Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**Proiect de curs**

*Disciplina: Analiza și Proiectarea Algoritmilor*

**Tema: Elaborarea unei aplicații software destinată rezolvării unei probleme utilizând diverși algoritmi ,,Joc de dame’’**

**Developing a software application to solve a problem using various algorithms ,,Checkers’’**

Au efectuat: Cozma Daniel

Untilă Victor

Ștefan Vlașițchi

Grupa TI-216

A verificat: asist. univ. Cebotari Daria

Chișinău – 2023

**Cuprins**

[**Introducere 3**](#_Toc125635761)

[**1. Analiza domeniului de studiu 4**](#_Toc125635762)

[**1.1 Formularea problemei prin abstractizare uneii situații 4**](#_Toc125635763)

[**1.2 Definirea datelor problemei şi stabilirea relaţiei între date şi soluţia problemei 5**](#_Toc125635764)

[**2 .Algoritmul sistemului 6**](#_Toc125635765)

[**2.1 Alegerea algoritmilor pentru rezolvarea problemei 7**](#_Toc125635766)

[**2.2 Argumentarea alegerii algoritmilor 8**](#_Toc125635767)

[**3 Realizarea sistemului 8**](#_Toc125635768)

[**3.1 Implementarea algoritmilor 10**](#_Toc125635769)

[**3.2 Interfața grafica a aplicației 13**](#_Toc125635770)

[**3.3 Analiza rezultatelor obtinute 13**](#_Toc125635771)

[**Concluzii 15**](#_Toc125635772)

[**Bibliografie 16**](#_Toc125635773)

[**Codul sursa:Anexa A 17**](#_Toc125635774)

# Introducere

Software-ul este un program sau un set de programe care rulează prin intermediul sistemului de operare al calculatorului sau dispozitivului electronic pe care este instalat. Sistemul de operare este un program care gestionează resursele hardware ale calculatorului, cum ar fi memoria și procesorul, și oferă o interfață pentru utilizator pentru a interacționa cu acestea.

Când un software este lansat, sistemul de operare îl "alocă" și îl "rulează" asignând resursele necesare. Software-ul rulează instrucțiunile din codul său și utilizează resursele alocate pentru a efectua sarcinile specifice pentru care a fost conceput.

De exemplu, un software de procesare de text va utiliza memoria și procesorul pentru a permite utilizatorului să scrie și să editeze documente, iar un joc va utiliza procesorul și placa video pentru a afișa grafica și animațiile.[1]

Există o varietate de alte tipuri de software, cum ar fi software-ul pentru jocuri:Minecraft,Fortnite, software-ul educațional:Moodle, software-ul de design grafic:AutoCAD,Blender, software-ul de birou:Pachetul Microsoft Office, software-ul pentru sistem de navigare:Google Chrome,Firefox. Acestea sunt doar câteva exemple dintre cele mai populare tipuri de software disponibile.

Lucrarea dată urmărește scopul de a elabora crearea unui joc video ,,Joc de dame’’ prin utilizarea unui algoritm care obiectivele jocului și regulile acestuia să genereze elementele jocului și să controleze comportamentul acestora.

Lucrarea este structurată în trei capitole, unde se va analiza concret modul de proiectare și de realizare a poiectului dat.

Primul capitole vine să stabilească obiectivele propuse a algoritmului și să definească noțiunea de algoritm, necesitatea lui și cum poate fi utilizat. De asemenea acest capitol urmărește formularea problemei propuse.

Al doilea capitol include algoritmii care vor putea fi utilizați în rezolvarea problemei noastre, descrierea fiecăruia în parte, compararea lor și identificarea celui mai eficient pentru soluționarea problemei.

În ultimul capitol se conține partea de implementarea a codului, realizarea și prezentarea sarcinii propuse, evidențierea și descrierea modului de utilizarea a algorimilor și analiza rezultatelor obținute în urma aplicării algoritmilor.

1. **Analiza domeniului de studiu**

Un algoritm este o secvență de instrucțiuni bine definite și finită, care descrie un mod de a rezolva o problemă sau a atinge un obiectiv. Algoritmii sunt utilizați în informatică, matematică și alte domenii pentru a automatiza procese și a găsi soluții eficiente.

In programare, un algoritm rezolvă o problemă sau atinge un obiectiv utilizând un calculator sau un sistem informatic. Acestea sunt utilizate pentru a automatiza procese, pentru a găsi soluții eficiente și pentru a realiza diverse operații, cum ar fi sortarea, căutarea sau procesarea de date. Algoritmii sunt esențiali pentru funcționarea sistemelor informatice și a aplicațiilor software.

Algoritmii pot fi clasificați în mai multe categorii, în funcție de diverse criterii, cum ar fi:

* După scopul lor: algoritmii de sortare, căutare, procesare de date etc.
* După complexitatea lor: algoritmii simpli și algoritmii complexi.
* După modul de funcționare: algoritmii recursivi și algoritmii iterativi.
* După natura datelor: algoritmii pentru date structurate și algoritmii pentru date ne-structurate.
* După natura problemei: algoritmii de optimizare, algoritmii de predicție, algoritmii de clustering etc.
* Dupa modul de funcționare : algoritmii deterministi si algoritmii stocastici

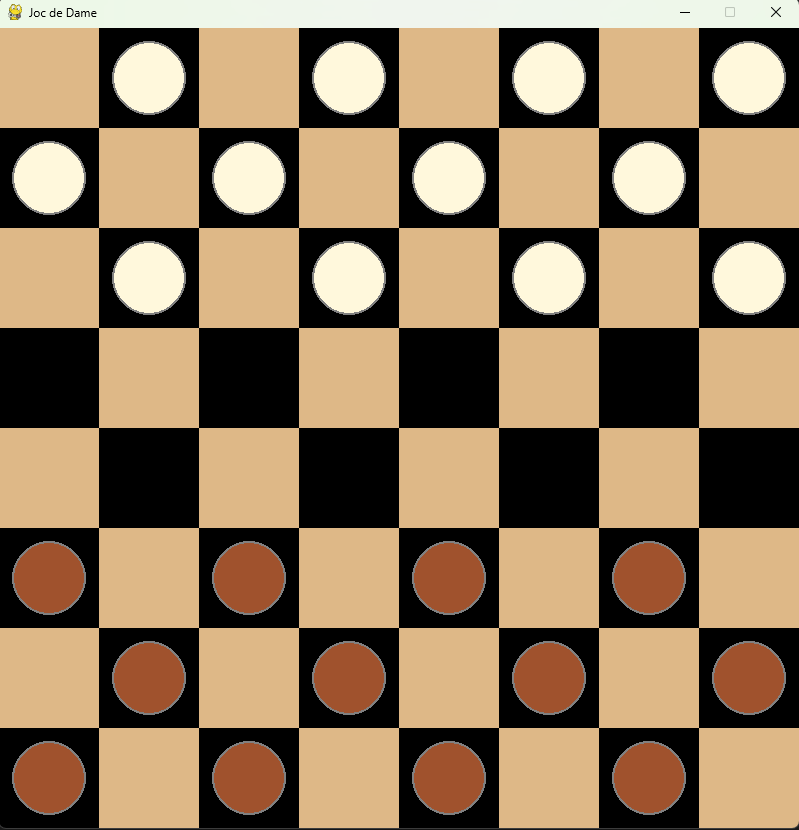
Acestea sunt doar câteva exemple de cum pot fi clasificate algoritmii, existând mai multe alte clasificari posibile.

Algoritmii în practică pot fi utilizați pentru sarcini precum data mining, învățarea automată și inteligența artificială. De asemenea, sunt utilizate în multe aplicații de zi cu zi, cum ar fi căutarea pe web, rețelele sociale și cumpărăturile online. În concluzie, algoritmii sunt un instrument esențial pentru rezolvarea problemelor din informatică și sunt utilizați în multe domenii diferite ale domeniului.

* 1. **Formularea problemei prin abstractizare uneii situații**

Jocul de dame este un joc de strategie pentru două persoane se joacă pe tabla de joc cu 8x8 celule 64 de căsuțe albe și negre, împărțită în patru zone, fiecare jucător având 12 piese pe culoarea lor (de obicei alb sau negru).(Fig. 1*)* Jocul de dame are o istorie lungă și incertă care datează din Antichitat, jocul fiind jucat în diferite forme în multe culturi diferite, inclusiv India, China, Egiptul antic și Grecia.În prezent, jocul de dame este jucat la nivel mondial, cu competiții și campionate organizate la nivel international existând mai multe variante ale jocului, cum ar fi dame americane, dame britanice și dame rusești, care au reguli reguli diferite.

Scopul jocului dar și a problemei noastre este de a captura toate piesele adversarului sau de a-l bloca astfel încât să nu mai poată face nici o mișcare. Piesele se mișcă numai în diagonală și pot sări peste alte piese pentru a le captura. Când o piesă ajunge în ultima coloană de pe tabla adversarului, aceasta se transformă într-o "damă" și se poate mișca atât în diagonală cât și înainte și înapoi. Câștigătorul este cel care capturează toate piesele adversarului sau îl blochează astfel încât să nu mai poată face nici o mișcare.[2]



*Fig. 1. Jocul de Dame*

## Definirea datelor problemei şi stabilirea relaţiei între date şi soluţia problemei

Obiectivele care trebuie de urmat pentru a ajunge la suloționarea problemei sunt:

* Dimensiunea plăcii trebuie să fie fixată de pătrat.
* Sunt 3 direcți în jocul de dame:pe diagonală atăt pe dreapta dar și pe stânga dar și în cazul când o piesă ajunge în ultima coloană de pe tabla adversarului, aceasta se transformă într-o "damă" și poate mișca atât în diagonală cât și înainte și înapoi.
* Încearci să captureze toate piesele sau să blochezi piesele adversarului astfel încât să nu mai poată face nici o mișcare iar adversarul având același scop.

Problema jocului de dame este definită ca o reprezentare matematică a unui joc de dame, care include regulile și constrângerile acestuia. Aceasta poate fi utilizată pentru a analiza și a găsi soluții pentru jocul de dame.Soluționarea problemei de dame implică utilizarea algoritmilor și a tehnicilor de inteligență artificială pentru a găsi cele mai bune mișcări posibile și pentru a prezice rezultatul jocului. Un exemplu de astfel de algoritm este algoritmul Minimax, care evaluează toate mișcările posibile pentru fiecare jucător și alege cea mai bună mișcare pentru a câștiga jocul sau pentru a obține cel mai bun rezultat posibil.

# .Algoritmul sistemului

Algoritmii sunt utilizați în programare pentru a automatiza procesele și pentru a rezolva probleme complexe.Programatorii folosesc algoritmi pentru a scrie programe care pot rezolva probleme specifice sau care pot efectua sarcini specifice. De exemplu, un programator poate scrie un algoritm care sortează elementele dintr-un array sau care caută un element specific într-o bază de date.În general, algoritmii sunt esențiali pentru programatori deoarece aceștia ajută la crearea unui cod eficient, ușor de înțeles și de menținut, care poate rezolva probleme complexe.

În cazul problemei noastre există o varietate de probleme care pot fi întâlnite atunci când se utilizează un algoritm în codul de programare, cum ar fi:

* Probleme de performanță: algoritmul poate fi prea lent sau consuma prea multă memorie, fapt care poate afecta performanța aplicației.
* Erori de logică: algoritmul poate conține greșeli de logică, care pot duce la rezultate incorecte sau la comportament neașteptat al aplicației.
* Probleme cu datele de intrare: algoritmul poate fi proiectat pentru anumite tipuri de date de intrare, iar utilizarea altor tipuri de date poate duce la erori sau la comportament neașteptat.

Conform problemei noastre ,,Jocul de dame’’ se joacă împotriva unui adversar uman sau unui bot(program de calculator)fiind cazul nostru,iar acest adversar(bot-ul) programarea unuia pentru a juca un joc de dame poate fi un proces complex, deoarece jocul are multe reguli și posibilități.Iar pentru realizarea acestui bot trebuie:

* Studierea regulilelor jocului de dame și modul în care acesta este jucat indiferent dacă știm jocul sau nu.
* De creat o structură de date care să reprezinte tabla de joc și să țină evidența pozițiilor pieselor și a mișcărilor valide.
* Implementarea unui algoritm de înțelepciune artificială adecvat, cum ar fi algoritmul ,,Minimax’’ sau algoritmul ,,Monte Carlo Tree Search’’ (MCTS) pentru a determina cele mai bune mișcări pentru bot
* Optimizare a algoritmului, pentru a reduce numărul de posibilități de joc care trebuie evaluate.
* Asigurarea că botul poate efectua toate mișcările valide din jocul de dame.
* Testarea bot-lui împotriva unui număr mare de oponenți umani pentru a ne asigura că acesta poate juca un joc competitiv.
* Asigurarea codului că este bine documentat și ușor de întreținut pentru a facilita modificările și actualizările ulterioare.

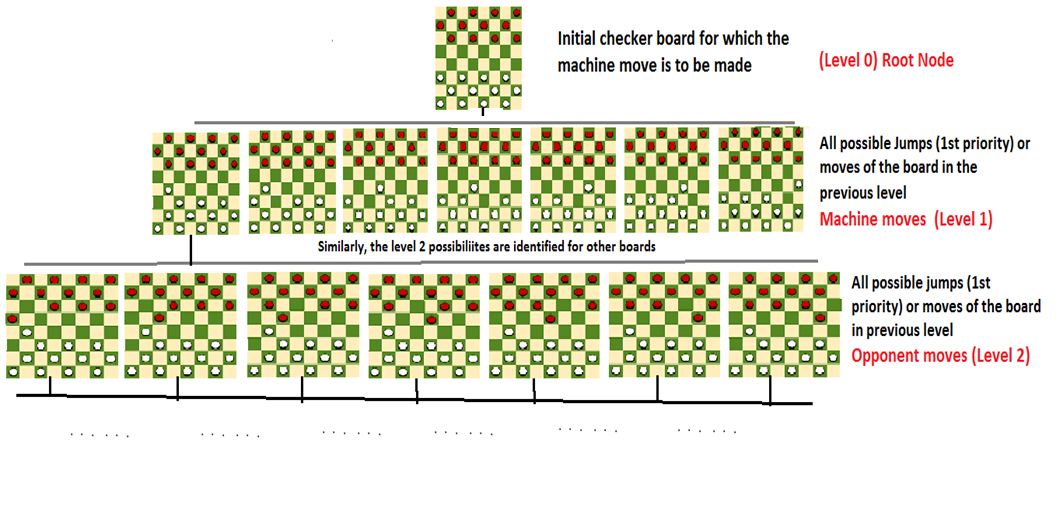
## Alegerea algoritmilor pentru rezolvarea problemei

Pentru a automatiza procesele jocului de dame și pentru a rezolva problema complexă am decis să ne conducem după metoda algoritmului ,,Minimax’’.

Algoritmul ,,Minimax’’[Fig. 2.] este un algoritm de decizie utilizat în programarea jocurilor de calculator, care evaluează toate posibilitățile de mutare și alege cea care va avea cel mai bun rezultat posibil pentru bot și cel mai prost rezultat posibil pentru oponent. Acest algoritm se bazează pe ideea că, dacă oponentul își va alege cea mai bună mișcare posibilă, bot-ul ar trebui să aleagă cea mai rea mișcare posibilă pentru oponent.

În cazul jocului de dame, algoritmul ,,Minimax’’[Fig. 2] poate fi utilizat pentru a evalua toate posibilitățile de mutare și pentru a alege cea mai bună mișcare posibilă pentru bot, ținând cont de starea actuală a jocului, cum ar fi poziția pieselor și numărul de piese rămase pe tabla de joc.Pentru a implementa algoritmul ,,Minimax’’ într-un joc de dame, trebuie să se definească o funcție de evaluare care să evalueze starea jocului și să returneze un scor pentru acea stare. Apoi, trebuie să se parcurgă recursiv toate posibilitățile de mutare și să se evalueze fiecare stare rezultată, returnând scorul care va fi cel mai bun pentru bot și cel mai rău pentru oponent.Cu toate acestea, algoritmul ,,Minimax’’[Fig. 2.] poate fi foarte costisitor din punct de vedere al memoriei și al timpului de procesare, deoarece trebuie să evalueze toate posibilitățile de mutare.[4][5]

În imaginea următoare este reprezentat modul de funcționare al algoritmului ,,Minimax’’ într-un joc de dame.



*Fig.2 Algoritmul minimax*

*Fig. 2.Modul de funcționare a algoritmului minimax*

## Argumentarea alegerii algoritmilor

Algoritmul Minimax este cel mai eficient pentru a crea un joc de dame deoarece:

* Minimax analizează toate posibilitățile de joc și determină valoarea fiecărei mișcări. Acest lucru permite jucătorului să facă cea mai bună mișcare posibilă, în funcție de mișcările anticipate ale oponentului.
* Acest algoritm este eficient pentru jocurile cu doi jucători și poate fi utilizat pentru a determina câștigătorul într-un joc de dame.
* Un plus, algoritmul Minimax poate fi optimizat care poate reduce numărul de posibilități care trebuie evaluate și astfel poate crește eficiența algoritmului.

În concluzie, algoritmul ,,Minimax’’ este una dintre cele mai bune opțiuni pentru a crea un joc de dame, deoarece acesta poate anticipa mișcările oponentului și poate alege cea mai bună mișcare pentru jucătorul curent.

# Realizarea sistemului

Programul jocului “Joc de Dame” a fost realizat cu ajutorul limbajului Python, acesta este considerat unul dintre cele mai bune limbaje de programare din cauza următoarelor caracteristici:

* Ușor de învățat: Sintaxa Python este ușor de înțeles și permite scrierea codului rapid.
* Versatilitate: Python poate fi utilizat pentru o varietate de aplicații, cum ar fi dezvoltarea web, analiza de date, inteligența artificială și automatizarea proceselor.
* Comunitate puternică: Există o comunitate mare și activă de dezvoltatori Python care contribuie la dezvoltarea și îmbunătățirea limbajului.
* Interoperabilitate: Python este compatibil cu alte limbaje de programare, cum ar fi C ++ și Java, ceea ce permite integrarea cu alte sisteme și aplicații.
* Documentație bună: Python oferă o documentație detaliată și exemple de cod, care fac mai ușoară înțelegerea și utilizarea limbajului.

Python oferă o mulțime de biblioteci și instrumente pentru dezvoltare, care fac mai ușoară crearea aplicațiilor complexe spre exemlpu: Numpy – o bibliotecă de calcul numeric care permite crearea și manipularea matricelor și vectorilor, Scikit-learn - o bibliotecă pentru învățare automată care oferă algoritmi de clasificare, regresie și clustering, TensorFlow - o bibliotecă pentru inteligența artificială și învățare automată dezvoltată de Google.

În cazul problemei de rezolvat am folosit biblioteca Pygame, aceasta reprezintă un set de module Python proiectate pentru scrierea de jocuri video. Este construit pe baza bibliotecii SDL (Simple DirectMedia Layer), care oferă acces la nivel scăzut la funcții de audio, tastatură, mouse și afișare. Pygame oferă o interfață la nivel mai ridicat pentru dezvoltarea de jocuri, facându-l ușor să se creeze jocuri cu grafică bogată, sunet și interacțiune.

Unul dintre principalele avantaje ale Pygame este simplitatea sa. Biblioteca oferă un set de funcții ușor de utilizat pentru crearea de jocuri, făcând-o accesibilă atât pentru începători cât și pentru progamatorii experimentați. De asemenea, are o comunitate mare și activă, care oferă o bogăție de resurse, tutoriale și suport.

Pygame suportă o gamă largă de platforme, incluzând Windows, Mac și Linux. Poate fi utilizat pentru a crea jocuri pentru PC, precum și pentru dispozitive mobile și web. Biblioteca este, de asemenea, open-source și gratuită de utilizat, făcând-o o alegere populară printre dezvoltatorii de jocuri indie.

Biblioteca oferă o varietate de module utile pentru dezvoltarea de jocuri, inclusiv:

• Afișare: pentru crearea și gestionarea fereastra jocului

• Suprafață: pentru crearea și manipularea de imagini și grafică

• Desen: pentru desenarea formelor și linii pe ecran

• Sprite: pentru crearea și gestionarea grupurilor de obiecte într-un joc

• Rect: pentru lucrul cu zone dreptunghiulare ale ecranului

• Eveniment: pentru gestionarea intrare utilizator, cum ar fi evenimente de tastatură și mouse

• Sunet: pentru redarea și controlul fișierelor audio

• Mixer: pentru amestecarea mai multor piste audio împreună

• Timp: pentru măsurarea timpului și controlul buclei de joc

Pygame suportă, de asemenea, o gamă largă de formate de fișiere, inclusiv PNG, JPG, BMP și WAV. Acest lucru facilitează utilizarea elementelor grafice și audio existente în jocurile dvs.

Pe lângă biblioteca de bază, Pygame oferă și o serie de module și biblioteci suplimentare care pot fi folosite pentru a vă îmbunătăți jocurile. Acestea includ biblioteci pentru lucrul cu grafică 3D, fizică și rețele.

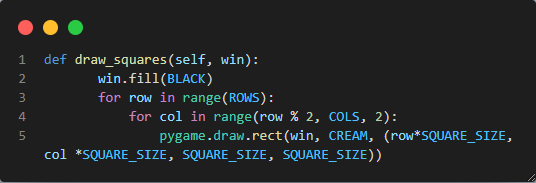
În general, Pygame este o bibliotecă puternică și flexibilă pentru dezvoltarea jocurilor, care oferă o interfață simplă și accesibilă pentru crearea de jocuri. Gama sa largă de module și suportul pentru mai multe platforme îl fac o alegere populară printre dezvoltatorii de jocuri[3]

* 1. **Implementarea algoritmilor**

În următorul sub capitol vom descrie minuțios modul de operare a aplicației noastre.

Board – clasa dată reprezintă table de joc, aceasta se ocupă de modul în care se mișcă piesele, de mutarea acestora si ștergerea lor.

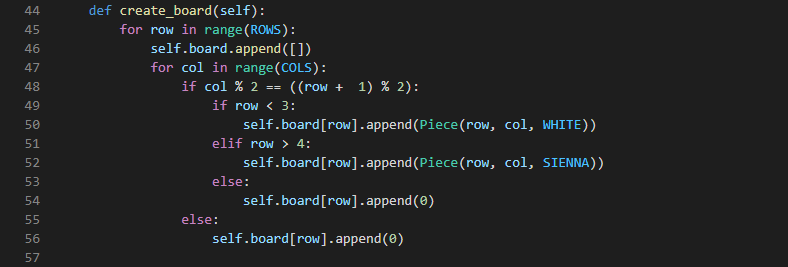
Cu ajutorul acestei clase [Fig. 3] vom pregăti tabla de joc, în special cu un mic algoritm recursiv, în care pentru început se umple cu culoarea neagră iar mai apoi cu ajutorul unui ,,for’’ se parcurge după rânduri și coloane umplând spațiile cu dimensiunea specificată, în primul ,,for” se parcurge întreaga matrice "ROWS" și în al doilea ,,for” se parcurge matricea ,,col” care începe de la ,,row % 2” și se incrementează cu 2 după fiecare iterație.



*Fig. 3. Metoda “draw\_squares*

Metoda ,,create\_board”[Fig. 4] este utilizată pentru a inițializarea tablei de joc și a plasa piesele în pozițiile inițiale.

Ea utilizează două bucle ,,for” pentru a parcurge fiecare linie și coloană a tabelei. Pentru fiecare element din matrice, se verifică dacă acesta este un patrat valid pentru a plasa o piesă. Dacă este valid, se verifică linia curentă pentru a determina dacă trebuie să plaseze o piesă albă sau siena. Dacă nu este un pătrat valid, se adaugă ,,0” în matrice.

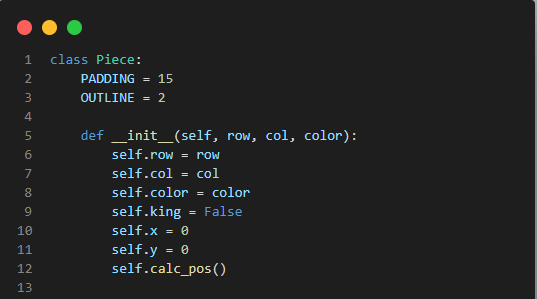
La finalul metodei, tabla de joc este inițializată cu piesele plasate în pozițiile inițiale.

*Fig. 4. Metoda “create\_board”*

În cele din urma ultima metoda din această clasă sunt ,,\_traverse\_left” și ,,\_traverse\_right” utilizate pentru a determina mutările valide pentru o anumită piesă. Ele parcurg linia în partea stângă sau dreaptă a piesei și verifică dacă există piese inamice sau poziții goale pentru a muta piesa.

Piece - această clasă reprezintă pieasa unui joc de dame care conține elemente care ajută la ilustrarea acestor piese

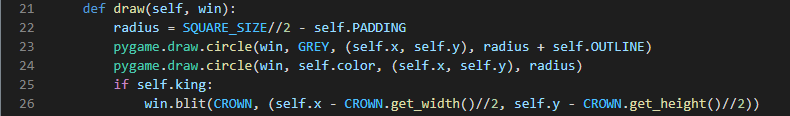
În această clasă[Fig. 5] se pastreză informația despre fiecare piesa, modul în care acesta arată și cum se schimba din piesă în damă.



*Fig. 5. Metoda “init”*

Funcția "init" [Fig. 5.] este constructorul clasei și este apelat atunci când o instanță a clasei este creată. Aceasta primește argumente pentru poziția piesei pe tabla de joc (rând și coloană) și culoarea piesei. Variabilele "self.row", "self.col", "self.color", "self.king", "self.x", "self.y" sunt initializate cu valorile date ca argumente. Apoi, se apelează metoda "calc\_pos" care calculează poziția exacta a piesei pe tabla de joc.

Metoda "make\_king" setează variabila "self.king" la "True", marcând astfel ca piesa este o damă.



*Fig. 6. Metoda “draw”*

Metoda ,,draw”[Fig. 6.] desenează piesa pe fereastra de joc ,,win” utilizând biblioteca Pygame.Desenând un cerc cu o culoarea anumită și un alt cerc cu culoarea inițială a piesei, iar dacă piesa este marcată ca damă se desenează o coroniță.

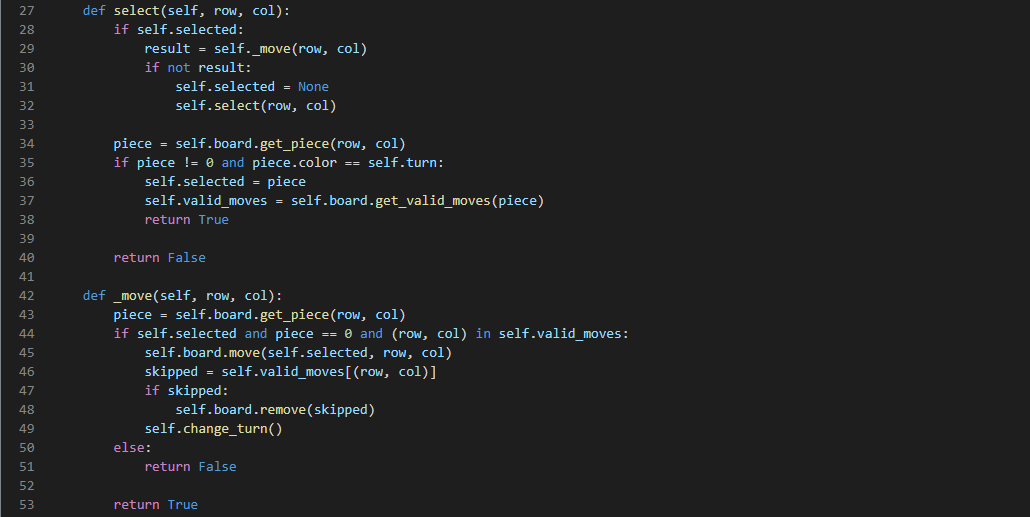


*Fig 7. Metoda “move”*

Metoda "move" [Fig. 7.] actualizează poziția a piesei pe tabla de joc și apoi se apelează metoda "calc\_pos" pentru a actualiza poziția exactă.

Game- clasa respectivă reprezintă jocul în sine

Pentru început vom avea constructorul ,,init” în care se inițializează variabilele necesare a jocului, cum ar fi tabla de joc, piesa selectată, culoarea căruia va permite să mută în momentul respectiv și mutările valide pentru piesa selectată.



*Fig 8. Metoda “Select”*

Metoda "select" [Fig. 8.]este utilizată pentru a selecta o piesă de pe tabla de joc. Dacă o piesă este deja selectată și se face o mutare validă, se schimbă culoarea căruia i se va permite să mută.

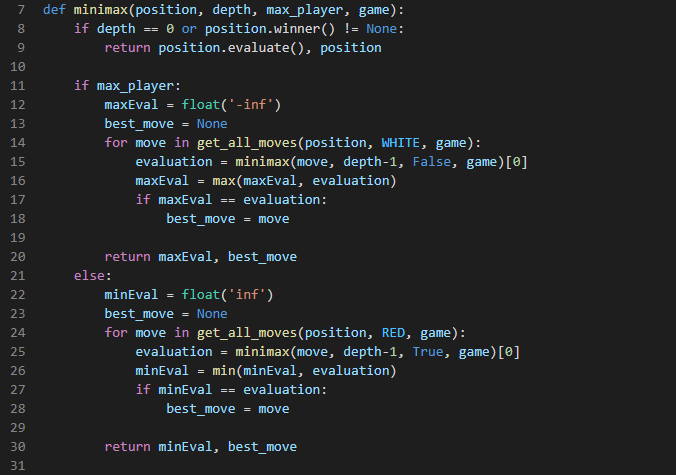
Metoda "ai\_move" [Fig. 9] este utilizată pentru a permite unui ,,AI” să facă o mutare.



*Fig 9. Metoda “ai\_move”*

La finalul jocului, metoda "winner" este utilizată pentru a determina cine a câștigat jocul.

În cele din urma avem implementat algoritmul ,,Minimax” pentru a determina cele mai bune mutări în jocul de dame. ,,Minimax” este un algoritm[Fig. 10.] de cautare în arbore care evaluează toate posibilitațile de joc și alege cea mai bună mutare pentru jucătorul current în funcție de poziția curentă.



*Fig 10. Functia “Minimax”*

La început, dacă adancimea este ,,0” sau dacă jocul s-a terminat, atunci funcția returnează scorul poziției curente si poziția.[Fig. 11.]

Dacă jucătorul curent este cel care dorește să maximizeze scorul, funcția va parcurge toate mutările posibile își va alege cea cu scorul maxim, returnând acest scor și poziția. La fel, dacă jucătorul curent este cel care dorește să minimizeze scorul, funcția va parcurge toate mutările posibile și va alege cea cu scorul minim, returnând acest scor și poziția.



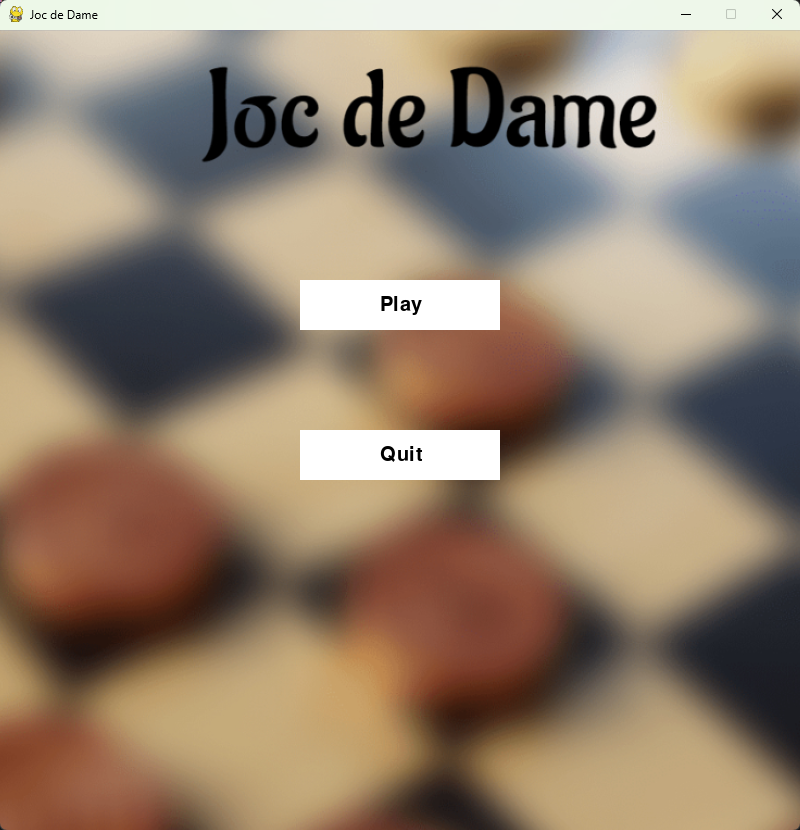
\

*Fig 11. Metoda “Minimax”*

* 1. **Interfața grafica a aplicației**

Interfața grafică a unei aplicații de joc de dame ar trebui să fie clară, intuitivă și ușor de utilizat pe care sunt afișate butoane sau opțiuni,o interfață bine proiectată poate face jocul mai ușor de înțeles și de jucat pentru utilizatori, indiferent de nivelul lor de experiență cu jocurile de dame. Există mai multe modalități de a crea o interfață grafică utilizator în Python,pentru a face interfața grafică a aplicației am folosit deasemenea limbajul Python,am creat o nouă clasă ,,Home” în care am creat 2 butoane complet funcționale, unul pentru a începe jocul și altu pentru a închide aplicația, deasemnea am creat o nouă variabilă background în care am salvat o imagine care servește acum ca o imagine de fundal în meniul aplicatiei noastre.Toate acestea au fost setate cu ajutorul bibliotecii pygame.

* 1. **Analiza rezultatelor obtinute**

Accesând aplicația [Fig. 12] rulează direct pe ecran, fără a fi nevoie de intreg codul al jocului.La pornirea acestuia suntem întălniți de un meniu modest în urma caruia alegem daca începem jocul sau închidem aplicația

*Fig 12. Meniul jocului*

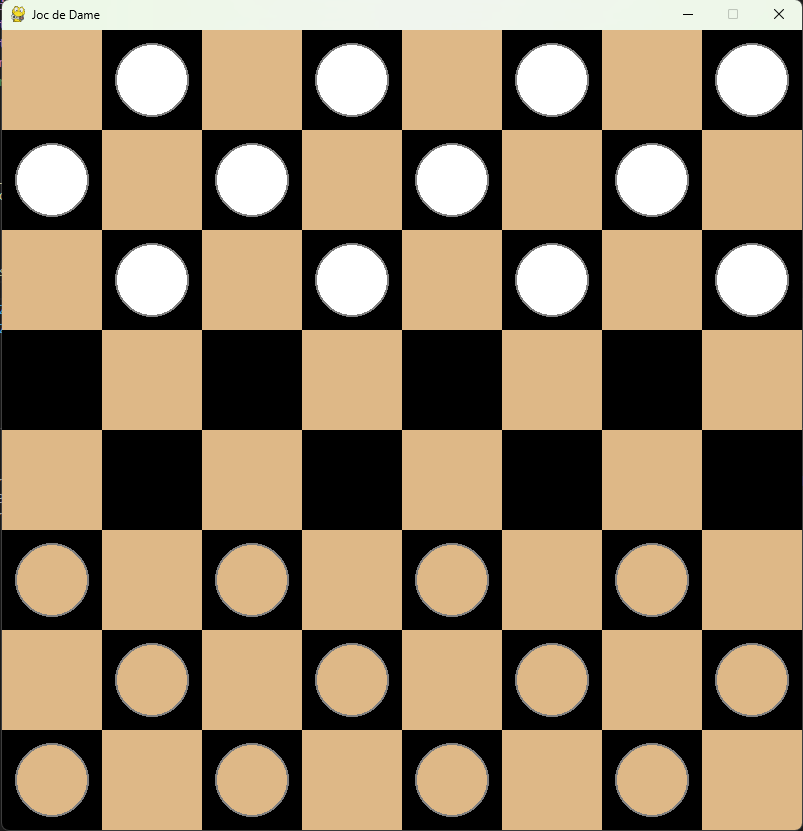


Fig 13. Tabla de joc

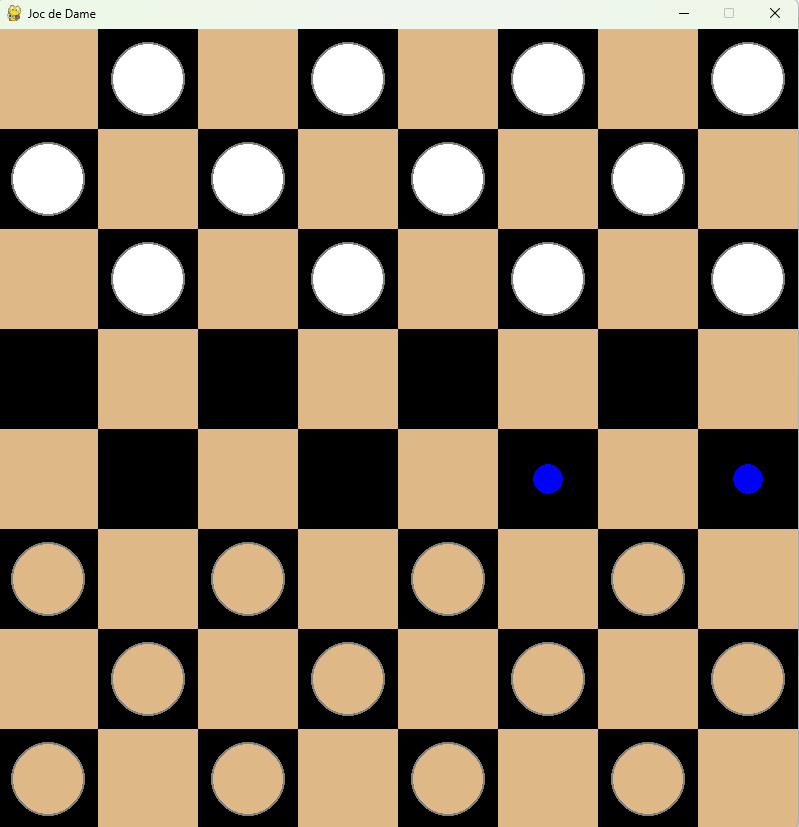
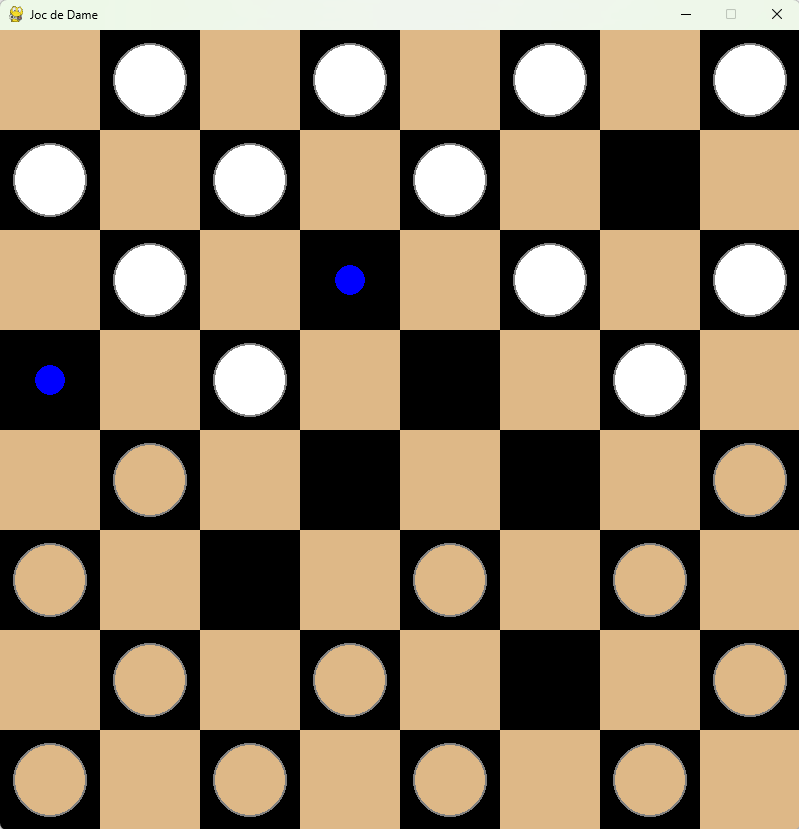
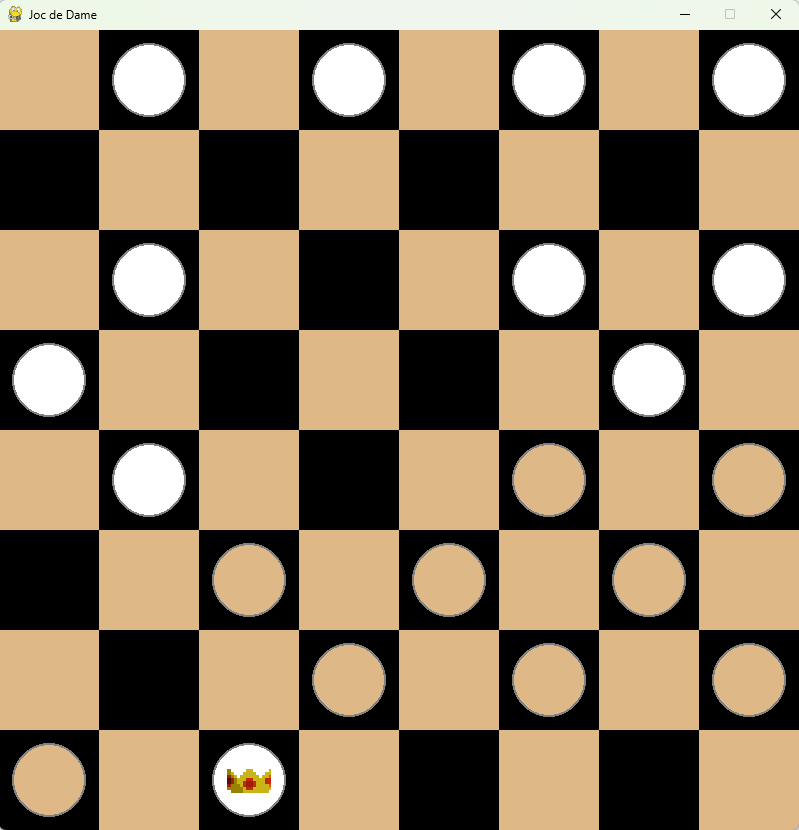
Din momentul în care este apasată tasta ,,play” se pornește jocul în sine,pe ecran avem o masă de joc pregătită

Fig 14. *Mișcarea pieselor*

Pentru a putea mișca pionul e suficient să facem click pe unul din pioni iar mai apoi pe table unde se va afișa modul în care putem să îl mișcăm , în cele din urmă calculatorul va face și el o miscare, timpul de așteptare va fi unul scurt.Jocul continua treptat până când unul din jucatori ramâne fără pioni. Deasemenea în joc sunt prezenți și dame, pentru aceasta e destul să ajungem cu unul dintre pioni la capatul opus al tablei.În general sunt respectate toate regulile unui joc obișnuit de dame.



*Fig 15.Luarea unui pion Fig 16. Damă(pionul care are o coroană pe el)*

# Concluzii

Lucrul în echipă poate fi benefic pentru atingerea obiectivelor comune și pentru îmbunătățirea rezultatelor. Jocul de dame poate fi un bun exemplu pentru a ilustra cum colaborarea, comunicarea eficientă și încrederea în coechipieri pot contribui la succesul în echipă.

În timp ce am creat acest joc de dame am studiat și utilizat diferite algoritmi, putem concluziona că algoritmii sunt un instrument esențial în proiectarea și rezolvarea problemelor în informatică. Algoritmii oferă metode precise și eficiente pentru a rezolva probleme, de la jocuri și aplicații interactive până la procesarea datelor și inteligența artificială. Este important să înțelegem cum funcționează diferitele algoritmi și să știm cum să alegem cel mai potrivit algoritm pentru o anumită problemă.Prin implementarea algoritmul ,,Minimax”, putem concluziona că am înțeles cum să proiectăm un bot inteligent capabil să ia decizii în jocurile cu două opțiuni. Algoritmul Minimax este un instrument puternic pentru evaluarea posibilităților de joc și alegerea celei mai bune opțiuni, luând în considerare opțiunile oponentului. Am reușit să implementăm acest algoritm într-un joc de dame și am observat cum afectează performanța bot-lui. Acest lucru ne-a oferit o înțelegere mai bună a algoritmului și a modului în care poate fi aplicat în alte jocuri sau probleme similar ,acumulând experiență am îmbunătățit abilitățile noastre de a proiecta boți inteligenți și suntem încântați să continuăm să învățăm și să utilizăm noi algoritmi.

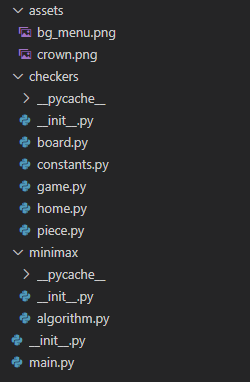
Jocul a fost scris în limbajul de programare Pyton care a dat dovadă acest limbaj de programare este ușor de învățat datorită sintaxei sale simple și intuitive, precum și faptul că este un limbaj interpretat. Acestea toate fac ca Python să fie un limbaj accesibil pentru începători, permițându-le să înțeleagă și să scrie codul mai ușor și mai eficient. În plus, Python este un limbaj versatil, care poate fi folosit pentru o gamă largă de scopuri,după implementarea unui joc de dame în limbajul Python, putem face concluzia că am învățat cum să utilizăm limbajul Python pentru a crea un joc funcțional și interactiv, cum să gestionăm logica și regulile jocului și cum să utilizăm modulele și bibliotecile necesare pentru a crea interfața grafică a jocului folosind biblioteca Pygame, am înțeles cum să combinăm abilitățile de programare cu Pygame pentru a crea jocuri interactive și grafice. Am reușit să utilizăm funcțiile și modulele Pygame pentru a crea elemente grafice precum tabla de joc, piesele, și interfața utilizatorului, precum și pentru a gestiona evenimentele de intrare și a crea un buclu de joc functional,folosind, algoritmul Minimax pentru a oferi agentului inteligență artificială și pentru a face jocul mai interesant. Am reușit să implementăm o varietate de caracteristici precum mutarea pieselor, verificarea câștigătorului. Prin aceasta implementare am îmbunătățit abilitățile noastre de programare și am înțeles cum funcționează jocurile de dame. În viitor, putem continua să optimizăm jocul, adăugând caracteristici suplimentare sau îmbunătățind performanța.

Într-un final rezultatele a fost pe masură obținând o aplicație software,aplicația merge perfect fără bug-uri.

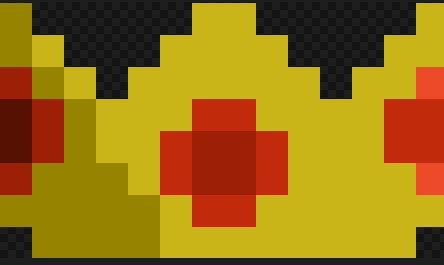
# Bibliografie

1. <https://invata-it.online/software/>
2. <https://www.roportal.ro/articole/despre/cum_se_joaca_dame_reguli_de_joc_1833/>
3. <https://realpython.com/pygame-a-primer/>
4. <https://www.youtube.com/watch?v=RjdrFHEgV2o&t=233s&ab_channel=TechWithTim>
5. <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>
6. <https://www.youtube.com/watch?v=l-hh51ncgDI>
7. <https://else.fcim.utm.md/course/view.php?id=773>

# Codul sursa



Assets



bg\_menu.png crown.png

checkers.py

board.py

import pygame

from .constants import BLACK, ROWS, RED, SQUARE\_SIZE, COLS, WHITE

from .piece import Piece

class Board:

#Initializam valori

    def \_\_init\_\_(self):

        self.board = []

        self.red\_left = self.white\_left = 12

        self.red\_kings = self.white\_kings = 0

        self.create\_board()

#Desenam patratele pe tabla

    def draw\_squares(self, win):

        win.fill(BLACK)

        for row in range(ROWS):

            for col in range(row % 2, COLS, 2):

                pygame.draw.rect(win, RED, (row\*SQUARE\_SIZE, col \*SQUARE\_SIZE, SQUARE\_SIZE, SQUARE\_SIZE))

    def evaluate(self):

        return self.white\_left - self.red\_left + (self.white\_kings \* 0.5 - self.red\_kings \* 0.5)

    def get\_all\_pieces(self, color):

        pieces = []

        for row in self.board:

            for piece in row:

                if piece != 0 and piece.color == color:

                    pieces.append(piece)

        return pieces

    def move(self, piece, row, col):

        self.board[piece.row][piece.col], self.board[row][col] = self.board[row][col], self.board[piece.row][piece.col]

        piece.move(row, col)

        if row == ROWS - 1 or row == 0:

            piece.make\_king()

            if piece.color == WHITE:

                self.white\_kings += 1

            else:

                self.red\_kings += 1

    def get\_piece(self, row, col):

        return self.board[row][col]

    def create\_board(self):

        for row in range(ROWS):

            self.board.append([])

            for col in range(COLS):

                if col % 2 == ((row +  1) % 2):

                    if row < 3:

                        self.board[row].append(Piece(row, col, WHITE))

                    elif row > 4:

                        self.board[row].append(Piece(row, col, RED))

                    else:

                        self.board[row].append(0)

                else:

                    self.board[row].append(0)

    def draw(self, win):

        self.draw\_squares(win)

        for row in range(ROWS):

            for col in range(COLS):

                piece = self.board[row][col]

                if piece != 0:

                    piece.draw(win)

    def remove(self, pieces):

        for piece in pieces:

            self.board[piece.row][piece.col] = 0

            if piece != 0:

                if piece.color == RED:

                    self.red\_left -= 1

                else:

                    self.white\_left -= 1

    def winner(self):

        if self.red\_left <= 0:

            return WHITE

        elif self.white\_left <= 0:

            return RED

        return None

    def get\_valid\_moves(self, piece):

        moves = {}

        left = piece.col - 1

        right = piece.col + 1

        row = piece.row

        if piece.color == RED or piece.king:

            moves.update(self.\_traverse\_left(row -1, max(row-3, -1), -1, piece.color, left))

            moves.update(self.\_traverse\_right(row -1, max(row-3, -1), -1, piece.color, right))

        if piece.color == WHITE or piece.king:

            moves.update(self.\_traverse\_left(row +1, min(row+3, ROWS), 1, piece.color, left))

            moves.update(self.\_traverse\_right(row +1, min(row+3, ROWS), 1, piece.color, right))

        return moves

    def \_traverse\_left(self, start, stop, step, color, left, skipped=[]):

        moves = {}

        last = []

        for r in range(start, stop, step):

            if left < 0:

                break

            current = self.board[r][left]

            if current == 0:

                if skipped and not last:

                    break

                elif skipped:

                    moves[(r, left)] = last + skipped

                else:

                    moves[(r, left)] = last

                if last:

                    if step == -1:

                        row = max(r-3, 0)

                    else:

                        row = min(r+3, ROWS)

                    moves.update(self.\_traverse\_left(r+step, row, step, color, left-1,skipped=last))

                    moves.update(self.\_traverse\_right(r+step, row, step, color, left+1,skipped=last))

                break

            elif current.color == color:

                break

            else:

                last = [current]

            left -= 1

        return moves

    def \_traverse\_right(self, start, stop, step, color, right, skipped=[]):

        moves = {}

        last = []

        for r in range(start, stop, step):

            if right >= COLS:

                break

            current = self.board[r][right]

            if current == 0:

                if skipped and not last:

                    break

                elif skipped:

                    moves[(r,right)] = last + skipped

                else:

                    moves[(r, right)] = last

                if last:

                    if step == -1:

                        row = max(r-3, 0)

                    else:

                        row = min(r+3, ROWS)

                    moves.update(self.\_traverse\_left(r+step, row, step, color, right-1,skipped=last))

                    moves.update(self.\_traverse\_right(r+step, row, step, color, right+1,skipped=last))

                break

            elif current.color == color:

                break

            else:

                last = [current]

            right += 1

        return moves

constants.py

import pygame

WIDTH, HEIGHT = 800, 800

ROWS, COLS = 8, 8

SQUARE\_SIZE = WIDTH//COLS

# rgb

RED = (222,184,135)

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

BLUE = (0, 0, 255)

GREY = (128,128,128)

CROWN = pygame.transform.scale(pygame.image.load('assets/crown.png'), (44, 25))

BACKGROUND =pygame.image.load('assets/bg\_menu.png')

game.py

import pygame

from .constants import RED, WHITE, BLUE, SQUARE\_SIZE

from checkers.board import Board

class Game:

    def \_\_init\_\_(self, win):

        pygame.init()

        self.\_init()

        self.win = win

    def update(self):

        self.board.draw(self.win)

        self.draw\_valid\_moves(self.valid\_moves)

        pygame.display.update()

    def \_init(self):

        self.selected = None

        self.board = Board()

        self.turn = RED

        self.valid\_moves = {}

    def winner(self):

        return self.board.winner()

    def reset(self):

        self.\_init()

    def select(self, row, col):

        if self.selected:

            result = self.\_move(row, col)

            if not result:

                self.selected = None

                self.select(row, col)

        piece = self.board.get\_piece(row, col)

        if piece != 0 and piece.color == self.turn:

            self.selected = piece

            self.valid\_moves = self.board.get\_valid\_moves(piece)

            return True

        return False

    def \_move(self, row, col):

        piece = self.board.get\_piece(row, col)

        if self.selected and piece == 0 and (row, col) in self.valid\_moves:

            self.board.move(self.selected, row, col)

            skipped = self.valid\_moves[(row, col)]

            if skipped:

                self.board.remove(skipped)

            self.change\_turn()

        else:

            return False

        return True

    def draw\_valid\_moves(self, moves):

        for move in moves:

            row, col = move

            pygame.draw.circle(self.win, BLUE, (col \* SQUARE\_SIZE + SQUARE\_SIZE//2, row \* SQUARE\_SIZE + SQUARE\_SIZE//2), 15)

    def change\_turn(self):

        self.valid\_moves = {}

        if self.turn == RED:

            self.turn = WHITE

        else:

            self.turn = RED

    def get\_board(self):

        return self.board

    def ai\_move(self, board):

        self.board = board

        self.change\_turn()

piece.py

from .constants import RED, WHITE, SQUARE\_SIZE, GREY, CROWN

import pygame

class Piece:

    PADDING = 15

    OUTLINE = 2

    def \_\_init\_\_(self, row, col, color):

        self.row = row

        self.col = col

        self.color = color

        self.king = False

        self.x = 0

        self.y = 0

        self.calc\_pos()

    def calc\_pos(self):

        self.x = SQUARE\_SIZE \* self.col + SQUARE\_SIZE // 2

        self.y = SQUARE\_SIZE \* self.row + SQUARE\_SIZE // 2

    def make\_king(self):

        self.king = True

    def draw(self, win):

        radius = SQUARE\_SIZE//2 - self.PADDING

        pygame.draw.circle(win, GREY, (self.x, self.y), radius + self.OUTLINE)

        pygame.draw.circle(win, self.color, (self.x, self.y), radius)

        if self.king:

            win.blit(CROWN, (self.x - CROWN.get\_width()//2, self.y - CROWN.get\_height()//2))

    def move(self, row, col):

        self.row = row

        self.col = col

        self.calc\_pos()

    def \_\_repr\_\_(self):

        return str(self.color)

home.py

import pygame

from .constants import BACKGROUND

#from .chekers

class Home:

    def \_\_init\_\_(self, win):

        pygame.init()

        self.win = win

    def menu(self):

        """Displays the menu and handles menu events"""

        bg\_image = BACKGROUND

        play\_button = pygame.Rect(300, 250, 200, 50)

        #settings\_button = pygame.Rect(300, 325, 200, 50)

        quit\_button = pygame.Rect(300, 400, 200, 50)

        while True:

            self.win.blit(bg\_image, (-170, 0))

            pygame.draw.rect(self.win, (255, 255, 255), play\_button)

            #pygame.draw.rect(self.win, (255, 255, 255), settings\_button)

            pygame.draw.rect(self.win, (255, 255, 255), quit\_button)

            font = pygame.font.Font(None, 30)

            play\_text = font.render("Play", True, (0, 0, 0))

            #settings\_text = font.render("Settings", True, (0, 0, 0))

            quit\_text = font.render("Quit", True, (0, 0, 0))

            self.win.blit(play\_text, (380, 265))

            #self.win.blit(settings\_text, (365, 335))

            self.win.blit(quit\_text, (380, 415))

            pygame.display.update()

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    pygame.quit()

                    # sys.exit()

                elif event.type == pygame.MOUSEBUTTONDOWN:

                    if play\_button.collidepoint(event.pos):

                        return "game"

                    #elif settings\_button.collidepoint(event.pos):

                        #return "settings"

                    elif quit\_button.collidepoint(event.pos):

                        pygame.quit()

                        # sys.exit()

Minimax

algorithm.py

from copy import deepcopy

import pygame

RED = (222,184,135)

WHITE = (255, 255, 255)

def minimax(position, depth, max\_player, game):

    if depth == 0 or position.winner() != None:

        return position.evaluate(), position

    if max\_player:

        maxEval = float('-inf')

        best\_move = None

        for move in get\_all\_moves(position, WHITE, game):

            evaluation = minimax(move, depth-1, False, game)[0]

            maxEval = max(maxEval, evaluation)

            if maxEval == evaluation:

                best\_move = move

        return maxEval, best\_move

    else:

        minEval = float('inf')

        best\_move = None

        for move in get\_all\_moves(position, RED, game):

            evaluation = minimax(move, depth-1, True, game)[0]

            minEval = min(minEval, evaluation)

            if minEval == evaluation:

                best\_move = move

        return minEval, best\_move

def simulate\_move(piece, move, board, game, skip):

    board.move(piece, move[0], move[1])

    if skip:

        board.remove(skip)

    return board

def get\_all\_moves(board, color, game):

    moves = []

    for piece in board.get\_all\_pieces(color):

        valid\_moves = board.get\_valid\_moves(piece)

        for move, skip in valid\_moves.items():

            #draw\_moves(game, board, piece)

            temp\_board = deepcopy(board)

            temp\_piece = temp\_board.get\_piece(piece.row, piece.col)

            new\_board = simulate\_move(temp\_piece, move, temp\_board, game, skip)

            moves.append(new\_board)

    return moves

#def draw\_moves(game, board, piece):

    valid\_moves = board.get\_valid\_moves(piece)

    board.draw(game.win)

    pygame.draw.circle(game.win, (0,255,0), (piece.x, piece.y), 50, 5)

    game.draw\_valid\_moves(valid\_moves.keys())

    pygame.display.update()

    pygame.time.delay(2)

main.py

import pygame

from checkers.constants import WIDTH, HEIGHT, SQUARE\_SIZE, RED , WHITE

from checkers.game import Game

from checkers.home import Home

from minimax.algorithm import minimax

#from checkers.home import init

FPS = 60

WIN = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption('Joc de Dame')

def get\_row\_col\_from\_mouse(pos):

    x, y = pos

    row = y // SQUARE\_SIZE

    col = x // SQUARE\_SIZE

    return row, col

def main():

    state = "menu"

    game = Game(WIN)

    home = Home(WIN)

    while state != "quit":

        if state == "menu":

            state = home.menu()

        elif state == "game":

            run = True

            clock = pygame.time.Clock()

            while run:

                clock.tick(FPS)

                if game.turn == WHITE:

                    value, new\_board = minimax(game.get\_board(), 4, WHITE, game)

                    game.ai\_move(new\_board)

                if game.winner() != None:

                    print(game.winner())

                    run = False

                for event in pygame.event.get():

                    if event.type == pygame.QUIT:

                        run = False

                        pygame.quit()

                    if event.type == pygame.MOUSEBUTTONDOWN:

                        pos = pygame.mouse.get\_pos()

                        row, col = get\_row\_col\_from\_mouse(pos)

                        game.select(row, col)

                game.update()

            pygame.quit()

main()