# Implementacja algorytmu min-max w Warcabach

Przygotował: Karol Profic 228836

# Opis projektu:

Celem projektu było dodanie algorytmu do gry w warcaby dla wcześniej przygotowanej aplikacji. W tym celu został zaimplementowany algorytm min-max. Poziom trudności będzie określany poprzez ilość kroków które algorytm ma analizować do przodu. Łatwy poziom trudności to jeden krok, natomiast trudny to cztery kroki do przodu.

# Zasady gry:

Gra warcaby klasyczne (określana też warcabami brazylijskimi) rozgrywana jest na planszy o rozmiarze 8 × 8 pól pokolorowanych na przemian na kolor jasny i ciemny.

Każdy gracz rozpoczyna grę z dwunastoma pionami (jeden koloru białego, drugi – czerwonego) ustawionymi na ciemniejszych polach planszy, po których się poruszają. Ustawione są w sposób przedstawiony na ilustracji.

Jako pierwszy ruch wykonuje grający pionami białymi, po czym gracze wykonują na zmianę kolejne ruchy.

Celem gry jest zbicie wszystkich pionów przeciwnika (w tym damek) albo zablokowanie wszystkich, które pozostają na planszy, pozbawiając przeciwnika możliwości wykonania ruchu. Jeśli żaden z graczy nie jest w stanie tego osiągnąć (każdy z graczy wykona po 15 ruchów damkami bez zmniejszania liczby pionów pozostających na planszy), następuje remis.

Piony mogą poruszać się o jedno pole do przodu po przekątnej na wolne pola.

Bicie pionem następuje przez przeskoczenie sąsiedniego pionu (lub damki) przeciwnika na pole znajdujące się tuż za nim po przekątnej (pole to musi być wolne). Zbite piony są usuwane z planszy po zakończeniu ruchu.

Piony mogą bić zarówno do przodu, jak i do tyłu.

W jednym ruchu wolno wykonać więcej niż jedno bicie tym samym pionem, przeskakując przez kolejne piony lub damki przeciwnika.

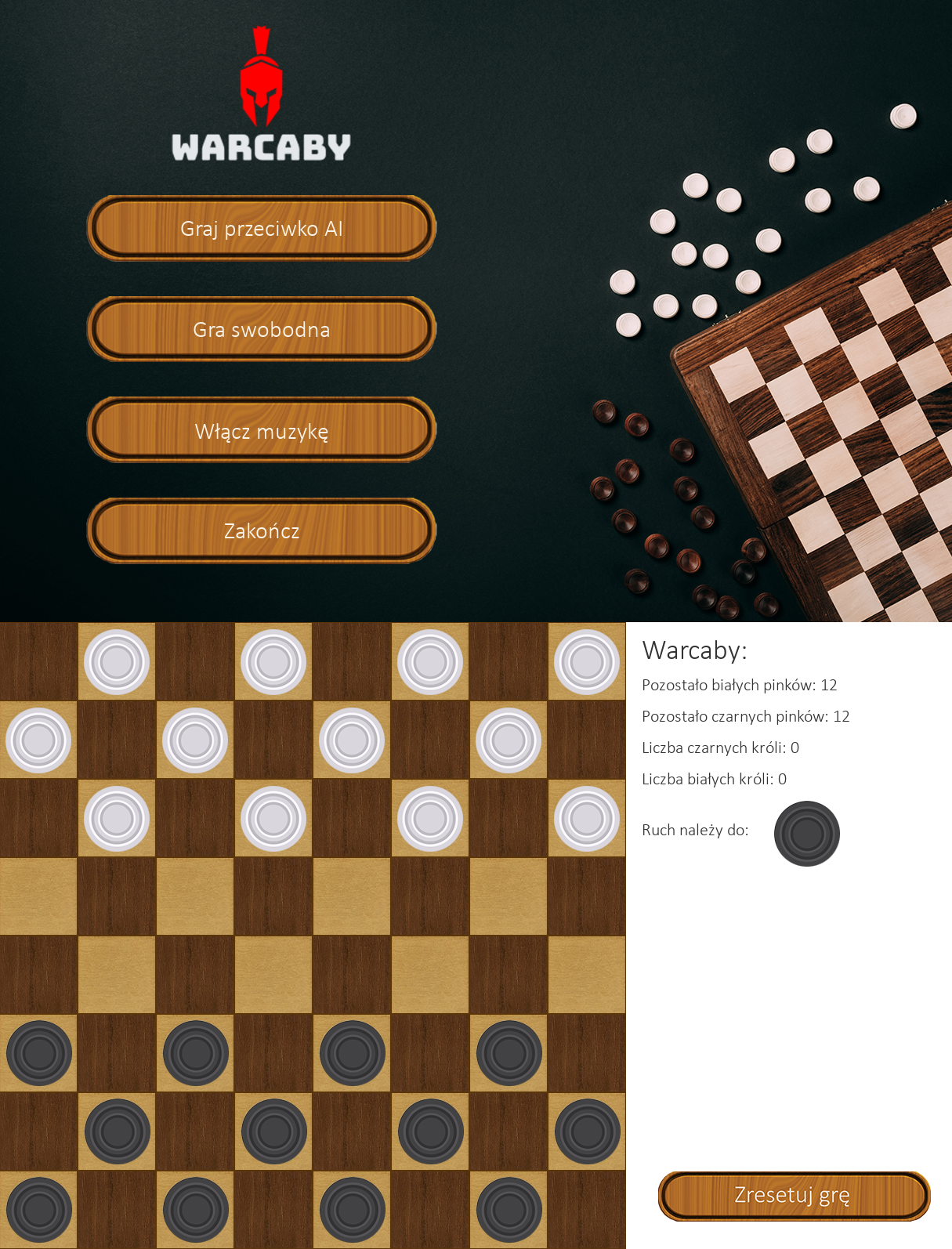
Bicia są obowiązkowe. Kiedy istnieje kilka możliwych bić, gracz musi wykonać wszystkie. Podczas bicia nie można przeskakiwać więcej niż jeden raz przez ten sam pion lub damkę[3].

Pion, który dojdzie do ostatniego rzędu planszy, staje się damką, przy czym jeśli w jednym ruchu w wyniku wielokrotnego bicia przejdzie przez ostatni rząd, ale nie zakończy na nim ruchu, to nie staje się damką i kończy ruch jako pionek.

Damki mogą poruszać się w jednym ruchu o dowolną liczbę pól do przodu lub do tyłu po przekątnej, zatrzymując się na wolnych polach.

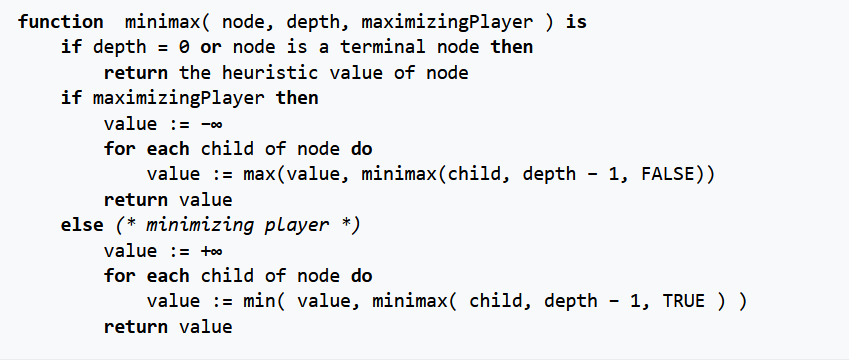
Bicie damką jest możliwe z dowolnej odległości po linii przekątnej i następuje przez przeskoczenie pionu (lub damki) przeciwnika, za którym musi znajdować się co najmniej jedno wolne pole – damka przeskakuje na dowolne z tych pól i może kontynuować bicie na tej samej lub prostopadłej linii.

# Wygląd GUI:



# Opis działania algorytmu:

Algorytm działa w sposób rekurencyjny, jeżeli chcemy znaleźć najlepszy ruch dla czarnego pionka na trudnym poziomie trudności, to wywołujemy metodę minmax z parametrami: obecny stan planszy, kolor pionka(czarny), ilość ruchów wprzód(4). Następnie wywołujemy funkcję znalezienia najlepszego ruchu w tym przypadku będzie to min, ponieważ wraz z zmniejszaniem się liczby białych pionków wartość oceny planszy będzie maleć, natomiast gdy wygrywać będą białe wartość planszy będzie rosnąć. Następnie znajdowany jest najlepszy ruch dla białych i jest to kontynuowane aż do osiągnięcia czwartego ruchu wprzód, następnie zwracane wyniki są porównywane i z każdego zagnieżdżenia zwracany jest najlepszy.

Przykładowa implementacja w pseudokodzie: 

# Analiza wyników:

Poziom trudności Łatwy – na tym poziomie AI jest w stanie analizować tylko jeden ruch do przodu, dlatego gdy tylko pojawia się możliwość bicia algorytm decyduje że jest to najlepsze wyjście ponieważ zmieni to funkcję oceny na korzyść obecnego gracza.

Poziom trudności Trudny – na tym poziomie AI rozgrywa wszystkie ruchy na cztery ruchu w przód, bardzo często zdarza się że algorytm nie bije od razu, zamiast tego woli poczekać na wielokrotne bici.

# Struktura projektu:

/assets/board/\* - pliki graficzne dla pojedynczych bloków planszy

/assets/menu/\* - pliki graficzne dla przycisków i tła

/assets/music/\* - pliki dźwiękowe

/assets/piece/\* - pliki graficzne dla pionków

/checkers/AI.py – Klasa odpowiadająca za znalezienie ruchu podczas gry z komputerem

Parametry:

board – kopia obecnego stanu planszy

depth – ile ruchu do przodu ma zostać sprawdzona

playerColor – kolor gracza dla której szukamy najlepszego ruchu

Metody:

minima – Znajduje najlepszy ruch

rateMove – Ocenia wartość ruchu, zlicza ilość pionków białych, czarnych ilość królów obu kolorów i w zależności od tego kto będzie miał więcej po wykonaniu ruchu przypisuje mu wartość

findWhiteMoves – Znajduje najlepszy ruch dla białych

findBlackMoves – Znajduje najlepszy ruch dla czarnych

getAllMoves – Zwraca wszystkie możliwe ruchy do wykonania dla konkretnego koloru

findAllPiecesOfColor – Zwraca wszystkie pionki danego koloru

/checkers/Board.py – Klasa która przechowuje stan planszy i pozwala na jego modyfikowanie

Parametry:

boardArray – Tablica z pionkami

numberOfRemainingBlack – Liczba czarnych pionków

numberOfRemainingWhite – Liczba białych pionków

numberOfBlackKing – Liczba czarnych królów

numberOfWhiteKing – Liczba białych królów

Metody:

createArrayBoard – Dodaje pionki do tablicy boardArray

drawBoard – Rysuje planszę na scenie

drawPiecesOnBoard – Rysuje pionki na scenie

movePiece – Zmienia położenie pionka

reset – Resetuje planszę

removePiece – Usuwa pionka z planszy

returnWinnerIfExists – Zwraca zwycięzcę jeżeli gra się zakończyła

findPossibleMoves – Zwraca możliwe do wykonania ruchy

getPiece – Getter dla pionka

/checkers/Controller.py – Kontroluje rozgrywkę, posiada informację o tym czyj ruch, planszę, scene

Parametry:

gameWindow - scena

selectedPiece – wybrany przez gracza pionek

board - plansza

whoseTurn – czyja kolej

possibleMoves – możliwe ruchu, w celu narysowania podpowiedzi

Metody:

update – Rysuje planszę

reset – Resetuje grę

returnWinnerIfExists – Zwraca zwycięzcę jeżeli gra się zakończyła

getBoard – Getter dla planszy

setBoard – Setter dla planszy

selectPiece – Wybieranie pionka, rysowanie podpowiedzi

movePiece – Wykonanie ruchu pionkiem

drawPossibleMoves – Rysowanie podpowiedzi

changePlayer – Zmiana gracza na przeciwnego

/checkers/Piece.py

Parametry:

x, y – odległość od lewego górnego rogu ekranu

row, column – rząd i kolumna pionka

color – kolor pionka

isKing – czy pionek jest królem

Metody:

calculateCoordinates – ustala położenie x,y pionka

makeKing – zmienia pionka na króla

draw – rysuje pionka na planszy

move – zmienia położenie pionka

/checkers/constants.py – Stałe wykorzystywane w projekcie, głównie zmienne tekstowe a także kolory i ścieżki dostępu

/commons/functions.py – Zawiera funkcje rysującą plik graficzny na ekranie

/layout/Button.py – Klasa dla przycisku

Parametry:

x, y – odległość od lewego górnego rogu ekranu

text – tekst który zostanie wypisany na przycisku

screen – scena na której zostanie narysowany przycisk

font – czcionka dla tekstu na przycisku

Metody:

drawButton – rysuje przycisk na planszy

/scenes/Game.py – Klasa która posiada obsługę menu i kliknięć jest rozszerzana w PlayerVersusAI i PlayerVersusPlayer

/scenes/Menu.py – Obiekt rysujący menu i wywołuje PlayerVersusAI lub PlayerVersusPlayer w zależności od decyzji użytkownika

/scenes/PlayerVersusAI.py – Scena z trybem gry gracz kontra komputer, główna pętla programu

/scenes/PlayerVersusPlayer.py – Scena z trybem gry gracz kontra gracz, główna pętla programu

/main.py – Tworzy obiekt Menu