# ChessPY

# Readme

Versie 1.1

# Inhoudsopgave

# Inleiding

Toen ik bedacht dat ik dit project wilde doen was ik ongeveer drie maanden bezig met programmeren in python. Ik houd erg van schaken en het mooiste van het spelletje is dat het eigenlijk alleen maar draait om logica. Er is geen geluk bij schaken. En daarom vond ik het bij uitstek geschikt om dit als project in python. Het idee was om de regels die de bekende schaaksites gebruiken om hun spel aan te sturen, ook toe te passen op mijn schaakspel. Dit is veruit het meest complexe wat ik tot nu toe gedaan op het gebied van programmeren. Ik wil graag met deze readme een uitleg geven hoe ik het geprogrammeerd heb en daarmee een inkijkje te geven in mijn gedachtenproces tijdens het project.

## Voorbereiding

Ik ben begonnen met een plan van aanpak met een planning en een globale structuur van hoe ik het programma eruit wilde laten zien. In eerste instantie wilde ik animaties uitvoeren met JavaScript, maar dit was niet zo makkelijk te combineren met python. Daarom heb ik later ervoor gekozen om alles alleen in python te doen.

Ik heb afbeeldingen gezocht op het internet van schaakstukken en heb deze bewerkt op pixlr.com om de juiste grootte, kleur, vorm etc. te krijgen die ik voor ogen had om het spel te maken

Voor het spelen van het spel en het main menu is er gebruik gemaakt van de pygame library.

## UI Classes

Bestand: “UIClassesV3.py”

Pygame heeft geen standaard library voor UI design. Ik heb nog kortstondig de PygameGUI library proberen te gebruiken, maar deze had niet de functionaliteit die ik nodig had voor mijn UI. Dus heb ik mijn eigen UI Classes gemaakt in het bestand: UIClassesV3. Pygame werkt met rectangles en deze rectangles dien je op het scherm te blitten. Door events toe te kennen aan deze rectangles zoals MOUSEDOWN of MOUSEUP kan je bijvoorbeeld een button maken.
Dit bestand bevat classes voor: Tekst, Image, Button en Entry.
Per class kunnen verschillende dingen worden aangepast. Bijvoorbeeld een button kan een image bevatten en tekst kan een underline krijgen.

## Main menu

Bestand: “Main_menu2.py”

Het main menu bestaat uit vier verschillende stukken waarin keuzes gemaakt kunnen worden. Het eerste stuk (links boven) kan de spel modus worden gekozen. Het tweede blok (links onder) kan de kleur worden gekozen. Het derde blok (rechts boven) kan de tijd per speler worden gekozen en het vierde blok bevat de start knop. Als een van deze keuzes niet zijn gemaakt zal er een waarschuwing verschijnen met de keuze die nog niet gemaakt is. Verder hebben de buttons een active status. Bij opstarten is de active status “OFF”. Wanneer er op een button gedrukt wordt met de muis is de active status “ON” De knoppen zijn zo geprogrammeerd dat er nooit twee knoppen in hetzelfde blok active status “ON” kunnen hebben. In het tijd blok waar drie buttons en vier entries te zien zijn, krijgen de buttons voorrang. Dus wanneer je zelf een tijd wil invoeren moeten de button active status alle drie “OFF” zijn. De keuzes die worden gemaakt in het menu worden opgeslagen in een .json bestand. Dit bestand wordt in een andere python file weer geladen.

## De schaakregels

Bestanden: “PiecesV10.py”, “Pieces_PvP.py”

Het verschil in bovenstaande bestanden is dat “Pieces_PvP.py” wordt opgestart bij een spel tegen de speler en “PiecesV10.py” wordt opgestart bij een spel tegen de computer.

Bij “Pieces_PvP.py” wordt het bord en de stukken steeds omgedraaid als de speler een zet heeft gedaan, zodat de stukken van de speler die aan de beurt is altijd onderaan staan. Bij PiecesV10 blijft de kleur die gekozen is in het main menu altijd onderaan staan. Verder bevat PiecesV10 een paar definities die ervoor zorgen dat de computer het board kan interpreteren en een zet kan doen. De basisregels zijn verder hetzelfde. Deze worden hieronder beschreven (hoe ze zijn geprogrammeerd). Als er een verschil tussen de bestanden is zal dit worden aangegeven.

==== Tekenen van bord ====

def draw_bord():

Een tweedimensionale array die elke keer om en om van iedere kleur een rectangle op het scherm tekent. Ook de letters en cijfers van de 1$^{e}$ rij en de 1$^{e}$ kolom worden in het bord gezet.

==== Aanroepen van schaakstukken ====

def __init__ ():
def init_state_and_reset():

In de def init_state_and_reset wordt de Class Piece zelf aangeroepen. Evenals een aantal booleans bijvoorbeeld: white_checkmate, black_checkmate, stale_mate etc. Deze worden een waarde gegeven (True of False). Er wordt ook een dictionary gemaakt namelijk: square_list. Dit is één van de vele dictionaries de zal worden gemaakt, echter is dit de hoofddictionary en zal tijdens het hele spel worden gebruikt. In eerste instantie worden hier de x- en y- values met de corresponderende naam van het veld toegevoegd waarbij de veldnaam als key wordt gebruikt en de de x- en y-waarden in een list als value van deze key.

De Class Piece erft over van de Sprite Class van Pygame. De schaakstukken worden aangeroepen als sprites. Deze sprites worden op het scherm getekend op x- en y-waarden die worden gehaald uit de square_list dictionary. Er wordt “OCCUPIED”, de kleur van het

stuk en de naam van het stuk toegevoegd aan corresponderende square in de square_list dictionary.

==== Stukken bewegen ====

def image_move():

De stukken (sprites) hebben een active(init=False) en een placed(init=True) status. Wanneer de er met de linkermuisknop op een stuk wordt gedrukt, veranderd de active status in True en de placed boolean in False. Zolang de muisknop ingedrukt blijft kan het stuk worden gesleept.

==== Stukken verplaatsen ====

def place_pieces():

def ... puntjes staan voor wit/zwart

Eerst wordt bepaald dat wanneer er zich een sprite buiten de kaders van het bord bevindt, deze terug wordt geplaatst naar de originele positie. Deze wordt gehaald uit de square_list dictionary.

Dan worden voor alle stukken de mogelijke stappen op het bord die dit stuk kan zetten gekaderd.

Vervolgens wordt voor het stuk dat gesleept wordt de afstand bepaald tussen de originele positie van het stuk en de positie waar deze heen gesleept wordt. Wanneer deze binnen de kaders vallen van het betreffende stuk worden de nieuwe file en de nieuwe row berekend. Deze nieuwe file en row worden in een string gecombineerd tot het nieuwe vlak waar het gesleepte stuk zou kunnen staan, maar daarvoor moet er eerst nog een aantal checks worden gedaan.

Alleen het paard kan over ander stukken springen, daarom wordt deze overgeslagen in deze check. Als het stuk geen paard is, wordt er een lijstje gemaakt van de squares tussen de originele square en de nieuwe square. (Bijvoorbeeld, loper gaat van c1 naar g5, dan wordt er een lijstje gemaakt als volgt: [d2, e3, f4]). Er wordt via de square_list dictionary gecheckt of deze velden bezet zijn. (Als één van de velden bezet is kan een stuk er niet overheen springen)

Daarna komt er een aantal vergelijkende condities. Deze doen de volgende dingen:

- White moves
- Black moves
- White takes
- Black takes
- White takes en passant
- Black takes en passant

Dit zijn de zes uitkomsten wanneer een stuk verplaatst wordt.
De laatste conditie (else) vangt af wanneer niet aan één van deze zes condities wordt voldaan. Het stuk wordt teruggeplaatst naar de originele positie en als nodig wordt de square_list ook weer hersteld.

Wanneer er wel wordt voldaan aan de voorwaarden van één van de blokken en het stuk wordt dus verplaatst, zijn er een aantal uitkomsten die deze zet teweeg kunnen brengen. Er wordt gekeken of één van de beide koningen schaak staat (def check_check_...). Wanneer deze schaak staat wordt in een andere definitie gekeken of de koning schaakmat (def check_checkmate_…) staat. Wanneer de eigen koning schaak komt te staan door de eigen zet, gaat hij naar de def place_back_...() van de corresponderen de kleur.

Wanneer de eigen koning niet schaak staat dan wordt de zet gedaan, maar er zijn dan nog wel wat checks die moeten worden gedaan om uitkomsten van het spel te bevestigen of te ontkrachten. De def fifty_move_rule() checkt de fifty move rule, de def threefold_repetition() checkt op een remise door de herhaling van zetten en de def draw_by_material() checkt op remise door materiaal.

Wanneer de koning van de andere speler niet in schaak staat nadat jij een zet hebt gedaan, wordt de def stale_mate_…() uitgevoerd. Deze checkt op remise door pat.

### PiecesV10.py:

Wanneer de zet wordt gedaan en de eigen koning staat niet in schaak wordt er de def FEN_notation() aangeroepen. Deze vertaald de huidige positie in een string die gelezen kan worden door de Stockfish library (chess engine). Daarna wordt de def stockfish_play() aangeroepen om de engine een zet te laten doen. Deze definities zitten niet de Pieces_PvP code.

### Pieces_PvP.py:

Het verschil met PiecesV10 is het plaatsen van de stukken op het bord nadat de zet is gedaan. In Pieces_PvP is er de def place_pieces_on_board(). In dit stukje code worden 2 dictionaries gemaakt met weer de veldnaam als key en de x- en y-waarden als value. De ene dictionary is eigenlijk een inverse van de andere. Deze worden afhankelijk van de beurt aangeroepen en de sprites worden op het bord geplaatst. Zo wordt ervoor gezorgd dat de speler die aan de beurt is altijd onder staat. Het bord draait als het ware om.

Verder is het bewegen en verplaatsen van de sprites iets anders dan de dictionary, deze dictionary wordt constant bijgehouden dat wanneer er iets verplaatst wordt, dit ook gebeurt in de square_list dictionary door values te verwijderen of te appenden.

Van alle bovenstaande definities worden nog verdere uitleg gegeven in volgende paragrafen.

==== Pion promotie ===

def pawn_promotion()

Het tekenen op het scherm gebeurt in “Engine_gameV2.py” en “PvP_gameV2.py”. Ook deze definitie wordt door deze bestanden aangeroepen en niet binnen het bestand “PiecesV10.py”of “Pieces_PvP.py” Hierover later een uitgebreidere uitleg.

Wanneer een pion de overkant van het bord bereikt (Voor wit 8$^{e}$ rij, voor zwart 1$^{e}$ rij) mag een pion de promotie maken. Dit wordt gedaan door een rectangle relatief aan de x- en y- waarde van de te promoten pion op het scherm te tekenen met daarin de vier verschillende stukken als buttons te plaatsen. Wanneer er op een van deze buttons wordt geklikt wordt de sprite van de pion verwijdert, en wordt de sprite van het gekozen stuk toegevoegd aan het bord. De sprite wordt aangemaakt door de Class Piece zelf aan te roepen. Door de Class aan te roepen worden de waardes via de def __init__() in de square_list opgenomen.
Wanneer het stuk geplaatst is wordt er gecheckt of de koning schaak of schaakmat staat.
Ook wordt er de def FEN_notation() aangeroepen om Stockfish een update te geven over de veranderingen op het bord. Als laatste wordt de def stockfish_play() aangeroepen. Normaal gesproken gebeuren deze dingen vanuit de def place_pieces() worden aangeroepen, maar de def pawn_promotion niet via “PiecesV10.py”of “Pieces_PvP.py” gaat zal dit niet automatisch gebeuren. Dit is een beetje een ontwerpfout, maar toen ik hierachter kwam dat het niet echt handig was, was ik alweer verder met de code. Het werkt wel zo.

==== Checken of een koning schaak staat ====

def check_check_white()

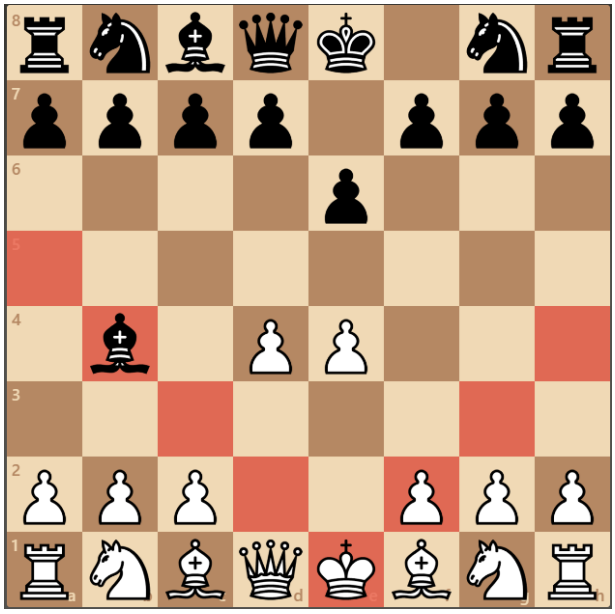
def check_check_black()

Beide definities zijn nagenoeg hetzelfde. def check_check_white() kijkt of de witte koning schaak staat. En def check_check_black(). Er wordt aan het begin van de een list() aangemaakt. In deze list komen de waardes “True” of “False”.  Als er één item in de list “True” is dan staat de koning schaak. Omdat er soms bij één actie in het spel voor beide koningen gecheckt moet worden of ze schaak staan vond ik het makkelijker en

overzichtelijker om hier twee aparte definities van te maken.

De huidige square van de koning wordt aangeroepen. Daarmee is rekening gehouden als de koning wel of niet verplaatst wordt. (Als de koning verplaatst wordt is het new_square). Vervolgens wordt er een list (between_check_square_list) gemaakt met daarin de namen van alle velden. Deze gebruiken we om te vergelijken met de square_list(hoofddictionary).

Daarna wordt vanaf de square van de betreffende koning een lijst gemaakt met de velden van de diagonalen vanaf de koning.



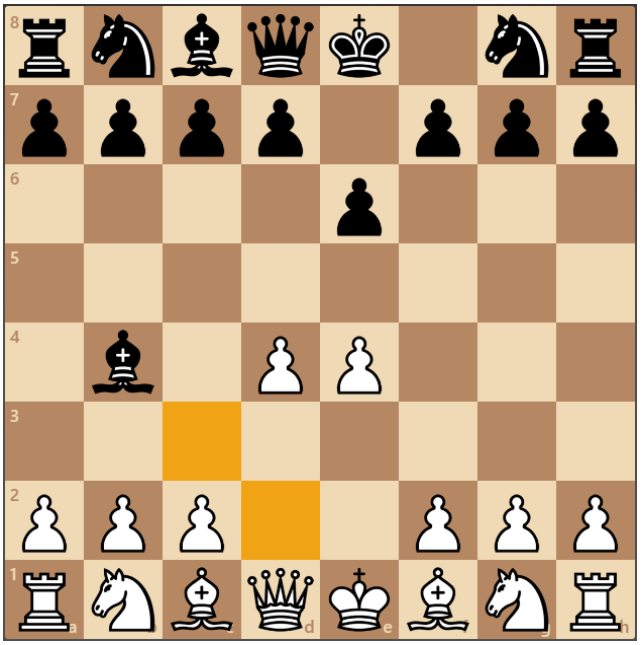
*Figuur 1 lijst met alle velden van de diagonalen*

Daarna wordt er gekeken of er in de lijst met velden een van de volgende stukken staat:

- Loper
- Koning
- Pion
- Dame

Dit wordt gedaan door de diagonalen-lijst te vergelijken me de square_list. Wanneer er een stuk gevonden wordt op een van deze squares, wordt aan de hand van de between_check_square_list de index van het veld genomen waarop dit stuk staat. Het is makkelijker om met de indexen te werken omdat deze over de diagonalen elke keer met dezelfde hoeveelheid toenemen. Van linksonder naar rechtsboven is het verschil telkens negen, en van rechtsonder naar linksboven is het verschil steeds 7.

Als we de index van de square van een van de stukken hebben en de index van de square van de koning, maken we een lijst met alle velden tussen deze squares.



*Figuur 2 Deze velden komen in de lijst tussen de koning en de loper*

Deze lijst ondergaat de volgende vergelijkende condities:

- Als er in deze lijst enig veld is bezet met een stuk die dezelfde kleur heeft dan de koning die gecheckt wordt, staat het niet schaak. De schaak wordt geblokkeerd. (“False” wordt toegevoegd aan check_list)
- Als er in deze lijst enig veld is bezet met het stuk die de andere kleur heeft dan de koning die gecheckt wordt, en dit stuk is een pion, koning paard of toren, staat de koning niet schaak. (“False” wordt toegevoegd aan check_list)
- Wanneer het bovenstaand niet waar is dan staat de koning schaak. (“True wordt toegevoegd aan de check_list)

Wanneer de diagonalen check is gedaan, doen we hetzelfde voor de verticale en horizontale lijnen. Echter nu zijn de stukken die gecheckt worden op de rechte lijnen:

- Toren
- Dame
- Koning

De koning nemen we overigens overal in mee, de koning mag nooit naast de koning komen te staan (of de koning slaan). Als dit voor beide koningen geldt dan kan dit niet gebeuren.

Een stuk wordt namelijk teruggezet wanneer iemand zichzelf in schaak zet.
(def place_back_...)

De laatste check die gedaan wordt om te kijken of de koning in schaak staat is door een lijst te maken van de squares rondom de koning van waaruit hij mogelijk schaak kan staan door een paard. Een paard een koning niet schaak zetten door een diagonaal, horizontaal of een verticale lijn, daarom is hier een aparte lijst voor nodig. Voor de rest worden dezelfde dingen uitgevoerd als hierboven, echter de check of er tussen de koning en een paard andere stukken staan is nu niet relevant omdat een paard over stukken kan bewegen en daarom ook de koning schaak kan zetten zonder dat dit afhankelijk is van de stukken tussen de koning en dit paard.

==== Stukken terugplaatsen ====

def place_back_white()

def place_back_black()

Wanneer door een eigen zet de koning in schaak komt te staan, wordt het stuk teruggezet naar de originele plaats. De definities hebben de parameters new_square en old_square. Daardoor kunnen gemakkelijk de x- en y- waarden van de sprite teruggehaald worden van de originele positie met square_list.

Ook zorg deze definitie ervoor dat de square_list hersteld wordt in de positie vóór deze zet.

==== Checken voor checkmate ====

def check_checkmate_black

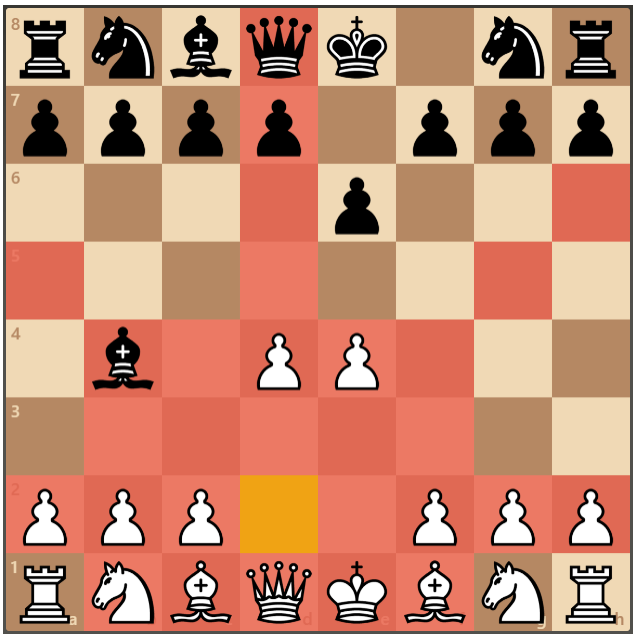
def check_checkmate_white

Om dezelfde redenen dat het checken voor schaak voor beide kleuren gescheiden wordt gehouden, wordt ook het checken van schaakmat gescheiden gehouden.

Het eerste stukje van de code heeft bijna dezelfde structuur als de check_check definites van zwart en wit. In de check_check de lijsten van de diagonalen etc. worden gemaakt vanaf de square van de koning. Hier wordt er dezelfde handelingen uitgevoerd voor alle mogelijke velden waar de koning heen zou kunnen verplaatsen en dan wordt gekeken of hij mogelijk in schaak staat als de koning hierheen zou verplaatsen. Daarmee is rekening gehouden dat de koning niet buiten het bord kan staan, en wanneer een veld bezet is door een stuk van de eigen kleur, wordt deze overgeslagen omdat de koning hier nooit heen zal kunnen bewegen. Als er een stuk van de andere kleur op één van de velden staat wordt deze wel meegenomen

in de check omdat dit stuk mogelijk geslagen kan worden door de koning. Wanneer er geen schaak is in een van de velden wordt ‘“K “+ deze square’ toegevoegd aan de legal_moves_list.

De koning verplaatsen is één van de manieren om uit schaak te komen, maar er zijn nog meer manieren. De schaak zou ook geblokkeerd kunnen worden door een eigen stuk. Dit is gedaan door de lijsten van figuur 2 uit def check_check_... op te halen.  Nu wordt voor elke square tussen de koning en het stuk én de square van stuk zelf, lijsten gemaakt van de diagonalen, horizontaal, verticaal en de mogelijke velden voor het paard.



*Figuur 3 Alle velden die gecheckt worden voor de square d2*

Alle rood getekende velden worden gecheckt voor mogelijke stukken die op d2 zouden kunnen staan, alle regels in acht houdende. Volgens de regels zou het paard op b1 naar d2 kunnen en hier zie je ook dat het veld b1 ook rood is gekleurd.

Er wordt echter nog 1 veld toegevoegd die gecheckt moet worden (aan de lijst van velden tussen het schaakstuk en de koning) en dat is het veld van het stuk wat de koning schaak zet. Door dit veld mee te nemen, check je of het stuk geslagen kan worden waardoor de koning dus ook niet meer schaak staat.

Wanneer er een mogelijke zet wordt gevonden wordt deze eerst in de square_list gezet (dit is dus nog niet visueel met de sprites) waarna dan wordt gecheckt of de koning door deze zet nogsteeds schaak staat. Wanneer dit niet het geval wordt de zet toegevoegd aan legal_moves_list daarna wordt de square_list hersteld in de positie voor deze check.  Schaakmat wordt bepaald door te kijken naar het aantal items in de legal_moves_list. Wanneer deze list leeg is het schaakmat.

==== Remise door pat ====

def stale_mate_white()

def stale_mate_black()

def virtual_move()

Om de king moves te bepalen, is het stuk waarin de zetten voor de koning worden bepaald in check_checkmate gebruikt.
Er is een lijst gemaakt met alle velden van het bord. Deze wordt vergeleken met de square_list. Door alle velden bij langs te gaan kunnen we bepalen waar de stukken staan. Daarna wordt een lijst gemaakt met de mogelijke velden waar dit stuk heen verplaatst zou kunnen worden. Deze wordt aan een aantal vergelijkende condities getoetst:

- Als een veld bezet is door een stuk van eigen kleur kan het stuk daar niet heen verplaatst worden, alle volgende velden ook niet.
- Als een veld bezet is door een stuk van de tegenstander kan het hierheen verplaatst worden(slaan), maar niet verder
- Is een veld niet bezet, dat is het een potentiële legale zet

Voor elke potentiële legale zet wordt dan ook weer getoetst of hier door de koning schaak komt te staan. Hier heb ik in plaats van de square_list aan te passen en dan een check te doen om te kijken of de koning schaak staat hier een aparte def virtual_move van gemaakt. Deze doet eigenlijk precies hetzelfde. De zet wordt gedaan in de square_list daarna wordt er gecheckt of de koning schaak staat vervolgens wordt de square_list hersteld. Als het een legale zet is wordt de waarde legal_move=True gereturned. Is het geen legale zet dan wordt de waarde legal_move=False gereturned. Deze gaat terug naar def stale_mate_white/def stale_mate_black. Wanneer True wordt gereturned zal de zet toegevoegd worden aan stale_mate_legal_moves_list. Wanneer deze lijst leeg is, is er geen legale zet en is het remise door pat.

==== Remise door herhaling ====

Remise door herhaling is een herhaling van positie, niet per se van zetten achtereenvolgend. Belangrijk daarbij is te checken of de rechten om te rokeren hetzelfde zijn evenals de rechten om en passant te slaan.

Er wordt initieel een list() gemaakt genaamd: square_list_list. Omdat de square_list een dictionary is met de positie op het bord, wordt deze bij elke zet toegevoegd aan de square_list_list.
Verder wordt er binnen deze definitie bepaald of een speler recht heeft om te rokeren of om en passant te slaan. Deze rechten worden weergegeven in een string die aan de square_list_list worden toegevoegd op dezelfde positie als de square_list van deze beurt. Ook de speler die aan de beurt is, is van belang om te kijken of een positie hetzelfde is. Deze wordt ook toegevoegd op dezelfde index als de rest.

De laatste waarde van de lijst square_list_list wordt vergeleken met alle andere waardes in de lijst. Wanneer er twee gelijke waardes in de lijst staan, wordt er gekeken of deze al in de rep_pos_list (een dict() aangemaakt in def init_state_and_reset()). Wanneer deze niet in rep_pos_list staat betekent

dit dat dit de eerste keer is dat een positie zich herhaalt. Deze positie wordt in de rep_pos_list dictionary opgeslagen met als key de positie(waarde van square_list_list) en als value de integer 2. Dit omdat het de tweede keer is dat dezelfde positie zich op het bord bevindt. Wanneer in de vergelijking van de positie met de rep_pos_list blijkt dat deze er al wel in staat dan is het een derde keer dat eenzelfde positie op het bord plaatsvindt. Dan is het dus remise.

==== Remise door fifty move rule ====

def fifty_move_rule():

Elke zet die er gedaan wordt, wordt deze definitie aangeroepen. De definitie heeft de parameters 'capture' en 'piece'. Deze worden telkens meegegeven als een string. Wanneer de string bij de aanroep van deze functie is: "no capture" en bij piece alles behalve "pawn", dan wordt er 1 bij de fifty_counter opgeteld (Deze begint op 0 in __init__()). Wanneer deze op teller op 100 staat, dus twee keer 50, want er zijn twee spelers die ieders vijftig zetten doen, dan is het remise.

==== Remise door materiaal ====

def draw_by_material():

Remise door materiaal kan op drie manieren plaats vinden. De eerste manier is een remise die meteen plaatsvindt. Deze kan voorkomen wanneer er bepaalde combinaties van stukken op het bord te vinden zijn:

- Koning vs. koning
- Koning vs. Koning & Paard
- Koning vs. Koning & Loper
- Koning & Loper vs. Koning & Loper (Loper zelfde kleurcomplex)

Om te bepalen welke stukken er op het bord zijn is het $5^{e}$ element van de lijst van de value van square_list genomen. Dit is namelijk de piece_name van elk stuk op het bord. Al deze namen van de stukken zijn op hun beurt weer opgeslagen in een list:
white_pieces_draw_list/black_pieces_draw_list. Daarna worden bovenstaande sets vergeleken met de met de draw_lists en aan de hand daarvan kunnen we de eerste drie meteen bepalen.
De Koning & Loper vs. Koning & Loper van hetzelfde kleurcomplex is als volgt opgelost. Er is een lijst gemaakt van de lichte velden, en een lijst van de donkere velden. Wanneer de set {"king", "bishop"} gelijk is aan beide lists en de lopers bevinden zich allebei in één van de licht/donker lijsten dan is het remise. (Instant_draw=True)

De andere manier om remise door materiaal te krijgen is wanneer één speler te weinig materiaal heeft om de andere speler mat te zetten, en de andere speler zijn klok komt op nul. Dan is het remise. Dit wordt op een soortgelijke manier gedaan door sets te vergelijken met de stukken op het

bord. De volgende combinaties kunnen remsie door materiaal op time- out veroorzaken:

- Koning & Paard
- Koning & Loper

Als wit deze stukken overheeft dan is white_draw_TO=True
Als zwart deze stukken overheeft dan is black_draw_TO=True

De volgende definities komen alleen in “PiecesV10.py” voor.

==== FEN-notatie ====

def FEN_notation()

De FEN-notatie is een weergave van het bord in één string. Deze string wordt opgebouwd door de eerste letter van de stukken te gebruiken (Knight is N omdat koning al K is). De witte stukken worden weergegeven met een hoofdletter. De zwarte stukken met een kleine letter. Lege velden worden weergegeven met een integer. Als er meerder lege velden naast elkaar zitten worden deze integers bij elkaar opgeteld. Elke rij van het bord wordt in de string gescheiden door een “\”.

Een voorbeeld van een FEN-notatie:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

Dit is de begin positie voordat het spel begint. Na de reeks aan letters en getallen zien we een ‘w’. Dat betekent dat wit aan de beurt is.
De “KQ” betekent dat de witte koning mag rokeren aan beide kanten (Kingside en Queenside) en de “kq” hetzelfde voor zwart. Wanneer beide kanten niet meer kunnen rokeren komt hier een “-“ te staan.
De “- “ daarna staat voor een potentiële en-passant veld. Als een pion twee stappen vooruitgezet wordt, komt hier het veld die de pion overslaat. Dit is namelijk een mogelijke en-passant veld. Het maakt niet uit of er daadwerkelijk en-passant mogelijk is.
De ‘’0” staat voor de fifty-move-rule counter.
De “1” staat voor de gezamelijke beurt.

De letters van de stukken worden opgehaald door alle velden van het bord te bekijken en welk stuk erop staat en hieraan een letter te binden. De -w- of -b- wordt bepaald door te kijken naar de global turn modulo (%) 2.
De rokade rechten worden bepaalt door te kijken of de koning al gerokeerd is. Of de koning wel of niet verzet is en of de toren van elke kant wel of niet verzet is.
De potentiele en-passent square wordt alleen weergegeven wanneer een pion twee stappen naar voren doet. Door naar de laatste zet te kijken(last_piece, last_row, before_row) kunnen we zien of dit het geval is. Wanneer dit het geval is maken we een string van het en-passant veld, zo niet dan staat er weer een “-“.
De fifty-move-rule wordt rechtstreeks gekopieerd uit de def fifty_move_rule():.
De fullmove (gezamelijke zet) wordt na elke keer dat zwart een zet heeft gedaan verhoogd met één.
Al deze dingen worden gecombineerd tot één string en dan heb je de FEN-notatie.

==== Spelen tegen Stockfish ====

def stockfisch_play()

Stockfish als engine is gedownload en het is geïnstalleerd in de ChessPy directory. Stockfish kan de FEN-notatie interpreteren. Deze wordt als positie meegegeven aan de engine en aan de hand hiervan geeft hij een beste zet. Deze wordt onder de variabele naam best_move opgeslagen. De best_move bevat vier karakters als er geen promotie plaats vindt en er een zet wordt gedaan. Deze zijn het veld van het stuk dat verplaats moet worden en het veld waar het stuk heen verplaatst moet worden bijv. e2e4. Wanneer een stuk promotie heeft het dezelfde velden als eerste vier karakters. De laatste letter is het promotiestuk bijv. e7e8q, wit maakt promotie naar een koningin. De andere letters zijn 'n' voor paard, 'b' voor loper en 'r' voor toren. Wanneer de best_move "None" is dan is er een een uitkomst in het spel daarom wordt deze afgevangen als eerst dan hoeven de rest van de checks niet worden gedaan.

De best_move string wordt opgedeeld in de variablen source_square_sf en dest_square_sf. De stukken die zich op deze velden bevinden worden verwijderd, en het stuk wat op de source_square_sf stond wordt verplaatst naar dest_square_sf. Dit gebeurt zowel met de sprites als met de square_list.

Wanneer er wordt gerokeerd door de engine zal hij ook vier karakters weergeven best_move. Bijvoorbeeld als wit naar de koningszijde rokeert is de best_move: e8g8. Omdat volgens bovenstaande logica alleen de koning verplaatst zal worden naar g8 zal de toren ook nog verplaatst moeten worden, daarom is deze move apart in een conditionele vergelijking gezet.

Het een na laatste stuk in de code is het zorgen voor promotie. Er wordt gekeken naar een string van best_move met de lengte vijf. Daarna wordt er naar het laatste karakter gekeken om te bepalen welk stuk er op het bord geplaatst moet worden. Een nieuwe sprite wordt aangeroepen. Hierdoor wordt de square_list bijgewerkt omdat dit ook al in de def__init__(): gebeurt.

Als laatste wordt gecheckt op schaak staan en schaakmat.

## Behandelen van spel verloop

Bestand: “Engine_gameV2.py”, “Pieces_gameV2.py”

Deze bestanden zorgen voor het spelverloop zoals de timer, pop-ups, het tekenen van de sprites, en een kleine UI om te navigeren. De twee bestanden verschillen niet veel van elkaar alleen de positionering van de timer verandert. Ik vind het echter gemakkelijker om hier twee aparte bestanden voor te maken omdat hier hierdoor voor mij overzichtelijker blijft.

==== Timer ====

def timer():

De timer functie loopt in een aparte thread met de Threading library van Python. Dit omdat zo het spelverloop en de timer tegelijkertijd uitgevoerd worden.
De timer is zo gemaakt dat wanneer de ene timer stil staat de andere gaat lopen (behalve wanneer er op pauze wordt gedrukt dan staan ze beide stil). De while_white_turn gaat lopen als wit aan de beurt is, automatisch gat while not white_turn lopen als zwart aan de beurt is. De pauze functie zit in een while loop die over de white_turn heen loopt. while_timer_run loopt wanneer de timer niet op pauze staat, while not timer_run wanneer de timer op pauze staat. Om te voorkomen dat hij uit de loop gaat nadat erop pauze is gedrukt wordt over dit alles weer heen geloopt met een while timer_run_always.

De tijd die gekozen is vanuit het menu wordt geladen uit het json bestand

De while white_turn en de while not white_turn hebben ieder een aparte clock. Deze is van de Pygame Library. De clock is 100 ticks per seconde gegeven. Door zoveel ticks per seconde te geven wordt bij het wisselen van de klok niet nog een seconde afgesteld voordat hij naar de volgende speler gaat, maar blijft hij ergens tussen de secondes in hangen. Verder wordt de tijd weergegeven in een string. Deze tijd wordt opgeslagen als een string in een global en wordt in de def main(): gebruikt om weer te geven op het scherm. Ook de score wordt bijgehouden in de timer. Dit omdat deze while-loop stop gezet kan worden wanneer er een uitkomst in is het spel hierdoor blijft de score niet telkens optellen als er een uitkomst is bereikt. De main ( die verantwoordelijk is voor het tekenen van alle onderdelen) kan dit niet doen omdat na een uitkomst van het spel nog steeds dingen op het scherm getekend moeten worden.
Binnen de timer zijn de conditionele checks voor de uitkomst van het spelen gedaan door de global variabelen van PiecesV10 voor een Engine_gameV2 of Pieces_PvP voor PvP_gameV2 te vergelijken met een True of False waarde.

==== Tekenen van het spelverloop ====

Er zijn voor alle uitkomsten van het spel teksten en buttons gemaakt die worden weergegeven als deze uitkomst is bereikt. Dit is gedaan weer met de UI_ClassesV3.
Binnen deze pop-ups zijn de buttons met “play again” en “main menu” toegevoegd met de respectievelijke functionaliteit.

Links van het scherm is een kleine UI getekend met de buttons “ main menu “ , “pauze”, “resign” en “offer draw”.  Deze button zijn met MOUSEBUTTON events gelinkt aan bepaalde functies.

Door de functie def place_pieces(): aan te roepen van PiecesV10 of Pieces_PvP kunnen de stukken worden getekend op hun stationaire positie. Door de functie def image_move(): aan te roepen kunnen de stukken ook worden gesleept, omdat deze telkens op hun nieuwe positie worden getekend terwijl er gesleept wordt met de muis.

Aan de rechterkant van het scherm staan de timer. Hierin zit het verschil met de twee bestanden die het spelverloop regelen. Namelijk tijdens het spelen van player vs player wisselen de kleuren telkens van kant, hierdoor staat de timer van de speler die aan de beurt is, net als zijn stukken altijd onder. Daarom worden de positie van de timers telkens omgewisseld als er van beurt gewisseld wordt. In de engine game is dat niet nodig omdat de timer alleen maar onder hoeft te staan. Het enige waar rekening mee moet worden gehouden in dit geval is of er met wit of zwart gespeeld zal worden.

Het spelverloop loopt ook op een klok en deze wordt 60 keer per seconde ververst zodat het zorgt voor een soepel spelverloop.