Eindopdracht Programmeren: SET

Jens Schenkhuysen (7768621), Mila Suboticki (7850360) & Chelsea Vanmeulebrouk (5652448)

Inhoudsopgave

[Vraagstelling 3](#_Toc76123641)

[Algoritme 5](#_Toc76123642)

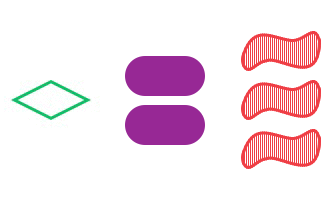
[Handleiding 12](#_Toc76123643)

[Resultaten 15](#_Toc76123644)

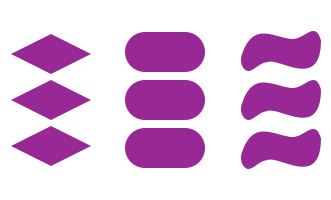
[Conclusie, discussie en taakverdeling 15](#_Toc76123645)

# Vraagstelling

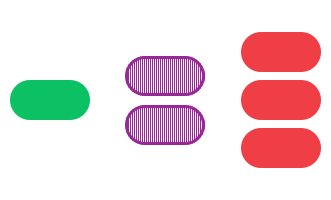
Voor de eindopdracht van ‘Programmeren in de Wiskunde’ werd ons gevraagd om een programma te schrijven waarmee het spel SET gespeeld kan worden tegen de computer. Het doel van het spel is om zoveel mogelijk sets te vinden. Op de kaarten van het spel staan symbolen die beschikken over vier verschillende eigenschappen, namelijk kleur (rood, groen of paars), vorm (ruit, golf of ovaal), vulling (gestreept, leeg of solide) en aantal (1, 2 of 3). Aangezien elke mogelijke combinatie van deze eigenschappen maar één keer voorkomt in het spel, komen we in totaal op 34=81 unieke kaarten. Een set bestaat uit drie kaarten. Per eigenschap moet worden nagegaan of deze op alle drie de kaarten verschillend is óf op alle drie de kaarten hetzelfde. Als een van de twee voorgaande dingen geldt voor alle eigenschappen van de symbolen op de drie kaarten dan spreek je van een set (Figuur 1). Als je twee willekeurige kaarten neemt, dan is er altijd maar één kaart uit de overige negenenzeventig kaarten die met die twee kaarten een set vormt. Er kunnen gebaseerd op deze 81 kaarten 1080 unieke sets gevormd worden. Geen twee kaarten zijn identiek aan elkaar, dus binnen een set moet in ieder geval altijd één van de vier eigenschappen verschillend zijn (Figuur 2). Zie Figuur 3 als voorbeeld voor een combinatie van drie kaarten die bijna een set lijken te vormen, maar net geen set zijn, omdat één van de vier eigenschappen niet voldoet aan de eisen. De vulling van de symbolen is namelijk op twee van de drie kaarten solide en op maar een van de drie niet.



Figuur 1: Dit is een set waarbij alle vier de eigenschappen verschillend zijn tussen de drie kaarten.



Figuur 2: Dit is een set waarbij er maar één eigenschap verschillend is tussen de drie kaarten, namelijk de vorm van de symbolen. Verder zijn kleur, vulling en aantal hetzelfde.



Figuur 3: Dit is dus geen set, omdat voor één van de vier eigenschappen, namelijk vulling, er geldt dat twee van de drie kaarten dezelfde vulling hebben en maar één kaart een andere vulling. Voor de overige drie eigenschappen geldt dat die of wel helemaal verschillend zijn of helemaal hetzelfde tussen de drie kaarten.

Het spelverloop van SET is als volgt: Er komen telkens twaalf kaarten op tafel te liggen, waartussen de speler een set moet vinden. Mocht het zo zijn dat er tussen de twaalf kaarten op tafel geen set te vinden is dan spreken we van een cap set. Bij het normale kaartspel wordt een cap set opgelost door drie nieuwe kaarten van de stapel bij te leggen tot een totaal van vijftien kaarten die dan op tafel liggen. Echter bij het spel tegen de computer zal een cap set opgelost worden door de eerste drie kaarten van de tafel te verwijderen en drie nieuwe kaarten van de stapel toe te voegen.

Het schrijven van een programma waarmee je SET kunt spelen tegen de computer brengt een aantal uitdagingen met zich mee. De voornaamste uitdagingen zijn om een code te schrijven voor een goed werkende timer, om het spelverloop zo soepel en duidelijk mogelijk te laten verlopen voor de speler en om de kaarten op een zo goed mogelijke manier te visualiseren. Je wil namelijk dat de speler een tijdslimiet heeft waarbinnen hij/zij/hun een set moet hebben gevonden. Als de speler binnen die tijd geen set heeft gevonden, zal de computer een punt krijgen. Het programmeren van een timer die op de achtergrond blijft lopen en de computer een punt geeft zodra die bij nul is aangekomen is nog best ingewikkeld en ook wat in ons geval de meeste tijd heeft gekost. Daarnaast wil je een code schrijven die het spelen van het spel zo makkelijk mogelijk maakt voor de speler. Je wil dat de speler zo min mogelijk zelf nog hoeft te doen en het liefst eigenlijk dat hij/zij/hun alleen nog maar op start hoeft te drukken en sets hoeft door te geven.

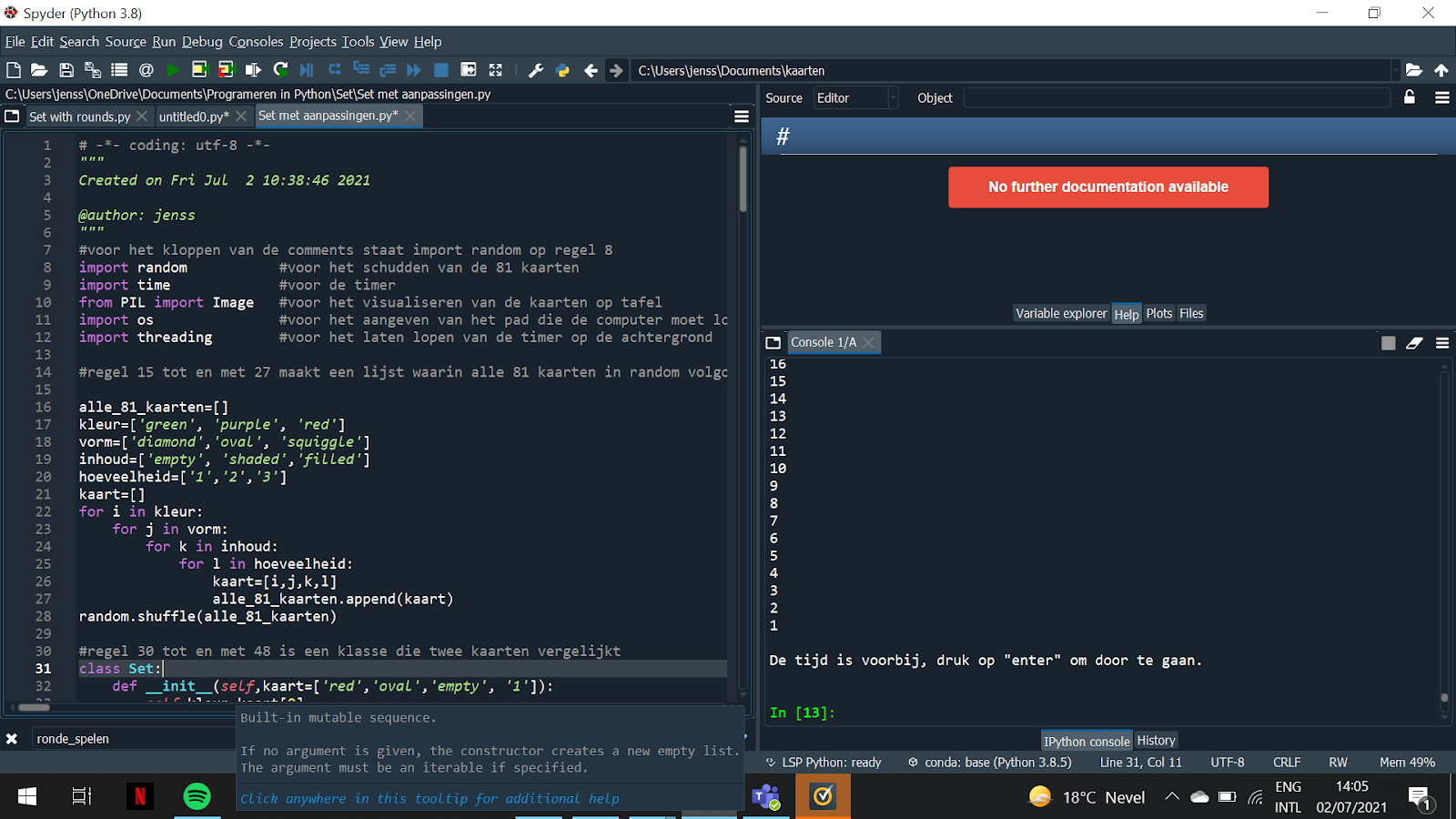
# Algoritme

Wij zullen nu eerst het algoritme achter onze code uitleggen, het doel van ons algoritme is om een ronde te spelen, hierdoor kunnen wij deze functie meermaals gebruiken om uiteindelijk het gehele spelverloop te programmeren.  
Ons algoritme kent de volgende algemene stappen:

1. Het aanzetten van een timer die tegelijkertijd als het spel loopt.
2. Kijken welke kaarten op tafel liggen.
3. Als dit een capset is de eerste drie aanmerken om later te vervangen en de ronde stoppen.
4. Anders de gebruiker vragen om een input.
5. Controleren of deze input juist is, de timer stoppen, punten geven en het aanmerken van een door de computer/ door de speler gespeelde set.
6. Vervangen van de aangemerkte kaarten.

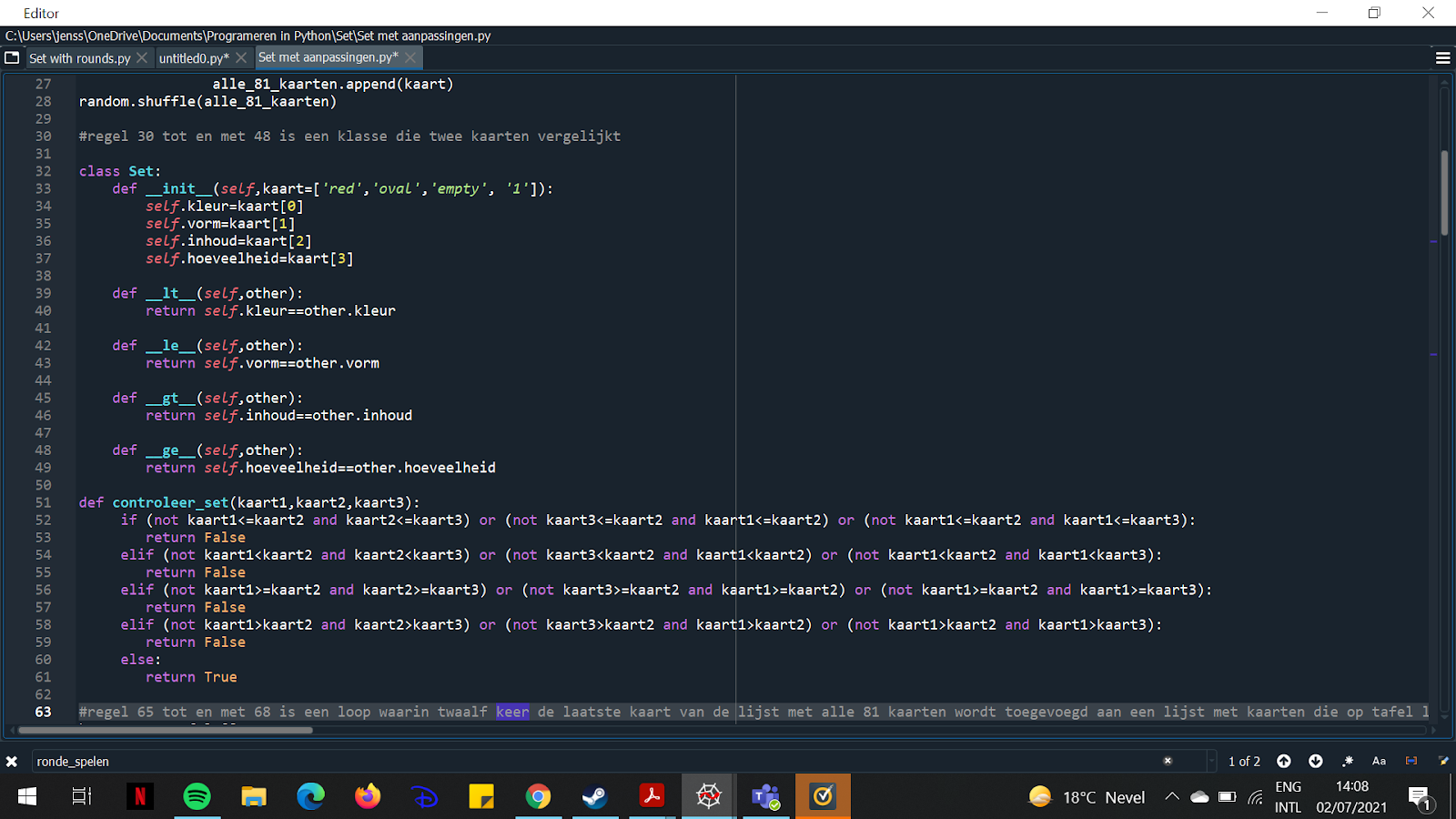
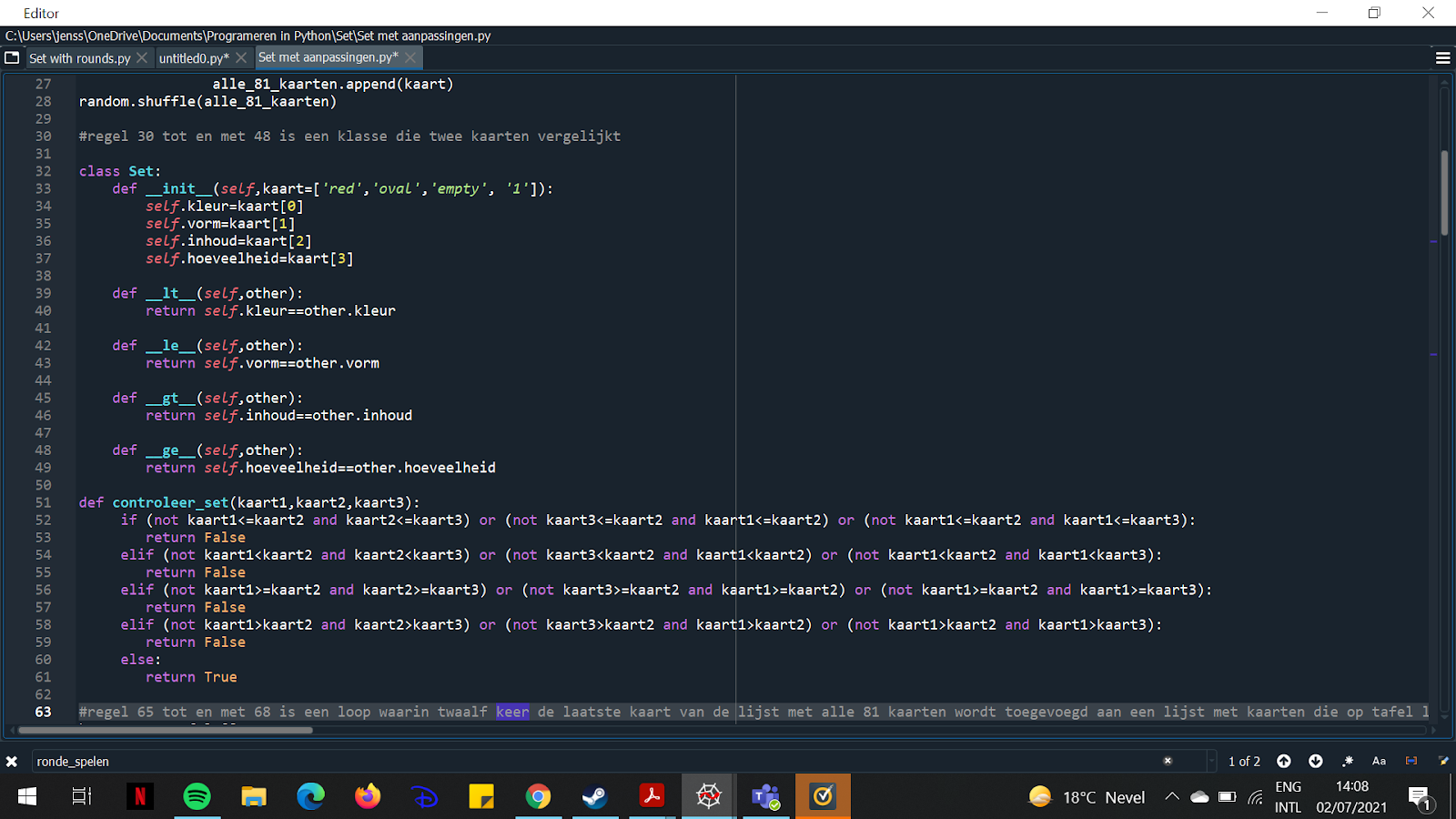
Bovendien hebben wij om het gehele spel te programmeren de beginsituatie nodig en tevens het stopzetten van het spel.

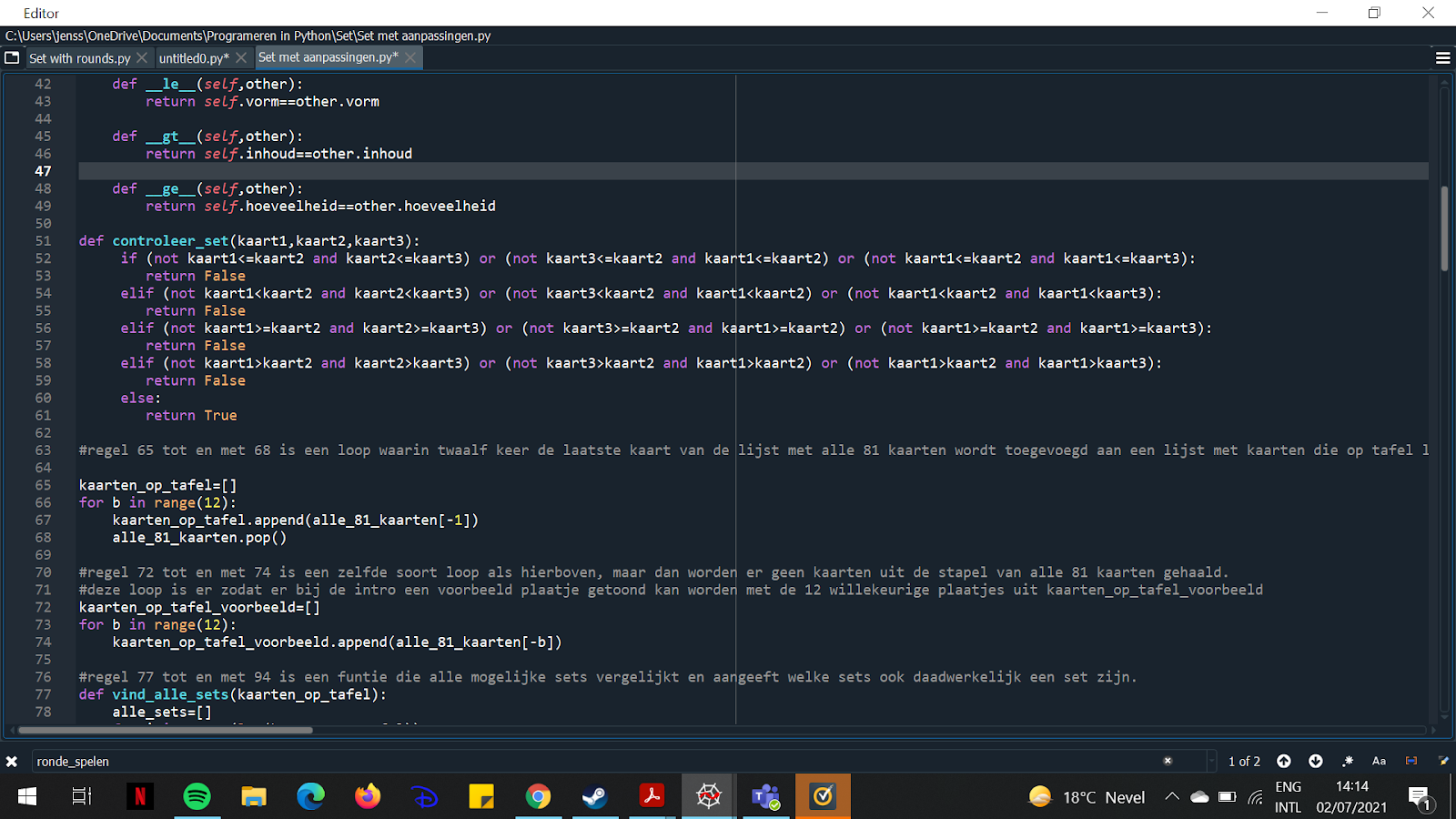
Om uitleg te geven over de werking van de code, en de gemaakte keuzes tijdens het programmeren zullen we de gehele code stap voor stap behandelen.

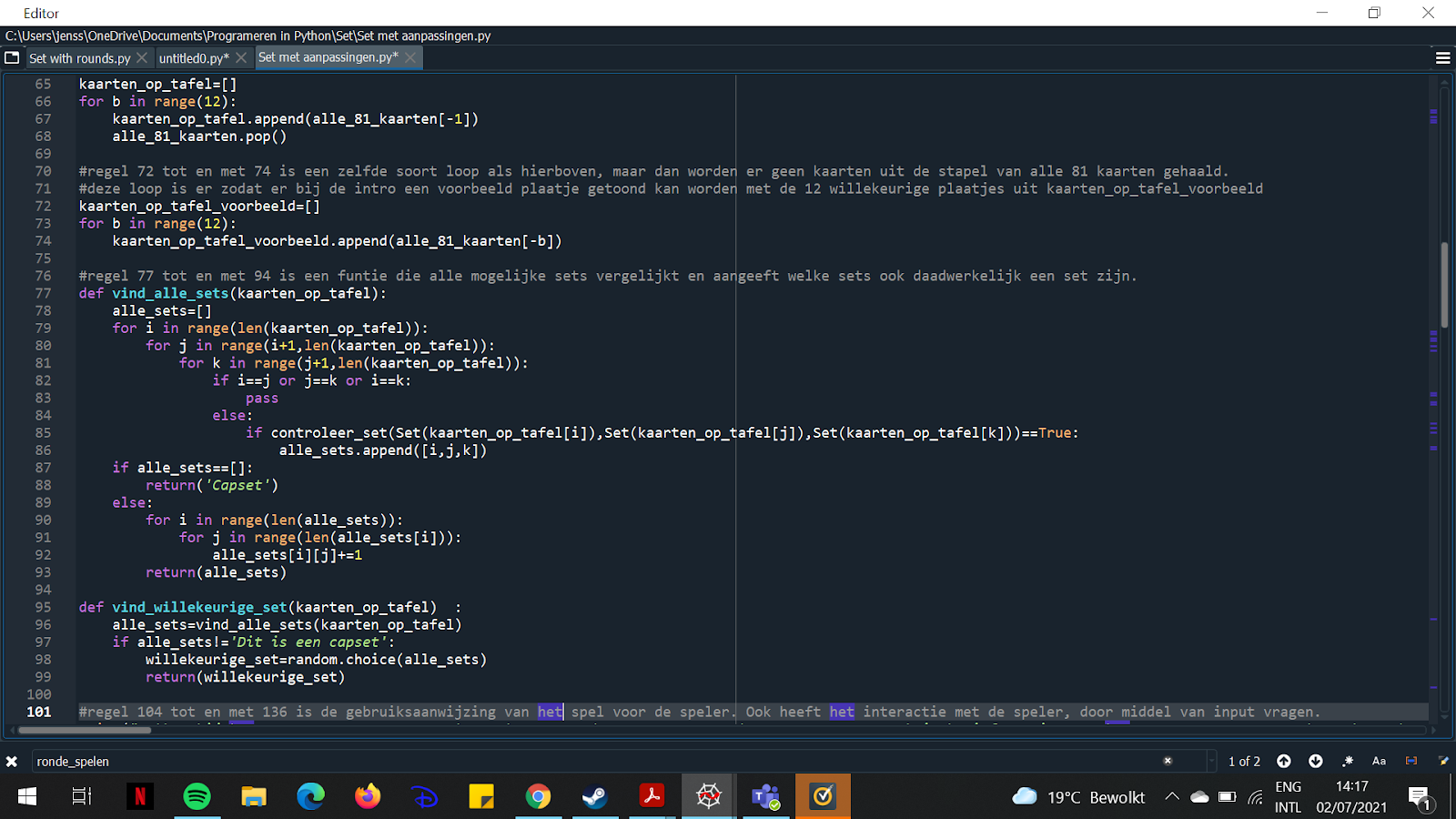
Om het spel Set te kunnen spelen moeten we eerst de kaarten aanmaken. Wij hebben ervoor gekozen om elke kaart uit een lijst te laten bestaan, een voorbeeld hiervan is: [red,oval,empty,1]. Op die manier kunnen we alle vier de attributen op een vaste plek in de lijst zetten, zodat we deze makkelijk individueel kunnen vergelijken. Dat doen we door naar het i’de element uit de lijst te kijken. Ook kunnen we de kaart als geheel aanroepen door de hele lijst te noemen. Aangezien elke unieke combinatie van de vier eigenschappen eenmaal voorkomt, kunnen we een algoritme schrijven dat voor elk van de drie mogelijkheden deze aan een lijst toevoegt. We doen dit d.m.v. vier for loops, de eerste for loop maakt drie lijsten met ieder een unieke kleur aan, de volgende maakt van elke verzameling nog drie verzamelingen met een unieke vorm, vervolgens gebeurt hetzelfde met inhoud en hoeveelheid en worden er in totaal 34 unieke kaarten aangemaakt.

We hebben ervoor gekozen de kaarten bij te houden met een lijst, omdat deze universeel is en we met behulp van het pop commando een kaart kunnen ‘trekken’. Ook was dit de handigste manier om het visualiseren goed te coderen, hier gaan we later meer op in.  
Tot slot hebben we de random module gebruikt om de lijst in een random volgorde van kaarten te zetten zodat de volgorde waarin kaarten getrokken worden uniek is, en ieder spel anders zal verlopen.

Omdat de opmaak van deze lijsten identiek blijft kunnen we gemakkelijk de verschillende eigenschappen van kaarten vergelijken. Om deze kaarten te kunnen vergelijken voeren we een klasse in. Deze klasse voert enkel notatie in om de 4 eigenschappen te kunnen vergelijken, wij hebben gekozen om deze vergelijkingen te schrijven als, >,>=,< en <=. Deze operaties werken al op twee kaarten en geven True of false aan. We kunnen deze nu koppelen aan de tekens en deze nieuwe betekenissen geven. We hebben de eigenschap is hetzelfde (True) en de eigenschap is verschillend (False) we kunnen als gevolg deze notatie gebruiken om kaarten te vergelijken. Wij hebben niet gekozen voor een == teken aangezien wij vier verschillende dingen controleren en het verwarrend zou kunnen werken voor de kaart is hetzelfde en de eigenschap is hetzelfde.

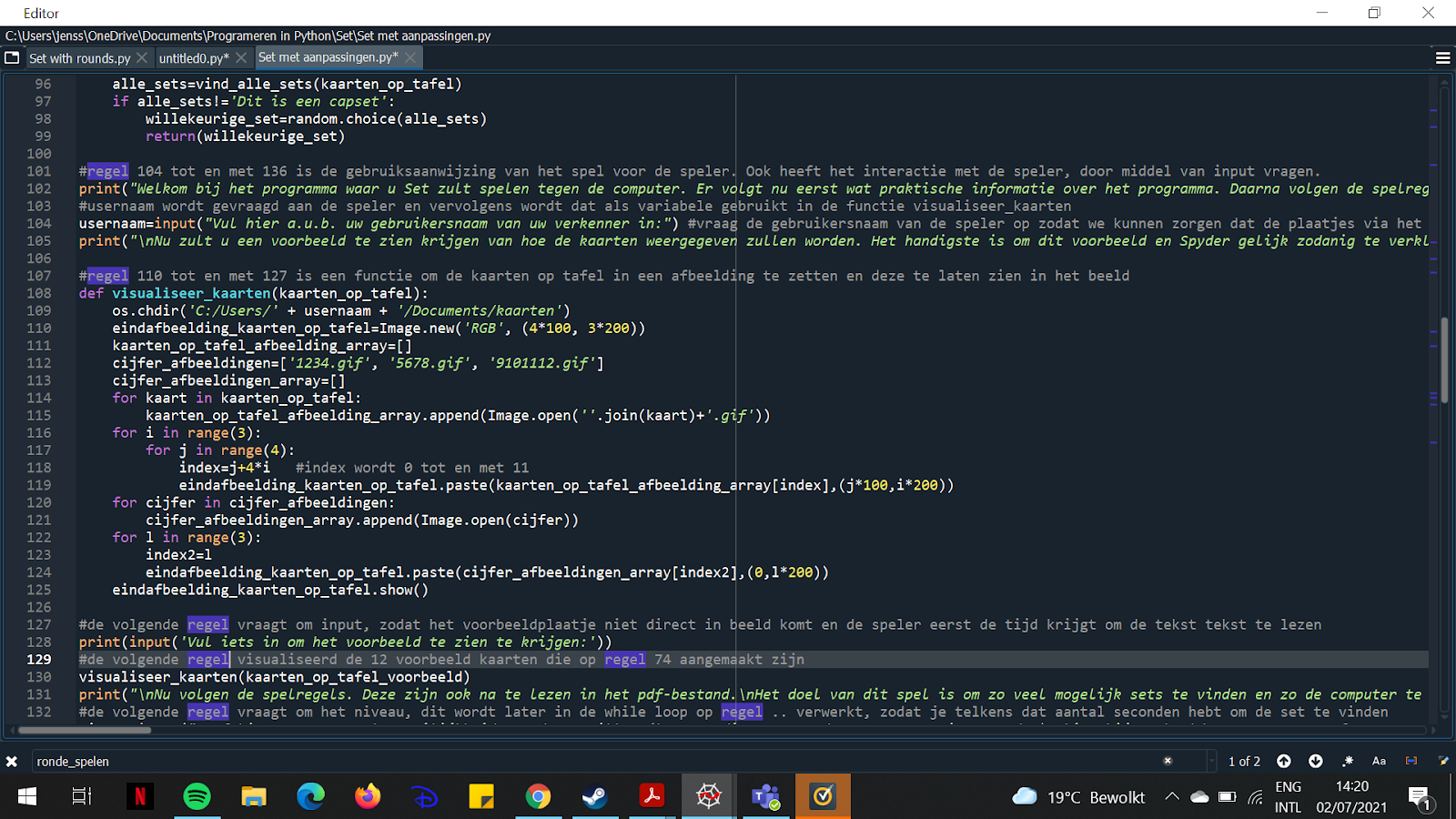
  
Het vergelijken van de kaarten wordt in de daarop volgende functie gedaan. Door deze notatie te gebruiken kunnen we alle vergelijkingen die we willen maken snel doorvoeren.  
Wij hebben ervoor gekozen om hierbij niet te controleren of een groep van 3 kaarten een set is, maar juist of dit géén set is. Als immers aan één van de vier eigenschappen niet voldaan is, dan kan de groep nooit een set zijn, en kunnen we dit dus meteen uitsluiten. We doen dit met return (False). We kunnen dit controleren door te kijken of de groep precies 2 kaarten heeft met dezelfde eigenschap, als dit voor alle vier de eigenschappen onwaar blijkt te zijn, vinden we dat de drie kaarten een set is. Dit is in het programma verwerkt door als de vier groepen eigenschappen niet voldoen aan de ‘faal’ conditie, door te geven met het commando return(true) dat de groep een set is.

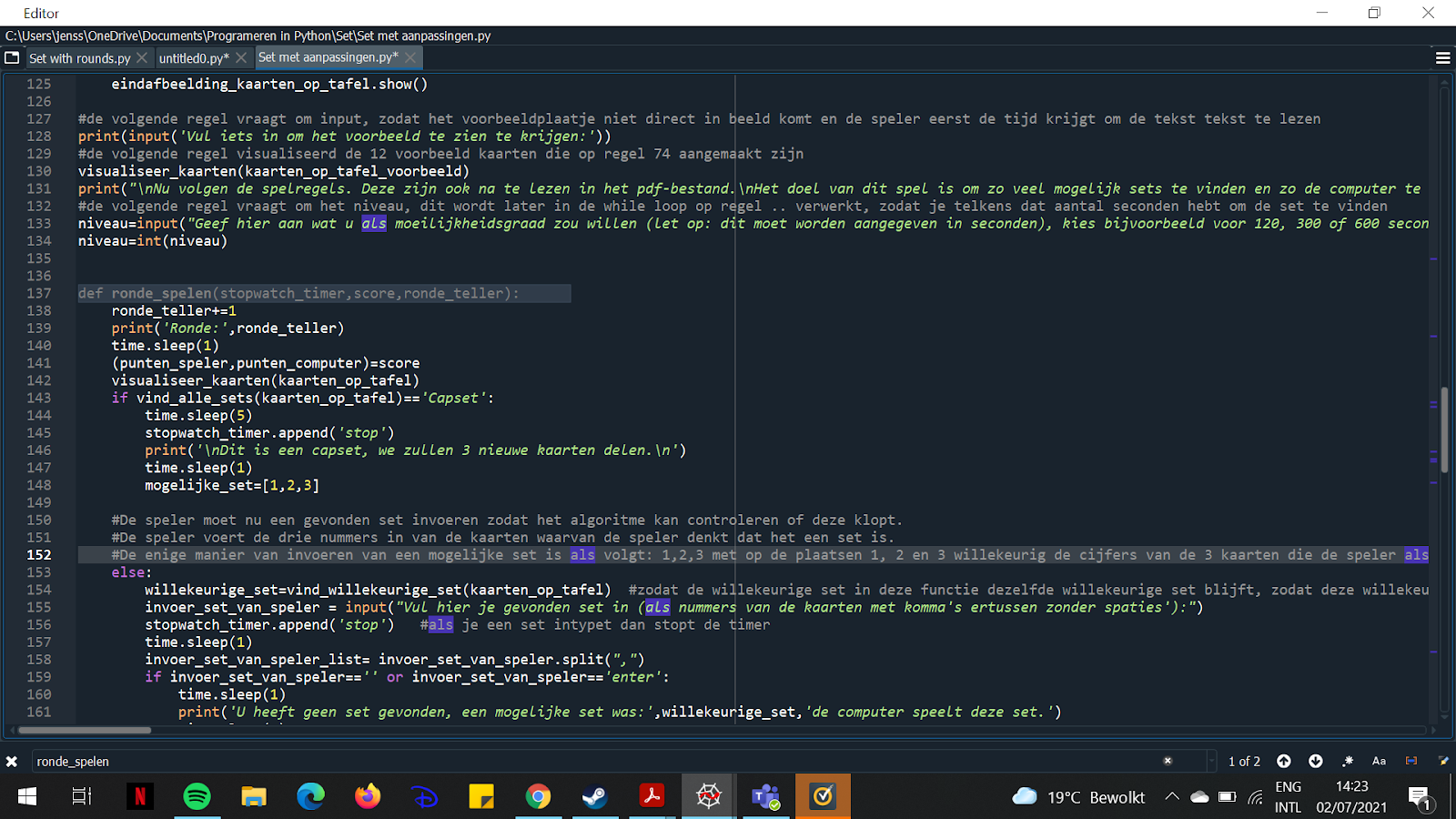
Ons spel wordt altijd gespeeld met een verzameling van 12 kaarten op tafel, deze wordt aangemaakt door de laatste twaalf kaarten te pakken in de al geschudde verzameling van alle kaarten, er is ook hier weer gekozen van een lijst zodat tijdens het spel gebruik gemaakt kan worden van de lijst-operaties welke het verwijderen en opnieuw toevoegen van de gespeelde, en nieuwe kaarten, makkelijk maakt.

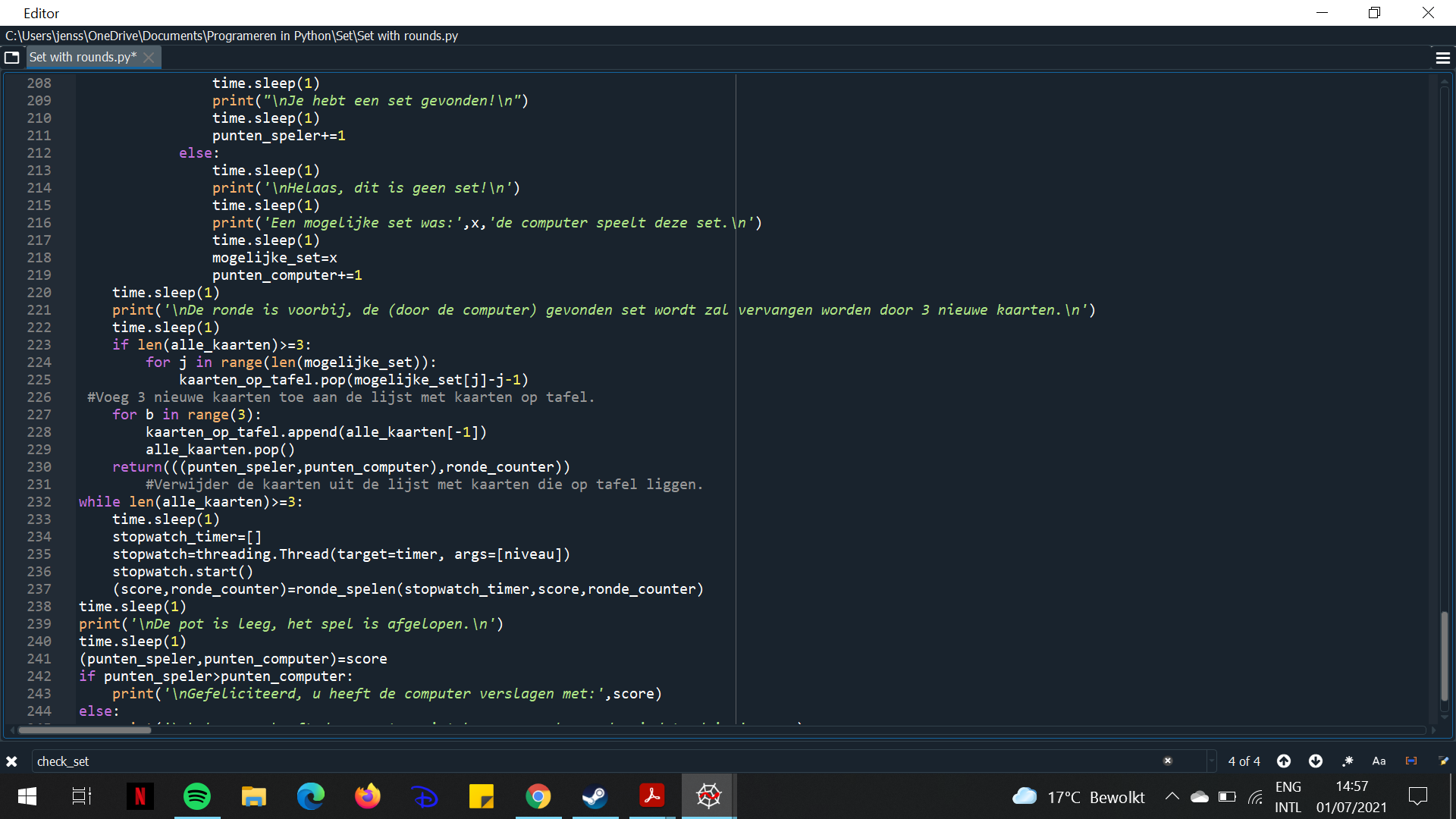
Wij hebben tevens een algoritme geschreven welke uit deze verzameling van 12 kaarten, alle mogelijke sets vindt. het achterliggende idee van dit algoritme is om dit te vinden door alle combinaties van 3 kaarten (i,j,k) te controleren dit wordt gedaan door middel van de eerder opgestelde controleer\_set functie, immers kan er bij elk tweetal kaarten een derde kaart gevonden worden zodat deze een set vormt, we kunnen hieruit dus niet afleiden of dit wel of geen set is.

We hebben de code efficiënter kunnen maken door i,j en k zo te kiezen dat (i<j<k), hierdoor worden groepen welke dezelfde kaart twee keer bevat niet gecontroleerd en dezelfde groepen enkel in een ander volgorde niet dubbel gecontroleerd. Bijkomend voordeel is dat deze groepen dus ook niet door het algoritme worden gevonden en dus enkel alle unieke sets gegeven worden. We kunnen met behulp van deze functie een willekeurige set vinden, namelijk door een willekeurig element uit de gegenereerde lijst te halen. Ook hier is gebruik gemaakt van lijsten om toegang te hebben tot for loops om makkelijk alle combinaties te krijgen, en ook om groep makkelijk door de controleer\_set functie te halen. We introduceren hier ook het begrip Capset, als we namelijk geen set gevonden hebben, betekent dit dat er tevens geen mogelijk is. We hebben dus twaalf kaarten waarin geen set te vinden is.  
we geven dit tevens als uitkomst van de functie zodat we dit op een later moment kunnen herkennen en onze functie erop aan kunnen passen. Zo hebben wij bij vind\_willekeurige\_set dit als uitzondering gemaakt, om te voorkomen dat het programma een set in de lege verzameling gaat zoeken, het programma laten sturen dat het onmogelijk is een set te vinden.

Als we deze functies gedefinieerd hebben kunnen we het spel zelf gaan programmeren.  
De eerste stap die wij hiertoe zetten is het aanpassen van de kaarten op tafel, immers is deze list handig voor het programmeren, maar minder handig om sets in te vinden, We hebben dus tevens een functie nodig welke het bord laat zien.

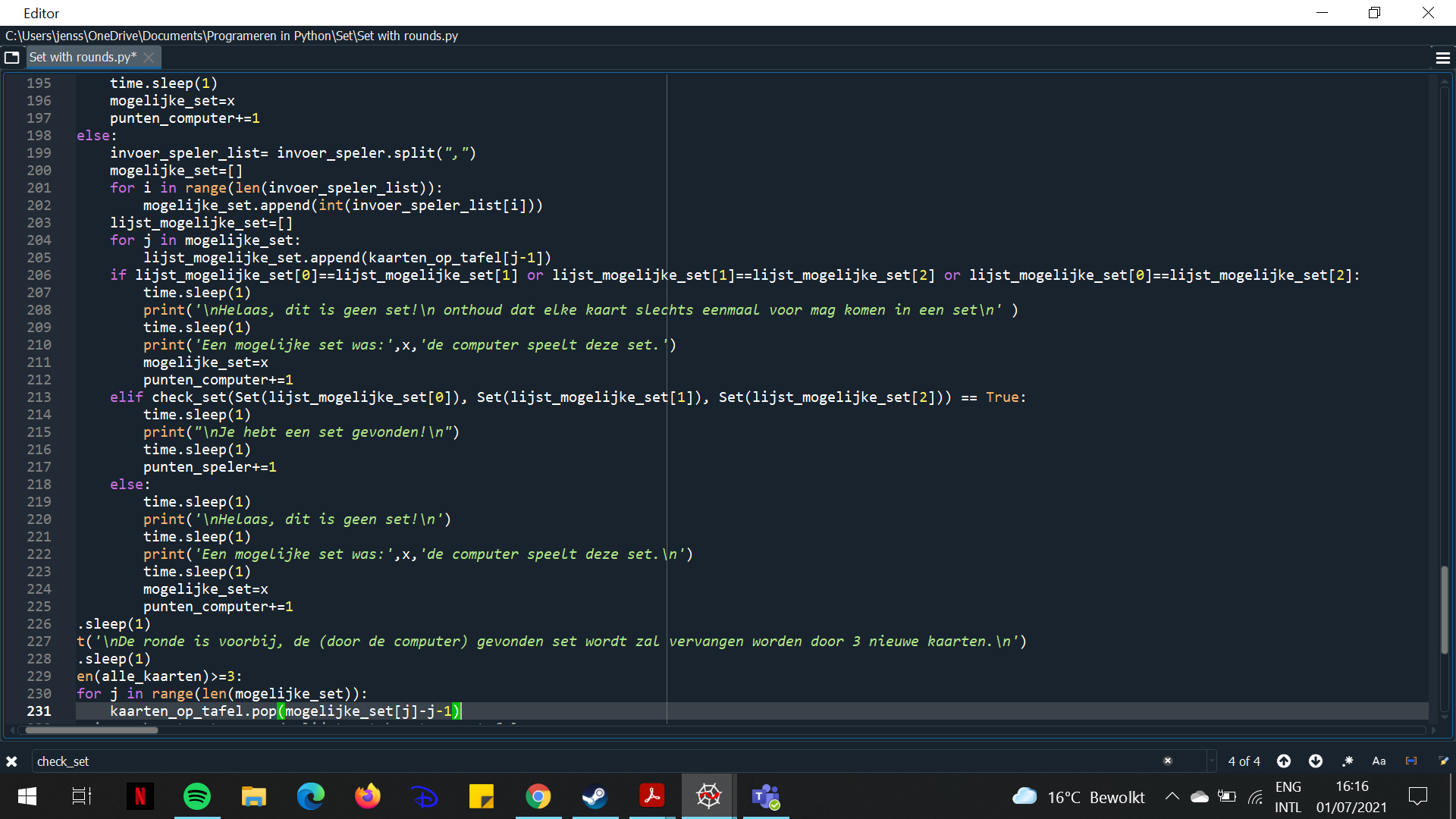
Het is je misschien al opgevallen dat de eigenschappen van de kaarten de enige code bevat welke niet in het nederlands maar in het engels geschreven is. Dit is gedaan zodat het visualiser van de plaatjes makkelijker wordt. Wij hebben een mapje met plaatjes van alle verschillende kaarten die het spel kent meegestuurd met de code. Deze kennen als naam allemaal de vier eigenschappen in het engels, gevolgd door .gif. We kunnen deze dus aanroepen door de eigenschappen achter elkaar te plakken, aangezien als je twaalf individuele fotos opent je twaalf verschillende files krijgt welke achter elkaar verschijnen. Hebben wij ertoe gekozen ze als een foto langs elkaar toe plakken, we plakken er tevens 3 extra files aan welke de juiste nummering weergeven om dit aan de gebruiker duidelijk te maken.

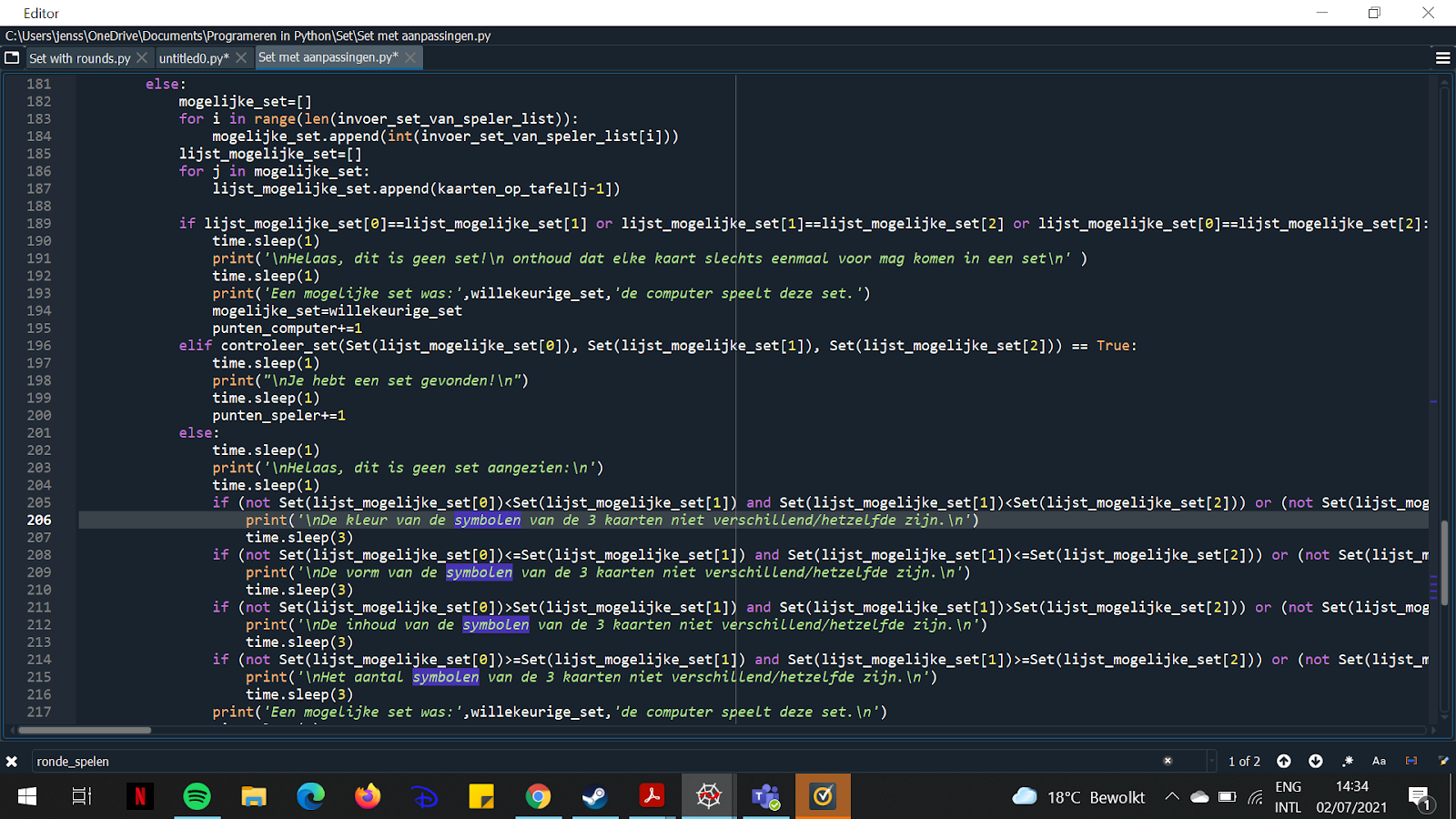
Het laatste wat nog geprogrammeerd diende te worden is het verloop van het spel. Wij hebben ervoor gekozen het grootste deel hiervan door middel van een functie te doen, immers kunnen we een programma schrijven wat elke keer een ronde speelt, met behulp van een while loop kunnen we dit vervolgens laten spelen totdat er niet meer genoeg kaarten in de pot zitten om een nieuwe ronde te starten. We laten eerst zien hoe we een ronde spelen, Er zijn twee variabelen die we willen bewaren over verschillende rondes, namelijk in welke ronde we zitten, en de tussenstand. Immers worden beide beïnvloed door de voorgaande situatie, doordat we deze als input en output van de functie hebben kunnen we deze bewaren met onderstaande functie.

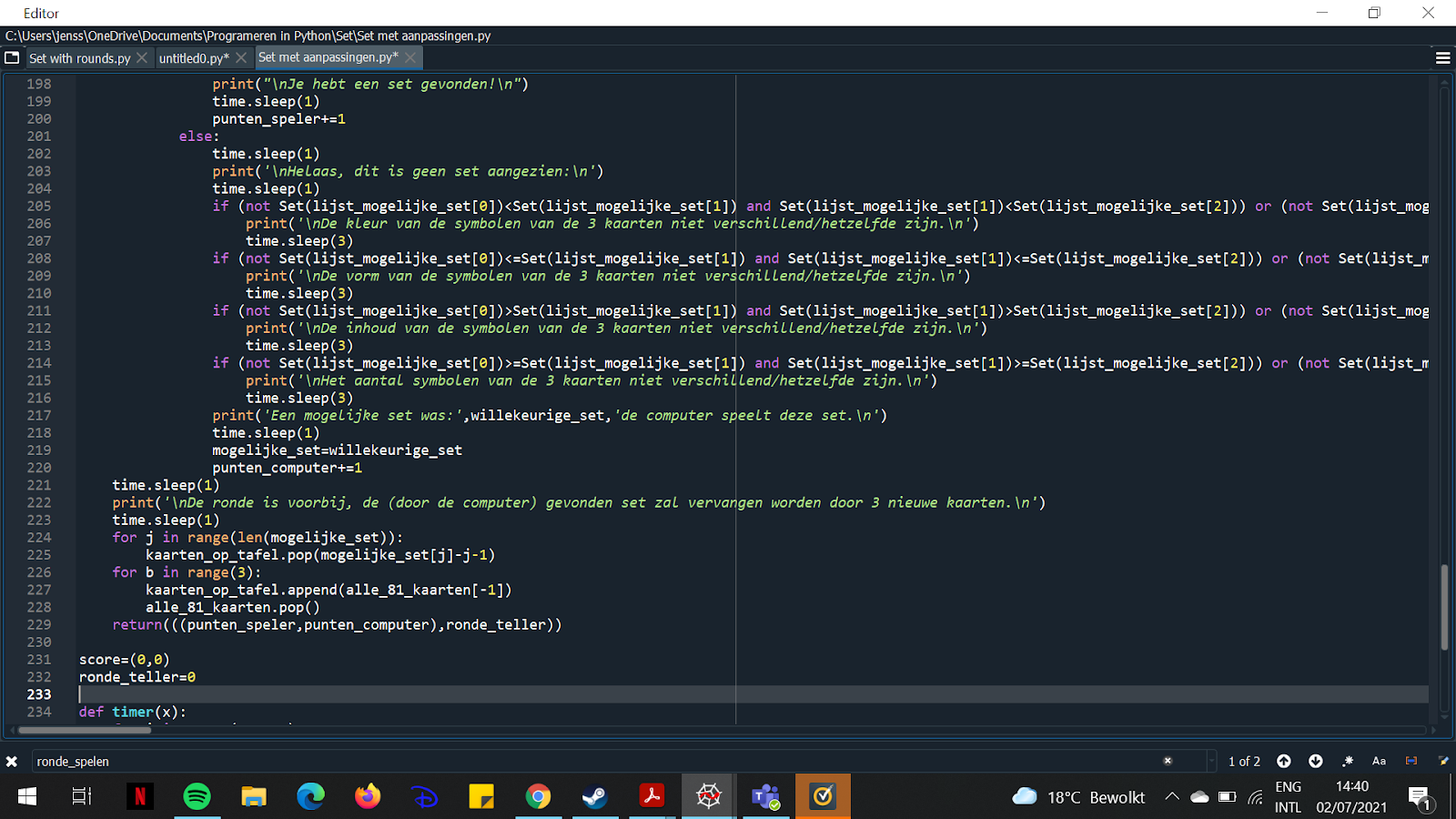
De ronde wordt enkel in het begin van de functie gebruikt om deze te weergeven, hiervoor wordt hij eerst verhoogt ten opzichte van vorige ronde. De score is een tuple en bevat twee elementen, op die manier kunnen zowel de score voor de computer als de speler bijgehouden worden, deze wordt in het begin gesplit in deze twee scores. Als bekend is of de input correct is verhoogd hij een van de twee waarden. Aangezien elke keer dat deze functie loopt, een ronde gespeeld wordt, kunnen we het hele spel spelen door deze in een while-loop te zetten. Elke ronde begint door één op te tellen bij de ronde en weer te geven aan de speler in welke ronde hij/zij zit, om vervolgens de kaarten die op dat moment in het spel zijn te laten zien.  


Het programma herkent zelf door middel van de eerder genoemde uitzondering in de vind\_alle\_sets functie of het bord een capset is. Het programma vertelt dit aan de speler zodra er een capset gevonden is. Er wordt dan een lijst aangemaakt, [1,2,3], (oftewel de eerste drie kaarten). Deze lijst zal toegeschreven worden aan de variabele genaamd mogelijke\_set. De variabele mogelijke\_set is ook de variabele waaraan de andere mogelijkheden, van of wel gevonden sets door de speler of door de computer, toegeschreven zullen worden. Deze variabele dient ervoor om aan het eind van de ronde het bord (kaarten\_op\_tafel) klaar te zetten voor de daaropvolgende ronde, door het verwijderen van de kaarten die behoren tot de nummers uit mogelijke\_set. Daarna zullen drie nieuwe kaarten toegevoegd worden aan kaarten\_op\_tafel.

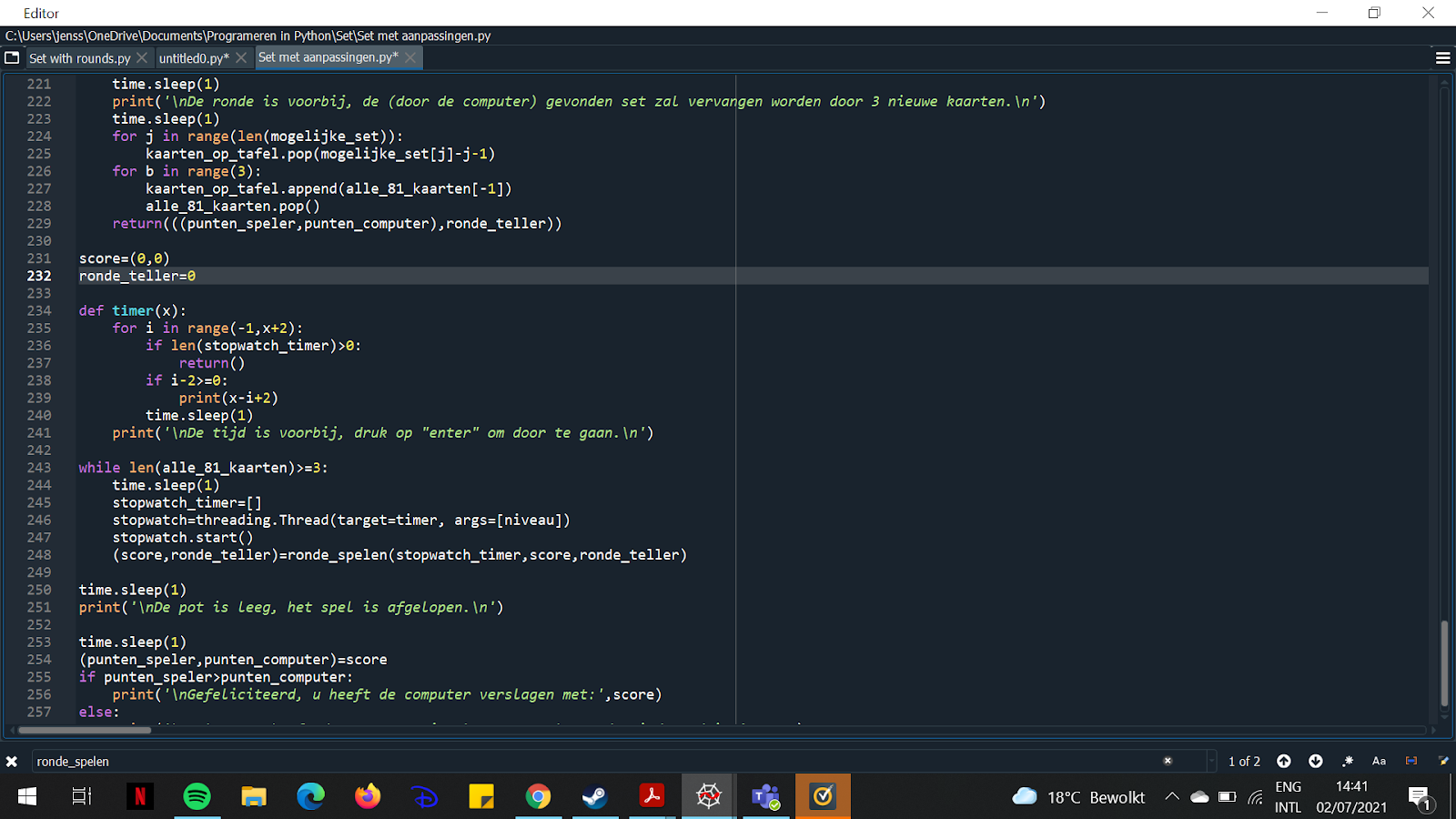
In deze code zal je misschien ook opvallen dat er gebruik wordt gemaakt van verschillende sleep functies uit de time module, wij gebruiken deze om het spel in goede banen te laten lopen. Ze hebben nauwelijks functie voor de werking van de code, en dienen er vooral toe de output van de computer te vertragen. De computer is in staat onze code zo goed als direct te doorlopen. Om de verschillende teksten die geprint wordt duidelijker te maken worden deze met een vertraging van de time.sleep() functies geprint. Met dezelfde reden worden voor en na de meeste regels tekst witregels geprint.

Als de ronde 12 kaarten op tafel heeft liggen waarmee gespeeld kan worden, zoekt de computer eerst met de eerder opgestelde functie een willekeurige code, dit is gedaan om te zorgen dat wanneer de computer een set moet vinden, deze de gehele ronde dezelfde set zal geven. Daarna wordt de gebruiker om een input gevraagd, we hebben hierin ook een uitzondering toegevoegd, namelijk als de input leeg wordt gelaten, wordt de mogelijke set verandert naar de door de computer gevonden set, hetzelfde is als de invoer foutief is, wij hebben ervoor gekozen het programma zo streng te maken, aangezien het ons niet gelukt was iets in de vorm van straftijd toe te voegen doordat de timer los staat van het spel, deze manier was ontvangt de speler alsnog een straf en bovendien wordt elke ronde die doorlopen wordt ook een echte ronde. Tot slot krijgt de computer een punt.

Aangezien de speler de gevonden set in een willekeurige volgorde in moet kunnen vullen controleren wij of dit een set is met de eerder gemaakte functie in plaats van te controleren of deze in alle sets zit, hiermeer bepalen we een winnaar.  
Als de speler wint wordt de mogelijke set niet verandert en dus wordt de door de speler gevonden set in plaats van de door de computer gevonden set aangepast, ook krijgt de speler een punt. De computer wint als de speler meerdere dezelfde kaarten invoert, of geen set gevonden heeft, met behulp van de eerder aangemaakte klassen kunnen we er bovendien achterkomen bij welke eigenschap de fout ligt en dit doorgeven.

De laatste acties die ondernomen moeten worden is het klaarzetten voor de volgende ronde, hiervoor hebben we de set bijgehouden die ‘gespeeld’ is. We schrijven dus code om deze set uit de verzameling houden, waarbij aangezien ze van klein naar groot staan en ze van links naar rechts verwijderd worden, voor ieder al verwijderde kaart, één kaart eerder verwijderd moet worden.

Het laatste element wat we willen toevoegen is een timer, wij willen dat deze een vooraf bepaalde tijd aftelt. aangezien wij willen dat deze blijft lopen als er gewacht wordt op de gebruiker hebben wij ervoor gekozen deze met behulp van de module threading te programmeren, hierbij kan code tegelijk lopen door elke keer als er een periode is waarin gewacht moet worden, zoals de tijd tussen onze timer en wachten op de gebruiker, code uit de andere thread the lezen. In de speel\_ronde functie heb je al kunnen zien dat om deze timer te stoppen enkel de term ‘stop’ aan een lijst wordt toegevoegd, hiertoe is gekozen aangezien de timer in een aparte thread zit deze niet direct beïnvloed wordt door acties van de main thread, en tevens geen input neemt aangezien hij midden in een functie zit.

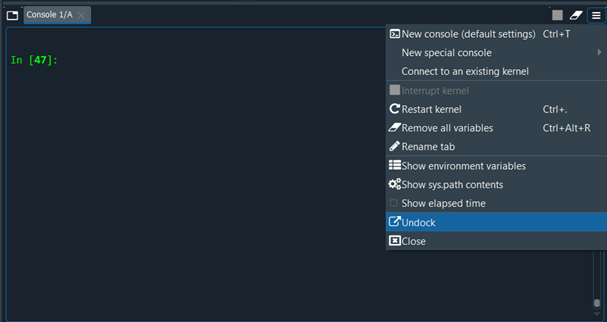
  
Aangezien lijsten universeel werken worden deze voor de hele code aangepast en kunnen we zo dus de stopconditie bij de timer krijgen. Hiervoor moeten we dus tevens in de while loop zetten dat deze lijst elke ronde leeggemaakt wordt voordat de timer gebruikt kan worden.

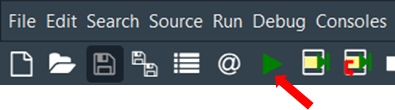
Tot slot hebben we de while loop, deze eindigt als er niet meer genoeg kaarten in de pot zitten om een nieuwe ronde te kunnen eindigen, de laatste ronde is dus gespeeld.  
Het spel eindigt met een if statement waarin de bijgehouden scores met elkaar vergeleken om te kijken wie de winnaar is. De eindstand en de uitslag van het spel worden dan gepresenteerd aan de speler m.b.v een print funtie.

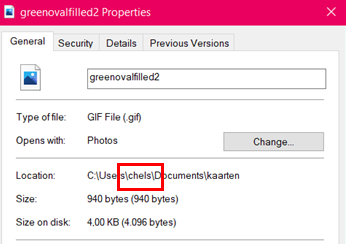
Aangezien alle processen van onze code afhankelijk zijn van relatief weinig elementen hebben maakt de complexiteit van ons programma minder uit, zo wordt om alle sets te vinden uit een verzameling kaarten, wat wij gedaan door de lijst 1 keer helemaal en 2 keer vanaf het eerste gekozen element, te doorlopen, dit geeft dus een complexiteit beter als n3, we mogen naar boven afronden, dit betekent immers dat het programma sneller is dan we claimen wat geen probleem is, Er zijn meerdere functies met deze complexiteit, zo is de vind een willekeurige set, functie hier bijvoorbeeld direct van afhankelijk, Aangezien enkel de hoogste term invloed heeft op de complexiteit is dit geen probleem. Je zou het programma kunnen stoppen als één set gevonden is, maar dit zou alleen onoverzichtelijk en onnodig zijn. De andere functies zijn ofwel lineair zoals het delen en pakken van de kaarten, of hebben een door ons bepaalde timer, wij hebben vaak de code bewust langzamer gemaakt om het spel zich op een voor mensen bij te houden snelheid af te laten spelen, zo is ook de timer een relatief langzaam programma aangezien hij pas afloopt als een vooraf bepaalde tijd voorbij is, is de gehele complexiteit van het programma is dus gelijk aan de hoogste macht, dus n3. Onze code heeft dus een relatief hoge complexiteit, toch zorgt dit niet voor problemen. Immers zijn de lijsten die altijd doorlopen worden best klein. Er wordt bijvoorbeeld voor het controleren of er sprake is van een set enkel gekeken naar 12 kaarten. Dit resulteert erin dat de snelheid van het programma altijd zo goed als direct is, dit aantal is bepaald en zal nooit veranderen als de code gelopen wordt, aangezien deze lijsten niet door de gebruiker aan te passen zijn zal de snelheid dus nooit veranderen en is heeft de code praktisch gezien een constante complexiteit.

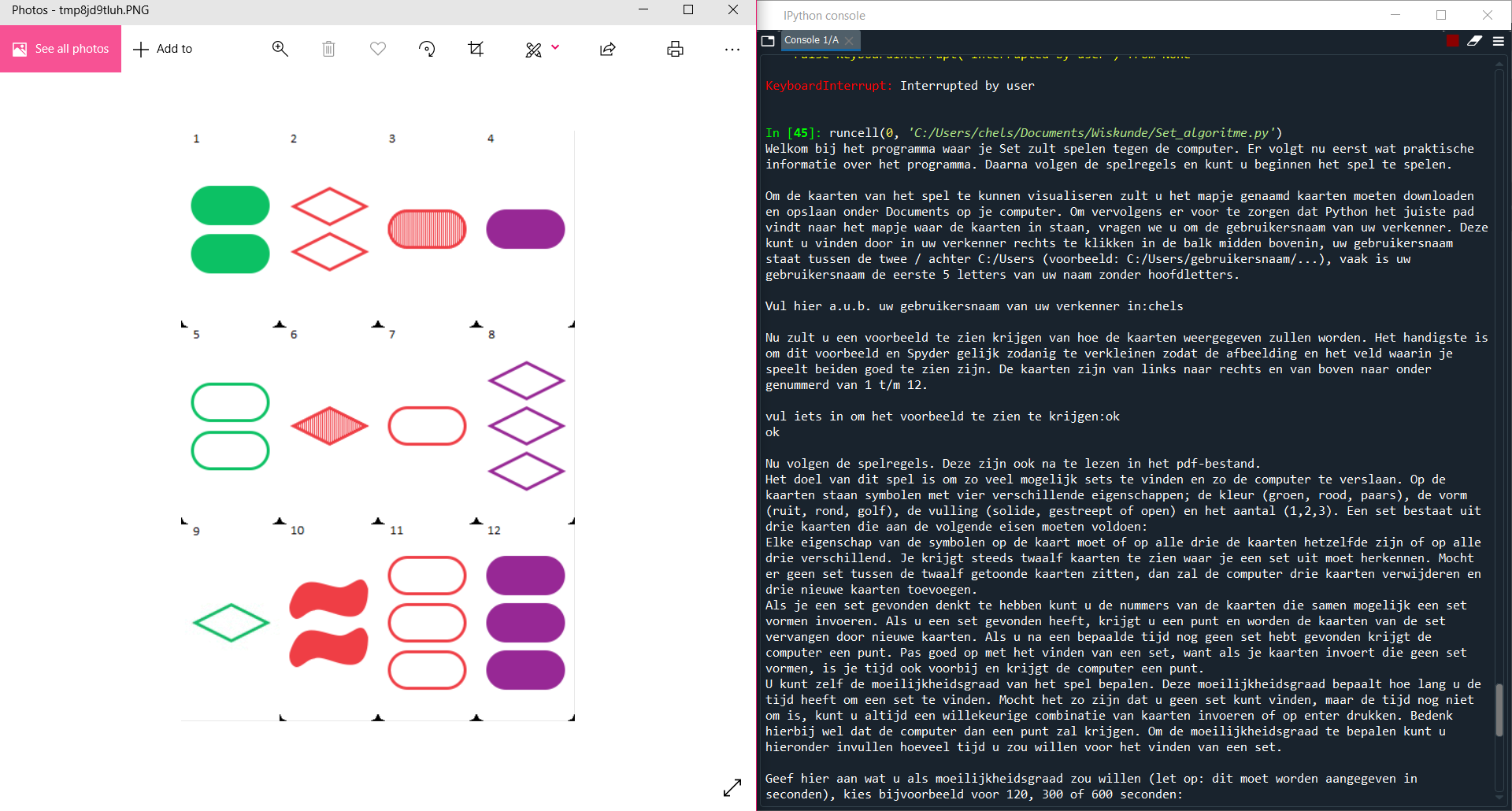
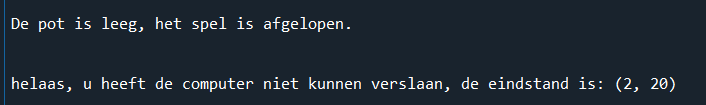
# Handleiding

* De eerste stap voor het spelen van SET tegen de computer is het downloaden van bestanden.
  + Downloaden via Github:
    - Op Github kun je de nodige code vinden en downloaden.
    - Naast de code zul je ook het mapje kaarten.zip en de afbeeldingen genaamd 1234.gif, 5678.gif en 9101112.gif moeten downloaden
    - De afbeeldingen 1234.gif, 5678.gif en 9101112.gif zul je moeten kopiëren naar het mapje ‘kaarten’ dat te vinden is in het zip-bestand.
  + Downloaden via Blackboard of via google drive:
    - Download het python programma en het mapje ‘kaarten’.
    - Controleer of de afbeeldingen genaamd 1234.gif, 5678.gif en 9101112.gif in het mapje ‘kaarten’ aanwezig zijn.
* Vervolgens moet het mapje ‘kaarten’ in de documenten map geplaatst worden van je computer.
* Zodra de bestanden gedownload zijn en op de juiste plek in je computer staan, kun je het programma openen in een python interface naar jouw keuze. Het spel is nu bijna gereed om gespeeld te worden.
* Om het spel zo optimaal mogelijk te kunnen spelen is het het handigste om de console te undocken van de rest van de python interface. Zo kun je dan steeds de console waarin je zult spelen en de afbeeldingen met de kaarten die gegenereerd worden naast elkaar op je scherm weergeven.



* Nu kun je het spel starten door op de ‘Run file’ knop te klikken of op F5 te drukken. 
* In de console zul je nu eerst wat praktische informatie over het programma te zien krijgen. Ook vraagt het programma je om je gebruikersnaam in te vullen. Dit is zodat het programma het juiste pad naar het mapje ‘kaarten’ kan vinden.
  + Je gebruikersnaam kun je bijvoorbeeld vinden door rechts te klikken op een van de afbeeldingen in het mapje ‘kaarten’ en dan naar eigenschappen of properties te gaan. Je gebruikersnaam is dan hetgeen wat in het voorbeeld in het rode vakje staat.



* Vervolgens zul je een voorbeeld te zien krijgen van hoe de 12 kaarten steeds gepresenteerd zullen worden. We raden aan het voorbeeld te verkleinen op zo’n manier dat deze naast je console scherm te zien is. Als je dit doet zul je alle andere afbeeldingen die daarna gegenereerd worden ook zo te zien krijgen. (zie figuur)
* Er wordt je ook gevraagd je eigen moeilijkheidsgraad te bepalen. Dit doe je door in te vullen hoeveel tijd (in seconden) je per ronde wil om een set te vinden.
* Na de moeilijkheidsgraad ingevuld te hebben en op enter gedrukt te hebben start het spel. Je zult de twaalf kaarten waaruit jij een set moet vinden te zien krijgen en in de console loopt een timer zodat je kunt zien hoeveel tijd je nog over hebt.
* Als je een set gevonden hebt, kun je onderin de console de nummers van de kaarten die tot de set behoren in voeren. Let er op dat je dit in het standaard format invoert, anders begrijpt het programma het niet. Het standaard format is a,b,c (waarbij je voor a, b en c de nummers van de kaarten invult).
* Als je een foutieve set invult wordt de timer gestopt en krijgt de computer een punt.
* Ook als de tijd voorbij is en je nog geen set hebt gevonden zal de computer een punt krijgen
* Het spel zal doorgaan voor 23 rondes. Zodra alle rondes gespeeld zijn krijg je te zien of je gewonnen hebt en wat de uiteindelijke puntentelling is geworden.

# Resultaten

Het doel van deze opdracht was om een programma te schrijven waarmee het spel set gespeeld kan worden en dat is ons zeker gelukt. Er zijn een aantal grote verbeterpunten aan ons programma, waar we in de discussie op in zullen gaan. Als we ons programma doorlopen is het eerste resultaat de introductie met informatie over het programma en de spelregels. Vervolgens hebben we een functie in ons programma die de kaarten, genummerd van 1 t/m 12, van ons spel visualiseert. Deze functie wordt elke keer aangeroepen als er drie kaarten verwijderd zijn na het vinden van een set en er drie nieuwe kaarten zijn toegevoegd. De speler kan dan in het plaatje met de twaalf kaarten zoeken naar een set. Het volgende resultaat dat we behaald hebben is de timer die elke ronde gaat lopen. De lengte van de timer is gebaseerd op de moeilijkheidsgraad die de speler zelf kan kiezen. Tussen de rondes door krijgt de speler informatie over de huidige situatie in het spel en aan het eind van het spel krijgt de speler te zien wat de eindstand is geworden en of hij/zij de computer heeft kunnen verslaan.

# Conclusie, discussie en taakverdeling

Het programmeren van het spel Set was een leuke uitdaging. We hadden eerst nog niet zo’n goed idee van hoe we het moesten aanpakken, maar toen we eenmaal waren begonnen en een goede start hadden gemaakt, verliep het goed. We liepen wel tegen verschillende problemen aan, waarvan de meesten goed zijn opgelost en het uiteindelijk een werkend spel is geworden. Wel zitten er nog hier en daar nog wat verbeterpunten aan ons spel, bijvoorbeeld dat als je het spel handmatig stopt, de timer af blijft tellen. Dit is natuurlijk niet de bedoeling, maar we hebben daarvoor helaas geen oplossing kunnen vinden binnen de tijd die we nog hadden om ons programma af te maken en in te leveren. Ook is het zo dat de computer een punt krijgt wanneer je een foute set invult of een foute invoer hebt, we hadden liever gehad dat de speler straftijd zou krijgen en dat de ronde niet gelijk zou stoppen. Er zijn nog meer van dat soort dingen die het spel net wat beter hadden kunnen maken, maar desalniettemin zijn we tevreden. De samenwerking ging ook erg goed, ieder van ons was bezig met een eigen taak, ook overlegden we veel en hielpen we elkaar waar nodig. Uiteindelijk heeft Jens de klasse Set, de functies controleer set, vind alle sets, vind willekeurige sets en timer geschreven. Ook heeft hij de analyse en complexiteit van de code geschreven en een laatste printcheck uitgevoerd. Chelsea heeft de beginsituatie van het spel met daarbij de kaarten op tafel functie en de gebruiksaanwijzing in het spel geschreven. Ook heeft zij de vraagstelling en de achtergrond van het probleem behandelt, de resultaten geschreven en de uiteindelijke opmaak van het verslag gedaan. Mila heeft de code waarin de 81 kaarten aan worden gemaakt en de functie visualiseren geschreven en de code leesbaar gemaakt en comments bijgevoegd. Ook heeft zij de conclusie, discussie en de taakverdeling geschreven en de grammatica gecontroleerd. Bij de timer en het visualiseren van de kaarten liepen we tegen de grootste problemen aan. De timer werkte in het begin namelijk niet goed samen met de overige code, met name het deel van de code waar de speler om input gevraagd wordt, maar dat probleem is uiteindelijk verholpen door threading te importeren, waardoor de timer goed op de achtergrond loopt. Nog niet alle problemen die in verband staan met de timer zijn opgelost, maar de grootste gelukkig wel. Met het visualiseren hadden we het probleem dat de afbeelding van de kaarten op tafel telkens zwart werd. Uiteindelijk is dit probleem opgelost door het op een andere manier aan te pakken. In plaats van de afbeeldingen naast elkaar te plakken, plakten we ze nu in een geheel nieuwe afbeelding. Op deze manier bleef de kleur wel behouden. Ook bij het visualiseren zijn er verbeterpunten, we hadden namelijk liever gewild dat de afbeelding gelijk verkleind werd en dat de afbeelding niet naar de achtergrond verdwijnt als je vervolgens weer op Spyder klikt. Nu moet de speler zelf de twee programma’s, Spyder en Foto’s, verkleinen en zo naast elkaar zetten de speler daar geen last meer van heeft. Dit probleem had misschien verholpen kunnen worden als we het visualiseren met PyGame hadden gedaan, maar we hadden helaas geen tijd meer om daar naar te kijken. Over het algemeen zijn we dus tevreden, maar het spel is nog zeker niet perfect. Wellicht als we meer tijd hadden gehad, hadden we het kunnen perfectioneren en het spel misschien zelfs in de vorm van een app kunnen presenteren, maar dat is helaas niet gelukt. Ook hadden we de code dan nog meer kunnen optimaliseren, de functie vind willekeurige set bijvoorbeeld, is nu afhankelijk van de functie vind alle sets. Als we dat onafhankelijk konden maken, zou het net wat efficiënter zijn. We vonden het een leuke opdracht en hebben er veel van geleerd.