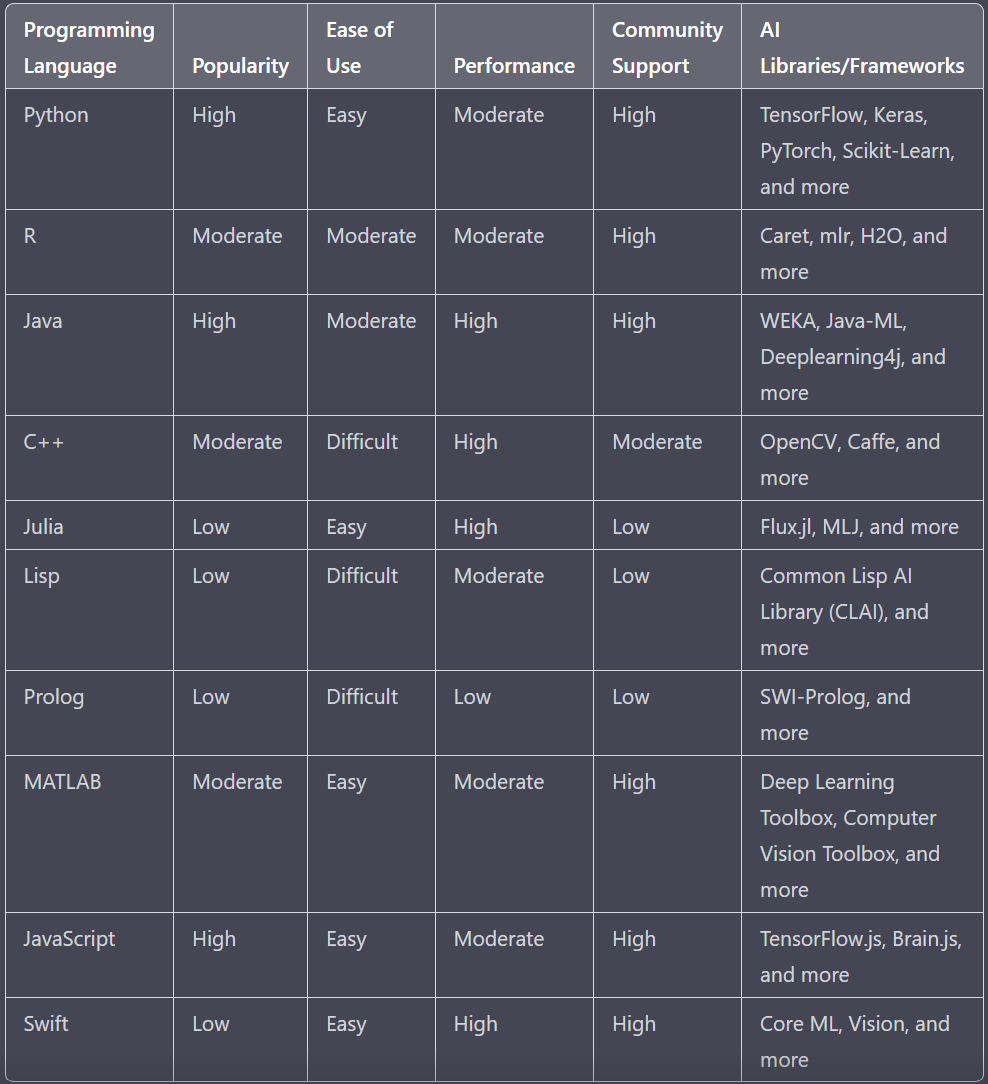
### JavaScript:

JavaScript е популярен език за програмиране, който се използва широко за разработване на уеб-базирани (AI) приложения, като чатботове и системи за препоръки. Той предоставя широк набор от библиотеки и рамки, като TensorFlow.js и Brain.js, които са много подходящи за разработване на (AI) приложения.

### Swift:

Swift е език за програмиране, който е предназначен за разработване на приложения за iOS и macOS. Той предоставя широк набор от библиотеки и рамки, като Core ML, които са много подходящи за разработване на (AI) приложения за платформи на Apple.

Лекотата на използване е важен фактор за начинаещи в разработването на (AI). Python, JavaScript и MATLAB са известни със своя лесен за научаване синтаксис, и са перфектен избор за предприемането на първи стъпки в сферата на изкуствения интелект.

Тук е конструирана таблица която просто и лесно представя написаната до сега информация.(*фиг 1*) 

## *фиг.1*

В заключение, изборът на език за програмиране за разработка на (AI) зависи от различни фактори, включително вида на приложението на (AI), средата за разработка, нивото на експертиза и наличните ресурси. Python е най-популярният и гъвкав език за разработка на (AI), следван от JavaScript, Java, R и MATLAB. Важно е да изберете език, който осигурява необходимата производителност, лекота на използване, поддръжка от общността и достъп до (AI) библиотеки и рамки за разработване на ефективни и мащабируеми (AI) приложения.

# Глава 2. Аркаден стил Шах с изкуствен интелект

Поставената цел е да създадем програма която лесно, бързо и интересно представя какво е изкуствен интелект и как работи. Главната идея е да бъде достъпно за всички и да е лесно разбираемо за хора на всякаква възраст. За разработка на програмата ще се използва Python а самата програма ще е под формата на шах в който играча може да играе срещу друг човек на същото устройство или да играе срещу (AI) на едно от 3 нива на трудност. Програма ще адаптира вид на аркадна игра от 80те/90те години, с лесно разбираемо начално меню. Идеята за стила произлиза от това че така програмата ще е ниско ресурсна и ще може да върви и на по-стари компютри и ще е достъпна за по-голяма аудитория. Избрана е играта на шах, защото е добре позната от повечето хора и е перфектна среда за тестване на логическите способности на изкуствения интелект, който ще бъде изграден от един нагледно прост но ефективен алгоритъм.

## Защо Python?

Python е популярен език за програмиране на високо ниво, който се превърна в предпочитан избор за разработване на системи с изкуствен интелект (AI) поради своята простота, четливост и широка подкрепа от общността. С възхода на (AI), Python се превърна в де факто езика за разработване на интелигентни системи, включително (AI) за игри, като (AI) за шах. В тази документация ще обсъдим защо Python е най-добрият програмен език за начинаещи които искат да създадат своят пръв изкуствен интелект(AI).

### Python е лесен за научаване и използване:

Python е език за програмиране на високо ниво, който е прост и лесен за научаване. Синтаксисът му е ясен и се чете като английски, което го прави лесен за четене, писане и поддържане на код. Минималистичният дизайн и четливостта на езика го правят лесен за разбиране и използване от начинаещи, което го прави популярен избор сред програмистите. За (AI) шах простотата на Python позволява на разработчиците да се съсредоточат повече върху алгоритмите и стратегиите, а не върху синтаксиса и техническите особености на езика.

### Python има силна подкрепа от общността:

Python има голяма и активна общност от разработчици и ентусиасти, които предоставят поддръжка, споделят идеи и си сътрудничат по проекти с отворен код. Тази общност е разработила множество алгоритми и библиотеки за шах (AI), които разработчиците могат да използват, за да изградят своите системи за шах (AI). Освен това огромната общност от разработчици на Python гарантира, че езикът остава подходящ и актуален, което го прави идеален избор за разработка на изкуствен интелект(AI).

### Python е съвместим с различни платформи:

Python е език, съвместим с различни платформи, което означава, че работи на различни операционни системи като Windows, Linux и macOS. Тази функция позволява на разработчиците да създават (AI) системи за шах, които могат да работят на различни устройства и платформи, което ги прави достъпни за по-широка аудитория.

### Python осигурява ефективно управление на паметта:

Python има ефективно управление на паметта, което е особено важно за разработването на (AI) в шах. Играта включва много изчисления и манипулации на състояния на играта, ходове и фигури, които могат бързо да изразходват много памет. Събирачът на отпадъци на Python и динамичното разпределение на паметта позволяват на разработчиците да управляват ефективно използването на паметта, намалявайки вероятността от изтичане на памет и други проблеми, свързани с паметта.

### Python поддържа обектно-ориентирано програмиране:

Python е обектно-ориентиран език за програмиране, който поддържа абстракция, капсулиране, наследяване и полиморфизъм. Тази функция улеснява моделирането на сложни системи като AI за шах, където различните компоненти на системата могат да бъдат капсулирани в класове и обекти. Този подход помага за управление на сложността, насърчава повторното използване на кода и намалява вероятността от грешки.

### Pygame е отличен избор за създаване на игри:

Pygame е отличен избор за създаване на игри в Python. Той предоставя прост API, съвместимост между платформи, поддръжка на звук и музика, графични функции, поддръжка на входно устройство, мрежови възможности и е с отворен код. Pygame улеснява начинаещите да разработват игри и е подходящ и за опитни разработчици, които искат да създават игри бързо и ефективно.

### Python предоставя множество парадигми за програмиране:

Python осигурява поддръжка за множество програмни парадигми, включително процедурно, функционално и обектно-ориентирано програмиране. Тази функция позволява на разработчиците да изберат най-добрата парадигма за своя проект, което улеснява писането на ефективен, четим и поддържаем код. Тази гъвкавост е особено полезна за разработването на (AI) за шах, където различни парадигми за програмиране могат да бъдат подходящи за различни части на системата.

## Къде живее кода ни?

За разработката на програмата ще използваме добре познатото и утвърдено IDE Visual Studio Code.

### Първо,

VS Code поддържа широк набор от езици за програмиране, включително популярни езици като Python, JavaScript и C++. Тази функция го прави идеален избор за разработчици, които работят с множество езици за програмиране, тъй като те могат да използват една IDE за всички свои проекти.

### Второ,

VS Code има богат набор от функции, които правят задачите за разработка по-ефективни. Тези функции включват интелигентно допълване на код, поддръжка за отстраняване на грешки, подчертаване на синтаксиса и интегриране на контрол на версиите. VS Code също има вградена поддръжка за Git, което позволява на разработчиците да управляват своя код по-лесно.

### Трето,

VS Code има стабилна система за разширение, която позволява на разработчиците да персонализират IDE, за да отговарят на техните специфични нужди. Има хиляди разширения, налични за VS Code, вариращи от прости помощни програми до сложни интеграции с други инструменти и платформи.

### Четвърто,

VS Code е лесен за използване, с прост и интуитивен интерфейс, който улеснява разработчиците да започнат. IDE има минималистичен дизайн, който не затрупва потребителите с ненужни функции или бъркотия. Освен това VS Code има богата документация и поддръжка от общността, предоставяйки на разработчиците ресурсите, от които се нуждаят, за да започнат бързо.

Като цяло използването на добро IDE може да ви помогне да бъдете по-продуктивни, да пишете код с по-високо качество и да си сътрудничите по-ефективно с другите.

## Компоненти и тяхното съдаржание

Обема на програмата е разпределен в 5 Python файла и две отделни папки, една за изображението на фигурите и една за звуковите ефекти

### ChessEngine.py:

Тук живее главната част от кода а именно създаването на полето, движението на различните фигури, правилата за специалните движения(Castling & En Passante), проверките за шах и мат, повдигането на пешките към царици и проверката дали даден ход е възможен. (Подробно разглеждане на тази част в Глава 3)

| self.board = [ ["bR", "bN", "bB", "bQ", "bK", "bB", "bN", "bR"],  ["bp", "bp", "bp", "bp", "bp", "bp", "bp", "bp"],  ["--", "--", "--", "--", "--", "--", "--", "--"],  ["--", "--", "--", "--", "--", "--", "--", "--"],  ["--", "--", "--", "--", "--", "--", "--", "--"],  ["--", "--", "--", "--", "--", "--", "--", "--"],  ["wp", "wp", "wp", "wp", "wp", "wp", "wp", "wp"],  ["wR", "wN", "wB", "wQ", "wK", "wB", "wN", "wR"]] |
| --- |

*диаграма на игровата дъска*

### Main.py

Тук е разположено началното меню на програма в което за вписани 4 опции, PVP(Player versus Player)и *снимка.1*

PVC(Player versus Computer) като тук има избор между 3 нива на трудност: Easy AI, Medium AI and Hard AI.

p.display.flip() актуализира целия екран с всички промени, които са направени след последното актуализиране на екрана. В контекста на този код той се използва за актуализиране на екрана на менюто с изобразените текстови обекти, така че да са видими за потребителя. Извиква се в края на цикъла while в main(), след като обработката на събитието и изчертаването на екрана са завършени, за да покаже крайната версия на екрана на потребителя.

### PlayerVsComputer.py

В този файл играта използва модула ChessEngine, който съдържа логиката на играта.

Функцията player\_vs\_computer\_launcher() е основната функция, която стартира играта. Първо инициализира Pygame, създава прозореца на играта, настройва часовника и инициализира състоянието на играта. Функцията load\_images() се извиква, за да зареди изображенията на шахматните фигури. Цикълът на играта се изпълнява, докато потребителят напусне играта. Вътре в цикъла функцията проверява дали е ред на човешкия играч и ако не е, използва ресурсите от файла SmartMoveFinder, за да намери най-добрия ход за компютъра. Ако е ред на човешкия играч, функцията изчаква щракване с мишката и запазва позицията на щракнатия квадрат. След като човешкият играч избере два квадрата, функцията проверява дали ходът е валиден и ако е така, съответно актуализира състоянието на играта. Функцията draw\_game\_state() се извиква, за да изобрази състоянието на играта на екрана. Функцията подчертава избраното поле и валидните ходове за текущата фигура. Цикълът на играта също така проверява за условия на мат и пат и приключва играта, ако някое от тези условия е изпълнено.

И накрая, функцията clock.tick() се извиква, за да ограничи честотата на кадрите, а p.display.flip() се извиква, за да актуализира дисплея.

### PlayerVsPlayer.py

В този файл играта се стартира чрез функцията player\_vs\_player\_launcher, която инициализира прозореца на играта и настройва цикъла на играта. Цикълът на играта обработва въведеното от потребителя и съответно актуализира състоянието на играта. Класът GameState в модула ChessEngine представя текущото състояние на играта, включително текущата конфигурация на дъската и кой играч трябва да се премести. Функцията load\_images зарежда изображенията за шахматните фигури, които ще бъдат показани на игралната дъска. Функцията draw\_game\_state е отговорна за изчертаването на текущото състояние на играта на екрана, включително дъската за игра, фигурите на дъската и всяко подчертаване на квадрати или валидни ходове. Функцията highlight\_squares се извиква от draw\_game\_state, за да подчертае текущо избрания квадрат и всички валидни ходове за избраната фигура. Функцията draw\_board се извиква от draw\_game\_state, за да нарисува шахматната дъска на екрана. Функцията draw\_pieces се извиква от draw\_game\_state за теглене на фигурите на дъската.

Като цяло кодът осигурява основна рамка за функционираща игра на шах за двама играчи с графичен интерфейс.

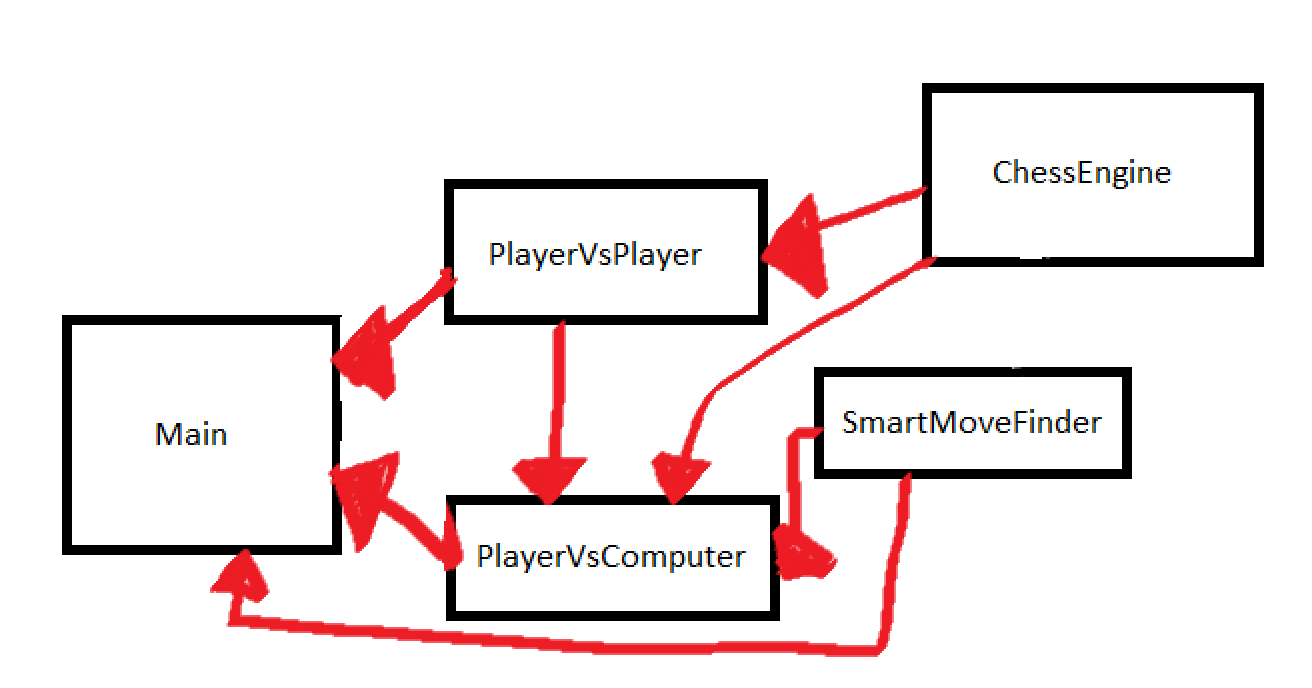
### SmartMoveFinder.py

Тук живее алгоритъма който компютъра използва за да играе срещу опонентите си .(Подробно разглеждане на тази част в Глава 3)

# Глава 3. Главен алгоритъм на програмата и структура на играта

## Архитектура на системата

Програмата се състои от 5 взаимо свързани Python файла, на диаграмата долу(диаграма.1) е показана как се осъществява връзката между тези файлове. Файловете работят заедно помежду си за осъществяването на програмата.

*диаграма.1*

## Имплементация на системата

На първо място в програмата е важно да се обясни файла SmartMoveFinder.py, това е алгоритъма който действа като главен мозък и базова структура на изкуствения интелект в програмата. Алгоритъма има 3 нива на (Depth) които съответно отговарят на 3те трудности в играта: Easy, Medium and Hard.

| import random  # your king as it would be a checkmate before that happened piece\_score = {"K": 0, "Q": 10, "R": 5, "B": 3, "N": 3, "p": 1}  CHECKMATE = 1000 STALEMATE = 0 DEPTH = 1   def find\_best\_move\_minmax(gs, valid\_moves):  global next\_move  next\_move = None  find\_move\_minmax(gs, valid\_moves, DEPTH, gs.white\_to\_move)  return next\_move   def find\_move\_minmax(gs, valid\_moves, depth, white\_to\_move):  global next\_move  random.shuffle(valid\_moves)  if depth == 0:  return score\_material(gs.board)   if white\_to\_move:  max\_score = -CHECKMATE  for move in valid\_moves:  gs.make\_move(move)  next\_moves = gs.get\_valid\_moves()  score = find\_move\_minmax(gs, next\_moves, depth - 1, False)  if score > max\_score:  max\_score = score  if depth == DEPTH:  next\_move = move  gs.undo\_move()  return max\_score   else:  min\_score = CHECKMATE  for move in valid\_moves:  gs.make\_move(move)  next\_moves = gs.get\_valid\_moves()  score = find\_move\_minmax(gs, next\_moves, depth - 1, True)  if score < min\_score:  min\_score = score  if depth == DEPTH:  next\_move = move  gs.undo\_move()  return min\_score   # A positive score good for white & a negative score good for black def score\_board(gs):  if gs.checkmate:  if gs.whiteToMove:  return -CHECKMATE # Black wins  else:  return CHECKMATE # White wins  elif gs.stalemate:  return STALEMATE   score = 0  for row in gs.board:  for square in row:  if square[0] == 'w': # White material advantage is positive  score += piece\_score[square[1]]  elif square[0] == 'b': # Black material advantage is negative  score -= piece\_score[square[1]]   return score   # Score the board based on material def score\_material(board):  score = 0  for row in board:  for square in row:  if square[0] == 'w': # White material advantage is positive  score += piece\_score[square[1]]  elif square[0] == 'b': # Black material advantage is negative  score -= piece\_score[square[1]]   return score |
| --- |

*код бракет.1*

## piece\_score dictionary:

Това е речник, който присвоява резултати на всеки тип шахматна фигура. Всеки ключ е низ, който представлява тип парче, а всяка стойност е число, което представлява резултата за това парче. Например царят има резултат 0, дамата има резултат 10 и т.н. Този резултат представлява относителната стойност на всяка фигура в играта на шах.

## Константи CHECKMATE и STALEMATE:

Това са постоянни стойности, които представляват резултатите за състоянията на играта на шах и мат. Ако играта е в състояние на мат, играчът, който е на път да загуби, получава резултат от -1000 (за черните) или +1000 (за белите). Ако играта е в патово състояние, и двамата играчи получават резултат 0.

## Константа DEPTH:

Това е постоянна стойност, която представлява нивото на дълбочина, което минимаксният алгоритъм трябва да търси. Колкото по-високо е нивото на дълбочина, толкова повече ходове алгоритъмът взема предвид напред, но толкова повече време отнема да намери най-добрия ход.

## Функция find\_best\_move\_minmax():

Тази функция взема състояние на играта (gs) и списък с валидни ходове и връща най-добрия ход за текущия играч, като използва алгоритъма minimax с алфа-бета съкращаване. Започва със задаване на глобалната променлива next\_move на None. След това извиква функцията find\_move\_minmax() с текущото състояние на играта, списъка с валидни ходове, константата DEPTH и булева стойност, показваща дали е ред на белия играч. Връща променливата next\_move, която сега трябва да съдържа най-добрия ход, намерен от алгоритъма.

## Функция find\_move\_minmax():

Това е рекурсивната функция, която прилага алгоритъма minimax с алфа-бета съкращаване(alpha-beta pruning). Изисква се състояние на играта (gs), списък с валидни ходове, ниво на дълбочина и булево значение, показващо дали е ред на белия играч. Започва с произволно разбъркване на списъка с валидни ходове (това е оптимизация на производителността, която не позволява на алгоритъма винаги да оценява едни и същи ходове в същия ред). След това проверява дали нивото на дълбочина е достигнало 0. Ако е така, връща резултата от текущото състояние на играта, който се изчислява с помощта на функцията score\_material(). Ако не, той стартира цикъл, който итерира всеки валиден ход. За всеки ход той прави хода върху копие на състоянието на играта (gs.make\_move(move)), получава списъка с валидни ходове за следващия играч (next\_moves = gs.get\_valid\_moves()) и рекурсивно се извиква с ново състояние на играта, новият списък с валидни ходове, нивото на дълбочина, намалено с 1, и противоположната стойност на white\_to\_move boolean. След това отменя хода (gs.undo\_move()) и продължава към следващия ход. По време на цикъла той следи максималния резултат за бяло (max\_score) и минималния резултат за черно (min\_score). Ако намери резултат, който е по-добър от текущия max\_score или min\_score, той съответно актуализира променливата. Ако нивото на дълбочина е равно на константата DEPTH, то също актуализира глобалната променлива next\_move до хода с най-добър резултат. Накрая връща max\_score, ако е ред на белия играч, или min\_score, ако е ред на черния играч.

## Функция score\_board():

Функцията score\_board(gs) изчислява резултата от състоянието на играта gs на базата на това дали играта е матирана, в безизходица или на материала на дъската. Ако играта е матирана, функцията връща резултат CHECKMATE за печелившия играч и CHECKMATE за губещия играч. Ако играта е в застой, функцията връща резултат STALEMATE. Ако нито едно от тези условия не е изпълнено, функцията преминава през всички квадратчета на дъската, добавя резултата на фигурата към променливата за резултата за белите фигури и изважда резултата на фигурата за черните фигури. Функцията връща крайния резултат.

## Функция score\_material(board):

Тази функция е по същество същото като score\_board(gs), с изключение на това, че приема дъска вместо състояние на играта като вход. Тази функция се използва в find\_move\_minmax(gs, valid\_moves, depth, white\_to\_move) за изчисляване на резултата за дадена позиция на дъската въз основа само на материал.

## (Castling) и (En Passant ).

| # en passant move  if move.is\_enpassant\_move: self.board[move.start\_row][move.end\_col] = "--" #capturing the pawn  # update enpassant\_possible variable  if move.piece\_moved[1] == "p" and abs(move.start\_row - move.end\_row) == 2:  # only on 2 square pawn advance  self.enpassant\_possible = ((move.start\_row + move.end\_row) // 2, move.start\_col) else: self.enpassant\_possible = ()  # castle move  if move.is\_castle\_move: if move.end\_col - move.start\_col == 2:  # king-side castle move  self.board[move.end\_row][move.end\_col - 1] = self.board[move.end\_row][ move.end\_col + 1]  # moves the rook to its new square  self.board[move.end\_row][move.end\_col + 1] = '--'  # erase old rook else:  # queen-side castle move  self.board[move.end\_row][move.end\_col + 1] = self.board[move.end\_row][ move.end\_col - 2]  # moves the rook to its new square  self.board[move.end\_row][move.end\_col - 2] = '--'  # erase old rook  self.enpassant\_possible\_log.append(self.enpassant\_possible)  # update castling rights - whenever it is a rook or king move  self.update\_castle\_rights(move) self.castle\_rights\_log.append(CastleRights(self.current\_castling\_rights.wks, self.current\_castling\_rights.bks, self.current\_castling\_rights.wqs, self.current\_castling\_rights.bqs)) |
| --- |

*код бракет.2*

За хода en passant, ако ходът е категоризиран като ход (en passant), уловената пешка се премахва от дъската чрез замяна на фигурата на целевото поле с празно поле ('--'). Освен това, ако движещата се фигура е пешка и тя се е преместила две полета напред, променливата enpassant\_possible се актуализира до полето зад преместената пешка, което може да се използва по-късно за улавяне на пешката en passant. За хода на рокада, ако ходът е извършен от цар, топът, участващ в хода, се премества на новото си поле и старата позиция се заменя с празно поле ('--'). Променливата enpassant\_possible също се актуализира чрез добавяне на последния възможен квадрат en passant.

След извършване на хода правата за рокада и правата за преминаване(en passant) се актуализират и съхраняват в регистрационни файлове за бъдещи справки. Това се прави с помощта на функцията update\_castle\_rights и списъците castle\_rights\_log и enpassant\_possible\_log. Функцията update\_castle\_rights актуализира правата за рокада, като проверява дали движещата се фигура е цар или топ и съответно актуализира правата. castle\_rights\_log съхранява текущите права за рокада след хода, а enpassant\_possible\_log съхранява текущото поле на (en passant) за всеки ход.

## Проверка за валиден ход

| def get\_valid\_moves(self):  # All moves considering checks.  temp\_castle\_rights = CastleRights(self.current\_castling\_rights.wks, self.current\_castling\_rights.bks,  self.current\_castling\_rights.wqs, self.current\_castling\_rights.bqs)  # advanced algorithm  moves = []  self.in\_check, self.pins, self.checks = self.check\_for\_pins\_and\_checks()   if self.white\_to\_move:  king\_row = self.white\_king\_location[0]  king\_col = self.white\_king\_location[1]  else:  king\_row = self.black\_king\_location[0]  king\_col = self.black\_king\_location[1]  if self.in\_check:  if len(self.checks) == 1: # only 1 check, block the check or move the king  moves = self.get\_all\_possible\_moves()  # to block the check you must put a piece into one of the squares between the enemy piece and your king  check = self.checks[0] # check information  check\_row = check[0]  check\_col = check[1]  piece\_checking = self.board[check\_row][check\_col]  valid\_squares = [] # squares that pieces can move to  # if knight, must capture the knight or move your king, other pieces can be blocked  if piece\_checking[1] == "N":  valid\_squares = [(check\_row, check\_col)]  else:  for i in range(1, 8):  valid\_square = (king\_row + check[2] \* i,  king\_col + check[3] \* i) # check[2] and check[3] are the check directions  valid\_squares.append(valid\_square)  if valid\_square[0] == check\_row and valid\_square[1] == check\_col: # get to piece and check  break  # get rid of any moves that don't block check or move king  for i in range(len(moves) - 1, -1, -1): # iterate through the list backwards when removing elements  if moves[i].piece\_moved[1] != "K": # move doesn't move king, it must block or capture  if not (moves[i].end\_row,  moves[i].end\_col) in valid\_squares: # move doesn't block or capture piece  moves.remove(moves[i])  else: # double check, king has to move  self.get\_king\_moves(king\_row, king\_col, moves)  else: # not in check - all moves are fine  moves = self.get\_all\_possible\_moves()  if self.white\_to\_move:  self.get\_castle\_moves(self.white\_king\_location[0], self.white\_king\_location[1], moves)  else:  self.get\_castle\_moves(self.black\_king\_location[0], self.black\_king\_location[1], moves)   if len(moves) == 0:  if self.under\_check():  self.checkmate = True  else:  self.stalemate = True  else:  self.checkmate = False  self.stalemate = False   self.current\_castling\_rights = temp\_castle\_rights |
| --- |

Методът започва със създаване на копие на текущите права за рокада, след което проверява дали текущият играч е в шах чрез извикване на метод check\_for\_pins\_and\_checks(). Ако играчът е в шах, той проверява броя на шаховете:

### Ако има само един шах,

Методът генерира всички възможни ходове и филтрира тези, които не блокират шаха или изместват царя извън шаха. Ако проверяващата фигура е кон, единственият валиден ход е да се улови конят или да се премести царят.

### Ако има две проверки,

Единственият валиден ход е да се извади царят извън шах, така че методът get\_king\_moves() се извиква, за да добави всички възможни ходове на царя към списъка с валидни ходове.

### Ако играчът не е в шах,

Методът генерира всички възможни ходове за играча и проверява дали рокадата е валиден ход, като извиква метода get\_castle\_moves(). И накрая, ако няма валидни ходове, методът проверява дали играчът е под мат или пат. Ако има валидни ходове, флаговете за мат и пат се задават на False.

Като цяло, този метод осигурява цялостно прилагане на правилата на шаха за генериране на валидни ходове в дадено състояние на играта, като се вземат предвид шахове, кегли(pins) и права за рокада.

## highlight\_squares

highlight\_squares е функция, която приема четири аргумента: screen, който е повърхността, където се показва играта, gs, който е екземпляр на класа GameState, представящ текущото състояние на играта, valid\_moves, който е списък на всички валидни ходове за текущия player и square\_selected, което е кортеж, представящ координатите на квадрата, който играчът е избрал.

Функцията проверява дали square\_selected не е празен кортеж, което означава, че е избран квадрат. Ако избраният квадрат съдържа фигура, която принадлежи на текущия играч, функцията създава жълт квадрат с прозрачност и го зачерква върху повърхността в координатите на избрания квадрат.

| def highlight\_squares(screen, gs, valid\_moves, square\_selected):  # Highlight square selected and moves for piece selected.   if square\_selected != ():  r, c = square\_selected  if gs.board[r][c][0] == ('w' if gs.white\_to\_move else 'b'): # square\_selected is a piece that can be moved  # highlight selected square  s = p.Surface((SQ\_SIZE, SQ\_SIZE))  s.set\_alpha(100) # transparency value 0 -> transparent, 255 -> opaque  s.fill(p.Color('yellow'))  screen.blit(s, (c \* SQ\_SIZE, r \* SQ\_SIZE))  # highlight moves from that square  s.fill(p.Color('green'))  for m in valid\_moves:  if m.start\_row == r and m.start\_col == c:  screen.blit(s, (m.end\_col \* SQ\_SIZE, m.end\_row \* SQ\_SIZE))   def draw\_game\_state(screen, gs, valid\_moves, square\_selected, colors):  draw\_board(screen, colors) # Draws the squares on the board  highlight\_squares(screen, gs, valid\_moves, square\_selected) # highlight the squares  draw\_pieces(screen, gs.board) # Draws pieces on top of those squares |
| --- |

След това функцията създава зелен квадрат с прозрачност и го зачерква върху повърхността в координатите на всички квадрати, които са валидни ходове за фигурата на избрания квадрат. Това се прави чрез итериране на всички валидни ходове и проверка дали техните начални координати съвпадат с координатите на избрания квадрат.

draw\_game\_state е функция, която също приема четири аргумента: screen, който е повърхността, където се показва играта, gs, който е екземпляр на класа GameState, представящ текущото състояние на играта, valid\_moves, който е списък на всички валидни ходове за текущ играч, square\_selected, който е кортеж, представляващ координатите на квадрата, който играчът е избрал, и colors, който е речник, съдържащ цветовете на шахматната дъска.

Функцията първо извиква draw\_board, който рисува квадратите на шахматната дъска върху повърхността според цветовете в речника на цветовете.

След това извиква highlight\_squares, който подчертава избрания квадрат и валидните ходове за избраната фигура чрез изчертаване на жълти и зелени квадрати върху повърхността.

И накрая, той извиква draw\_pieces, който рисува шахматните фигури върху квадратите на повърхността, според текущото състояние на играта, представено от gs.board.

## Внедряване (deployment)

Цялото приложение е направено на лесно достъпен EXE File който може да бъде намерен на флашката в архивирана папка с име (Game exe), от там разархивираме папката и намираме файла на име Main(важно е да не се вади Main файла извън Game Exe папката, понеже файловете вътре за нужни за правилното стартиране на приложението) и го отваряме. Приложението тръгва без нуждата да се инсталират външни пакети.

# Глава 4. Ръководство за стартиране на програмата

Програмата е разработвана изцяло на Python и като допълнителни библиотеки са използвани Pylance и pygame. След като успешно инсталирате гореспоменатите пакети и отворите проекта през VS Code. От там просто трябва да се отвори Main.py и да се пусне през конзолата с команда ctrl+F5. Ако изберете да играете (PVP) има опция за връщане на ход с натискането на клавиша Z.

Кратък Guide за инсталиране на pygame: <https://www.pygame.org/wiki/GettingStarted>

Source Code: <https://github.com/YeetmaisterMartin/Martin-Nedelinov-Petrov-12V-Graduation-project>

# Заключение

Програмата използва шахматен двигател, за да анализира състоянието на дъската и да генерира възможни ходове и оценява тези ходове, използвайки техники като минимаксен алгоритъм, алфа-бета съкращаване и машинно обучение. За момента изкуствения интелект винаги играе с черните фигури, защото трябва да получи input от хода на противника за да може да реагира адекватно. Като възможности за усъвършенстване може да се напише Opening Book function който ще даде възможност на изкуствения интелект(AI) да започне първи с белите фигури. Друга възможна оптимизация е добавянето на опция за издигане на пешките в други фигури.(за момента пешките директно стават царици като стигнат до края на дъската).

# Информационни източници

1. [Coding a Complete Chess Game AI With Python (Part 1) | PVP Game Mode](https://www.youtube.com/watch?v=OpL0Gcfn4B4&t=358s)
2. [I Made a Weird Chess AI from Scratch](https://www.youtube.com/watch?v=t6JA78oRFMI&t=180s)
3. <https://youtube.com/playlist?list=PLBwF487qi8MGU81nDGaeNE1EnNEPYWKY_>
4. <https://blog.devgenius.io/simple-min-max-chess-ai-in-python-2910a3602641>
5. <https://medium.com/dscvitpune/lets-create-a-chess-ai-8542a12afef>
6. [Making an Executable from a Pygame Game (PyInstaller)](https://www.youtube.com/watch?v=lTxaran0Cig)

# Рецензия на дипломен проект

| **Тема на дипломния проект** |  | | |
| --- | --- | --- | --- |
| **Ученик** |  | | |
| **Клас** |  | | |
| **Професия** |  | | |
| **Специалност** |  | | |
| **Ръководител- консултант** |  | | |
| **Рецензент** |  | | |
| **Критерии за допускане до защита на дипломен проект** | | **Да** | **Не** |
| Съответствие на съдържанието и точките от заданието | |  |  |
| Съответствие между тема и съдържание | |  |  |
| Спазване на препоръчителния обем на дипломния проект | |  |  |
| Спазване на изискванията за оформление на дипломния проект | |  |  |
| Готовност за защита на дипломния проект | |  |  |
| Силни страни на дипломния проект |  | | |
| Допуснати основни слабости |  | | |
| Въпроси и препоръки към дипломния проект |  | | |

**ЗАКЛЮЧЕНИЕ:**

Качествата на дипломния проект дават основание ученикът/ ученичката ................................................... да бъде допуснат/а до защита пред членовете на комисията за подготовка, провеждане и оценяване на изпит чрез защита на дипломен проект- част по теория на професията.

.........05.2023г. Рецензент:................................

град София