**Ingenieursproject: Stratego**

**Groep 2**

**Academiejaar 2018-2019**

****

**Stijn Debyser**

**Ayco Neyt**

**Colin Vercruysse**

**Ward Vereecken**

**Tweede Bachelor Industriële Wetenschappen:**

**Informatica**

Inhoudstafel

[1.Inleiding 1](#_Toc8941268)

[2.Onderwerp 1](#_Toc8941269)

[3. Code 2](#_Toc8941270)

[3.1 Main.py 2](#_Toc8941271)

[3.1.1 Tkinter (de klasse SampleApp) 2](#_Toc8941272)

[3.2 Speelscherm.py 2](#_Toc8941273)

[3.2.1 De klasse Spel 2](#_Toc8941274)

[3.3 Spelbord.py: 6](#_Toc8941275)

[3.4 Pion.py: 6](#_Toc8941276)

[3.5 AI.py 7](#_Toc8941277)

[3.5.1 De klasse AI 7](#_Toc8941278)

[4.UML-diagrammen 8](#_Toc8941279)

[4.1.Sequentiediagram 8](#_Toc8941280)

[4.2. Klassendiagram 10](#_Toc8941281)

[4.3. Activiteitendiagrammen 12](#_Toc8941282)

[4.4.Toestandsdiagrammen 14](#_Toc8941283)

[5. Reflectie 15](#_Toc8941284)

[6.Bronnen 16](#_Toc8941285)

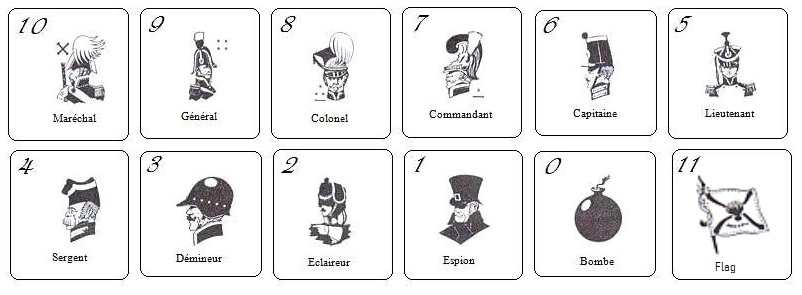
# 1.Inleiding

Voor het vak multidisciplinair ingenieursproject werd ons opgedragen om een algoritme te schrijven in een project naar keuze.

De opbouw van ons project en hoe het allemaal verliep is terug te vinden in dit verslag.

# 2.Onderwerp

Na enig overleg werd er uiteindelijk gekozen voor Stratego. Stratego is een wereldwijd bekend gezelschapsspel. Je kunt het met twee spelers spelen, maar om het voor ons haalbaar te maken werd er snel beslist om één speler te laten spelen tegen een bot/AI.

Stratego is een zeer tactisch spel, iedere speler krijgt een leger van 40 pionnen en moet deze in het begin over zijn hele helft plaatsen. Eenmaal beide partijen al hun 40 pionnen hebben gezet begint het spel. Elk om de beurt mag de speler een pion verplaatsen of een pion van de tegenpartij aanvallen. Elke pion heeft twee vaste waardes, het aantal stappen deze mag zetten en de sterkte van de pion. Op figuur 1 zijn de verschillende pionnen te zien.

Figuur 1: Stratego pionnen

Het doel van het spel is om de vlag te bemachtigen van de tegenpartij. Als je de vlag kunt aanvallen, dan stopt het spel en win je.

Als pionnen aanvallen is het de pion met de hoogste rank die wint, maar er zijn enkele uitzonderingen. Wanneer de spion de maarschalk aanvalt, dan zal de maarschalk verliezen. Wanneer de maarschalk de spion aanvalt, dan zal de maarschalk wel winnen. Een bom wint altijd, behalve tegen de mineur.

# 3. Code

Praktisch ieder lijntje code is volledig zelf geschreven door onze groep, op een paar functies (zoals on\_enter en on\_leave) na. Hieronder extra uitleg i.v.m. iedere klasse en de belangrijkste bijhorende functies.

# 3.1 Main.py

In deze file worden alle schermen bestuurd.

# 3.1.1 Tkinter (de klasse SampleApp)

De view werd volledig uitgebouwd met de package Tkinter. Deze keuze werd vooral gemaakt omdat Tkinter de de-facto standaard GUI package van Python is, en hier dus veel documentatie (en hulp online, bv. op StackOverflow) van te vinden is. Bij nader inzien was dit niet de beste keuze voor ons project. Er werd veel tijd verloren aan problemen die er niet zouden mogen zijn naar onze mening. Zo moesten praktisch al onze methodes waarin een actie van Tkinter moest uitgevoerd worden (Pionnen verplaatsen, knoppen laten werken, etc.) in één klasse geplaatst worden: SetupScherm. Het probleem vormde zich omdat, wanneer een nieuwe klasse werd aangemaakt en in deze klasse moet een methode van Tkinter opgeroepen worden (op de Frame van het op dat moment lopende spel), dan ging dit niet zomaar. De klasse moest afgeleid zijn van de Tkinter-klasse, en zelfs dan ging dit nog niet. Na veel opzoekwerk en proberen, werd er toch besloten om de methodes van het spel in één klasse te plaatsen. Ook werd er te veel tijd verloren aan de layout van het spel: iedere label moest op basis van pixels geplaatst worden en knoppen en dergelijke moesten worden aangemaakt in een externe website en als png op de Buttons geplakt worden.

# 3.2 Speelscherm.py

# 3.2.1 De klasse Spel

In deze klasse komen alle methodes voor die gebruik moeten maken van de package Tkinter. Bij initialisatie worden alle nodige dictionaries en lists aangemaakt, het bord aangemaakt, de lijst met pionnen op het scherm en alle knoppen geplaatst. Hieronder een (korte) uitleg voor belangrijke functies.

* **maak\_bord**:

Het doel van deze functie is vanzelfsprekend. Het raster van knoppen wordt aangemaakt, en aan iedere knop wordt via een lambda-expressie meegegeven wat zij moeten doen wanneer ze worden aangeklikt (cfr. functie ‘klik’), of wanneer de muis erover hovert.

* **update\_lijst**:

Aanvankelijk werd bij iedere nieuwe / verwijderde pion de lijst compleet verwijderd en opnieuw aangemaakt, met 1 waarde die veranderd is (die van de pion in kwestie). Dit is niet echt efficiënt, en moeilijk om aanpassingen aan te brengen zonder veel werk. Uiteindelijk werd deze gigantische functie veranderd naar de kleine functie die ze nu is, door ieder label in de lijst in een aparte dictionary bij te houden (“pionnenaant”, op basis van naam van de pion). Bij een gebeurtenis wordt dan enkel de label van de pion in kwestie aangepast.



Figuur 2: geüpdatete lijst van de speler

* **maak\_label**:

Om duplicated code te vermijden, is er een methode die een pion-label aanmaakt voor ofwel een computer- of spelerpion, in ofwel lijst- of spelmodus.

* **klik**:

De functie klik is een erg belangrijke functie, omdat het bepaalt welke functie wordt opgeroepen bij het aanklikken van een pion. Dit wordt gedaan a.d.h.v. 5 verschillende RadioButtons (waarvan er twee op het scherm getoond worden: Verwijder en Voeg toe).

Iedere RadioButton staat voor een bepaalde fase van het spel. Wanneer RB 0 of 1 aan staan, zit het spel nog in de setup. Klik verwijst de coördinaten van de klik dan door naar de functies voeg\_toe of verwijder. De andere fases zijn dan: de beurt van de speler vóór er een pion is aangeklikt en ná dat er een pion is aangeklikt, en de beurt van de tegenstander.

* **toon\_pion**:

Een belangrijk aspect aan Stratego is dat de pionnen van de tegenstander niet geweten zijn. Bij een aanval van of tegen een vijandelijke pion, moet de tegenstander deze pion dan tonen. Dit bleek moeilijker te implementeren in Tkinter dan gedacht. Wat een triviale functie leek, heeft bijna een dag programmeerwerk opgeleverd. Er werd eerst gewerkt met de functie sleep(), maar dit bleek geen goed idee: het hele programma pauzeert bij het oproepen hiervan, en dit is niet wat we wouden. Bovendien crashte het programma hierdoor geregeld. Daardoor werd er gekozen voor de functie “after”: dit is een functie die simpelweg even wacht en dan een bepaalde functie oproept. Zo wordt de methode toon\_pion recursief opgeroepen, en wordt er d.m.v. een teller gekeken of het de eerste of de tweede keer is dat dit wordt opgeroepen, en dus of de pion zichtbaar, dan wel terug onzichtbaar moet gemaakt worden. Een extra probleem hierbij was dat het programma tijdens het wachten dus verder ging, en het resetten van de teller dus moeilijker bleek dan verwacht. Om dit op te lossen wordt de teller steeds vóór het oproepen van de functie gereset, en niet erna.

* **zet\_ai:**

De methode zet\_AI() zal initieel een lijst met ‘goede’ pionnen aanvullen. Deze lijst wordt verkregen door elke pion, die behoort tot het spelbord van de AI, te gaan controleren of hij een zet kan/mag plaatsen. Zo ja, wordt deze pion toegevoegd aan de lijst. Vervolgens wordt een pion uit de lijst gekozen. Deze zal eerst kijken of hij een pion van de tegenstander al dan niet kan aanvallen. Als dat het geval is, valt hij deze aan, tenzij hij weet dat de pion van de tegenstander een hogere rang heeft. De AI is dus relatief aanvallend ingesteld. Als de pion geen andere pionnen kan aanvallen dan zal deze pion zich verplaatsen.

* **is\_legal\_move:**

Deze methode checkt welke moves een bepaalde pion kan maken. De methode geeft een dictionary terug met een aantal coördinaten als keys en bij elke set coördinaten een Boolean die aantoont of de zet legaal is. Dit wordt gecheckt via de attributen van de klasse Bord: de array ‘opgevuld’ om te zien of er een pion staat, de dictionary ‘pionnen’ om te zien welke pion er staat, de lijst ‘water’ om te zien of de coördinaten wel land bevatten, etc.

* **val\_aan**:

In deze methode kijkt het programma welke pion er wint/verliest in de aanval. Als een pion verliest, wordt deze verwijderd en wordt de lijst geüpdate. Bij het geval dat één van beide pionnen een vlag is, wordt de functie ‘eind’ opgeroepen.

* **eind:**

Aanvankelijk was het de bedoeling om het eindscherm als volledig aparte klasse te schrijven. Omwille van de reeds toegelichte problemen met Tkinter, konden attributen als aantalVeroverdePionnen niet overgezet worden naar andere klasses. Hierdoor werd er besloten om dit in een methode te gieten. Zo kunnen statistieken tijdens het spel worden bijgehouden en op het einde aan de speler getoond worden (cfr. figuur 3).



Figuur 3: eindscherm na overwinning

* **toevoegen:**
* Computer:

Deze methode plaats\_computer gaat een AI aanmaken en de methode voeg\_toe\_computer per pion van de AI opvragen.

Die methode gaat de naam van de aangeklikte pion opvragen en de pion toevoegen aan het bord (,nog voor de setup van start gaat). Ook gaat het het label aanmaken bij de juiste knop.

De laatste methode voeg\_computer\_toe\_spel gaat bij de move, na het verwijderen van de pion, diezelfde pion toevoegen op een andere plaats.

* Speler:

Voor het toevoegen van een spelerpion in de setup wordt gebruikt gemaakt van de methode voeg\_toe. De tweede methode voeg\_toe\_spel doet hetzelfde als bij de computer

* **verwijderen:**

Het verwijderen van pionnen gebeurt met 3 methodes. De eerste methode is een verwijdermethode bij de setup.

De resterende twee zijn een verwijdermethode voor een spelerpion en een computerpion.

# 3.3 Spelbord.py:

Het bestand Spelbord.py bevat de belangrijke klasse 'Bord'. Deze klasse behoort tot de model van het project.

Bij initialisatie krijgt het bord enkele variabelen mee.

Deze instantievariabelen zijn:

1) De lengte en breedte van het bord

2) Een lege dubbele array van Booleans om te checken welke coördinaten zijn opgevuld

3) Een dictionary voor een pion met zijn coördinaten om te checken welke pion op een bepaalde plaats staat

4) Twee dictionaries met elk de pion met zijn aantal, één voor de speler en de andere voor de computer

Verder bevat de klasse twee functies, namelijk het toevoegen en verwijderen van een pion op het bord. De methode voeg\_pion\_toe\_spel bevat een parameter zijde. Deze parameter bepaalt van wie de pion is.

# 3.4 Pion.py:

De klasse pion is vooral opgebouwd uit de eigenschappen van een bepaalde pion. Hier bevinden zich geen methodes die iets zouden veranderen aan het spel.

Wel bevat de klasse een hoop getters en setters, zodat we gemakkelijk de eigenschappen van de pionnen kunnen aanpassen of opvragen.

# 3.5 AI.py

# 3.5.1 De klasse AI

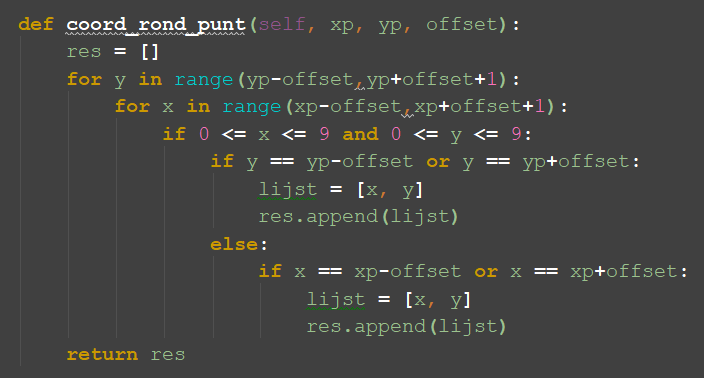
In de klasse AI wordt logischerwijs alles voor de AI berekend. De AI houdt een bord met zijn eigen pionnen (aib) en een (aanvankelijk leeg) bord met pionnen van de Speler bij (ais)\*. Op deze manier kunnen strategische beslissingen gemaakt worden. Bij het opstarten van het programma gaat deze klasse meteen een bepaalde setup berekenen. Dit wordt gedaan op basis van bekende opstellingen en de meest succesvolle opstellingen uit de jaarlijkse kampioenschappen van Stratego. Om een voorbeeld te geven, wordt de Spion steeds vlak bij de Generaal geplaatst, omdat de Generaal enkel door de Maarschalk kan verslagen worden, maar de Maarschalk enkel door de Spion. Ook maakt de klasse bij initialisatie een Boolean aan die aangeeft of de opstelling agressief of defensief gaat worden aangemaakt (bvb. de Maarschalk op de voorste rij of bij de vlag). Aanvankelijk was het het idee om dit attribuut ook te gebruiken in het spel zelf, maar door tijdsgebrek werd dit niet geïmplementeerd. Vanaf de meest belangrijke pionnen geplaatst zijn, wordt de rest willekeurig geplaatst. Aanvankelijk was het idee om het Partially Obstructive Markov Decision Process (POMDP) algoritme te implementeren. Dit algoritme wordt speciaal gebruikt bij situaties zoals in Stratego: de AI weet in het begin niet waar de pionnen van de speler staan. Deze berekent dan statistisch via de kracht van iedere pion en het aantal pionnen van deze soort, wat de kans is van iedere pion om ergens te staan. Dit werd uiteindelijk niet geïmplementeerd.

\*De AI houdt 2 extra borden bij:

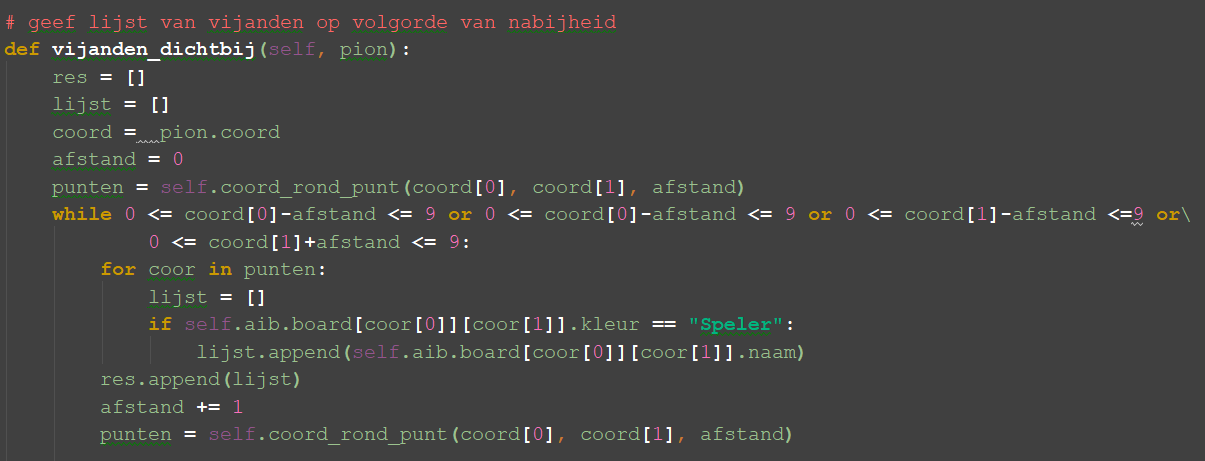
* Het eerste houdt de gekende pionnen bij met hun waarde van de tegenstander. Dit wordt gebruikt door de AI om slimmer te kunnen aanvallen. Indien de AI een pion wil aanvallen zal hij eerst checken of die tegenstander gekend is, indien ja, zal het afhangen of de AI-pion een hogere waarde heeft of niet. Als de AI-pion de aanval zal winnen zal hij aanvallen. In het andere scenario zal de pion proberen weg te vluchten.
* Het tweede bord houdt alle pionnen bij die kunnen bewegen. Hierdoor kan de AI weten welke pionnen er sowieso geen bom zijn. In bepaalde scenarios, zoals wanneer de sterkste pion van de speler dood is maar die van de AI niet, kan de AI dan weten dat deze iedere beweegbare pion achterna kan gaan.

**Ongebruikte functies coord\_rond\_punt en vijanden\_dichtbij:**

Om de AI slimmer te maken werden de functies functies coord\_rond\_punt en vijanden\_dichtbij geschreven. Vanwege tijdsgebrek konden deze niet geïmplementeerd worden. De functies werken samen om van een bepaalde pion van de AI, verschillende lijsten van speler-pionnen terug te geven rondom deze AI-pion, op volgorde van afstand (één tegel, twee tegels,… tot het hele bord is overlopen). M.b.v. deze methodes zou de AI dan meer in de toekomst kunnen denken: een bepaalde pion kan nu bijvoorbeeld 3 vakken verder zien dat er hier een sterkere pion staat en beslissen om niet in die richting te bewegen.



Figuur 4: de functie coord\_rond\_punt



Figuur 5: de functie vijanden\_dichtbij

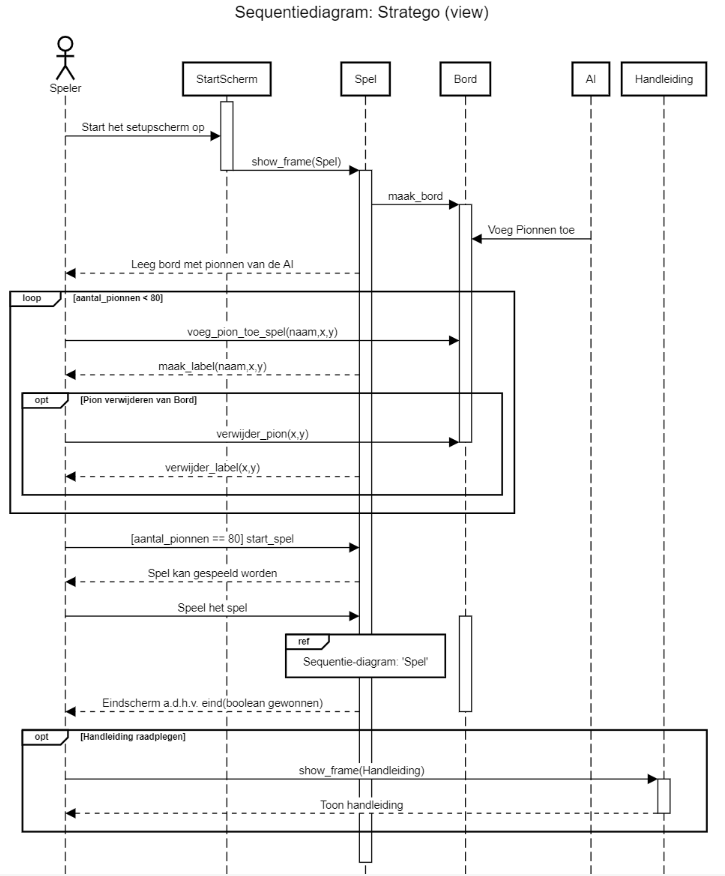
# 4.UML-diagrammen

Hieronder extra uitleg bij de UML-diagrammen.

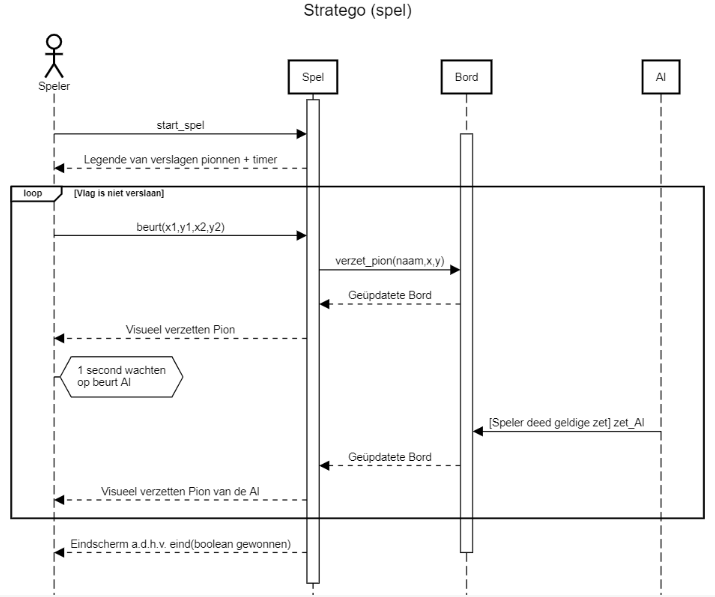
## 4.1.Sequentiediagram

Het sequentiediagram voor dit project wordt opgesplitst in twee delen. Waarbij het tweede diagram gerefereerd wordt in het eerste om plaatsgebrek en onduidelijkheden tegen te gaan. Het eerste sequentiediagram, zoals te zien in figuur 6, stelt het algemeen verloop voor van de applicatie. Bij het starten van de applicatie zullen alle schermen automatisch aangemaakt worden, maar niet gebruikt. Daarom staan deze objecten ook bovenaan, ondanks ze niet direct gebruikt zullen worden. Deze methode maakt het verwisselen van schermen gemakkelijker en gebruiksvriendelijker. Bij het toevoegen en verwijderen van pionnen wordt een verschil gemaakt tussen de model en de view. Wanneer de speler handmatig een pion wil toevoegen of verwijderen worden enkele stappen doorlopen. Ten eerste zal de model ervoor zorgen dat de pion correct wordt behandeld. Ten tweede geeft de klasse Bord de gegevens door naar de view, namelijk het SetupScherm. Als laatste zal de juiste pion zichtbaar worden toegevoegd.

Het tweede diagram, zoals te zien in figuur 7, is een simpel diagram betreffende het spelverloop. Wanneer de speler aan de beurt is zal de methode ‘beurt’ opgeroepen worden. Deze methode bevat 4 parameters, namelijk tweemaal een coördinatenpaar. Het eerste paar (x1 en x2) zijn de huidige coördinaten van de pion, gegeven om de juiste pion te verzetten. Het laatste paar is de eindpositie. Hierbij wordt vermeld dat de methode test op een mogelijke zet, omdat sommige zetten verboden of onmogelijk zijn in het spel. Wanneer het spel beëindigd is, zal er een gepast eindscherm tevoorschijn komen.



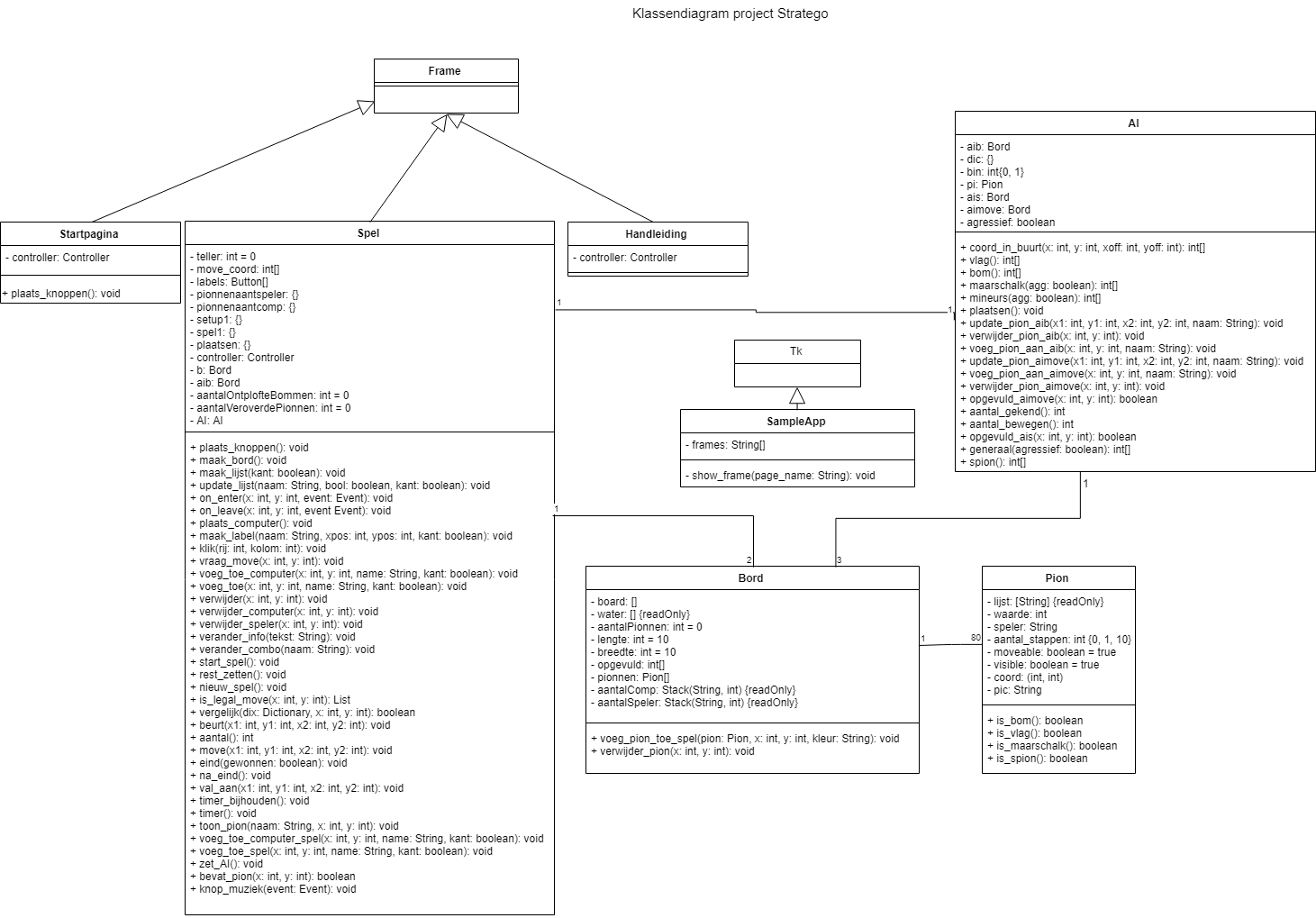
Figuur 6: Sequentiediagram applicatie



Figuur 7: Sequentiediagram spel

## 4.2. Klassendiagram

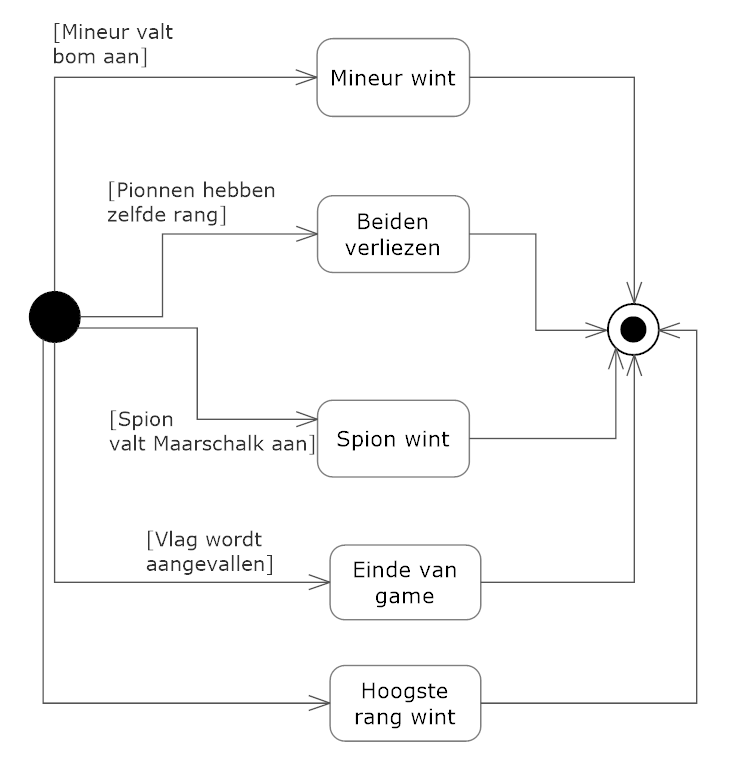
In het klassendiagram worden alle klassen met bijhorende methodes en attributen gebundeld. Het klassendiagram staat weergegeven in figuur 9.



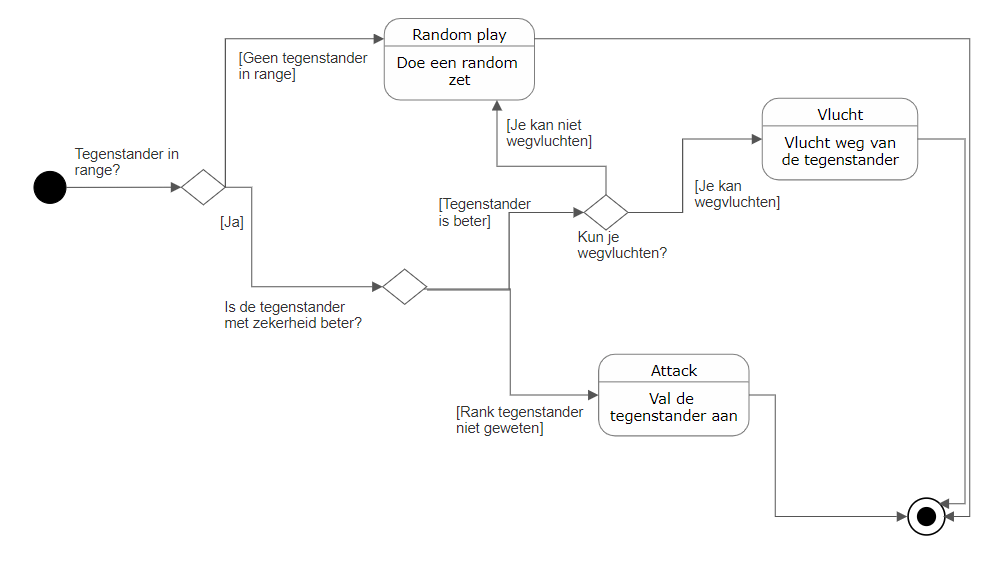
Figuur 8: Klassendiagram

## 4.3. Activiteitendiagrammen

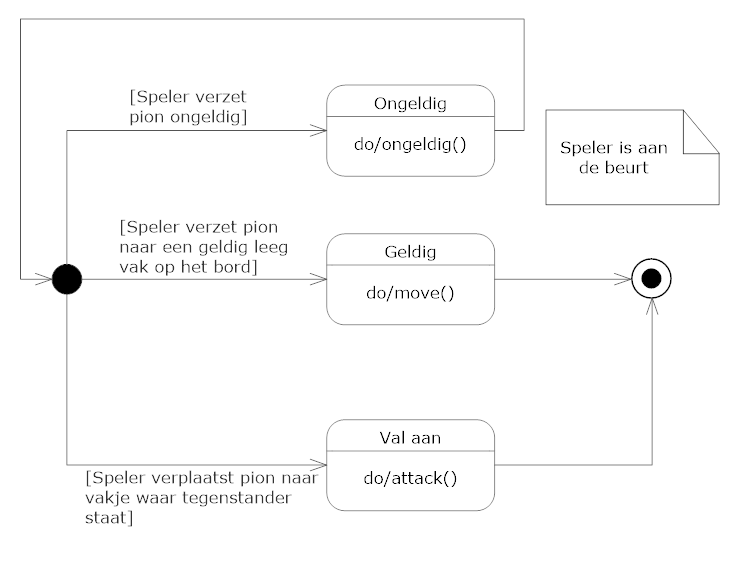
Activiteitendiagrammen geven een gedetailleerde beschrijving van alle activiteiten die binnen een methode worden uitgevoerd en laten toe het intern verwerkingsproces binnen een methode gedetailleerd voor te stellen. Hieronder staan enkele activiteitendiagrammen uitgewerkt. In figuur 5 en figuur 6 wordt respectievelijk de methode val\_aan en de aanval van de AI getoond. Ten slotte wordt in figuur 7 het activiteitendiagram van de beurt van de speler weergegeven.



Figuur 9: Activiteitendiagram attack

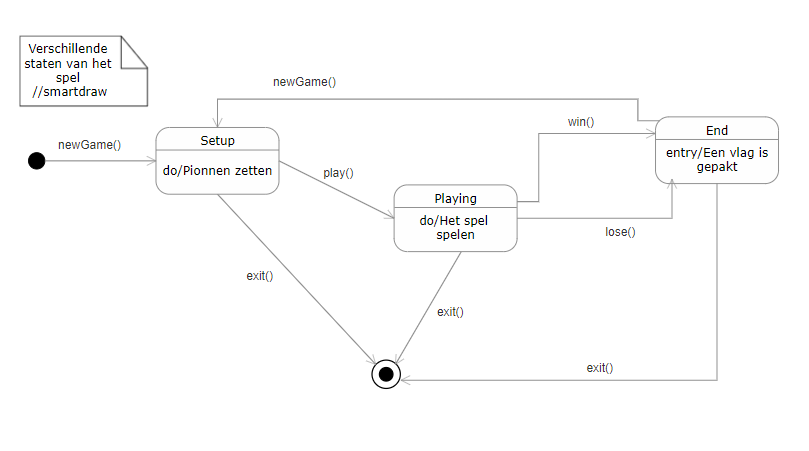


Figuur 10: Activiteitendiagram AI



Figuur 11: Activiteitendiagram beurt speler

## 4.4.Toestandsdiagrammen

In een toestandsdiagram worden de mogelijke toestanden van een object weergegeven. In figuur 8 wordt een toestandsdiagram van het spel getoond.

Figuur 12: Toestandsdiagram spel

# 5. Reflectie

Bij het begin van het project wisten we niet direct hoe we het moesten aanpakken. Maar eenmaal er een eerste GUI tot stand kwam en de groep wat beter begon te communiceren, ging alles plots een stuk sneller. Leren werken met Tkinter was het grootste probleem. Hieraan werd het meeste tijd besteed, waardoor er uiteindelijk minder tijd was om aan de eigenlijke AI te werken (die pas getest kon worden van zodra het spel werkende was). Beter voor dit project was de package PyGame (de naam gaf dit misschien een beetje weg…).

Uit dit project werd zeker en vast geleerd. Zo blijkt dat het splitsen van de model en de view van een programma enorm belangrijk is. Hiermee werd in het begin rekening gehouden, maar na enkele commits niet meer. Dit is ook te merken aan de vele methodes die zich in de klasse Spel bevinden. Als team hebben we hier zeker de nadelen van ondervonden, maar het was te laat om de model en view te herverdelen. Hieruit heeft het team veel geleerd.

Uiteindelijk is de AI dus niet zo uitgebouwd als we wouden. Dit kwam voornamelijk door tijdsgebrek. Een betere AI was waarschijnlijk mogelijk indien we een werkende versie van Stratego op voorhand hadden gedownload en geïnstalleerd van het internet. Onze argumentatie hiertegen was dat we veel minder goed zouden weten hoe de code van dit spel in mekaar zit, en het dus veel uitdagender zou worden om hiervoor een AI uit te bouwen.

Drie weken voor het indienen van het project kregen we een tegenslag te verduren. Het 5de lid van de groep gaf aan dat hij het project zou laten vallen. Deze persoon vertelde dat hij zich vooral met het algoritme al ging bezighouden en dat deel grotendeels al ging klaarzetten. Toen hij ermee stopte kwamen we te weten dat hij nog niets had gedaan. Hierdoor moesten we de AI grondig aanpassen en beperken.

# 6.Bronnen

Oakley, B.(2011, January 24). How do I achieve the following Tkinter GUI layout with either pack or grid? Retrieved from <https://stackoverflow.com/questions/4785856/how-do-i-achieve-the-following-tkinter-gui-layout-with-either-pack-or-grid>

Oakley, B.(2011, September 26). Switch between two frames in tkinter. Retrieved from <https://stackoverflow.com/questions/7546050/switch-between-two-frames-in-tkinter/7557028#7557028>

Furas(2016, January 28). Using a frame class in a Tk class in Python tkinter. Retrieved from <https://stackoverflow.com/questions/35051463/using-a-frame-class-in-a-tk-class-in-python-tkinter>

Best Stratego Piece Setups. (n.d.). Retrieved from <https://www.ultrastratego.com/setups.php>

Tips to win Stratego. (n.d.). Retrieved from <https://www.ultrastratego.com/tips.php>

Petkun, Jeff & Tan, Ju. “Senior Design Project: Stratego”. University of Pennsylvania Spring 2009, April 2014. Retrieved from <https://www.seas.upenn.edu/~cse400/CSE400_2008_2009/websites/petkun_tan/download%20files/Stratego%20Senior%20Design%20Report.pdf>