Gedragscode

Dit practicum wordt gequoteerd, het examenreglement is dan ook van toepassing. Soms is er echter wat onduidelijkheid over wat toegestaan is, en wat niet, inzake samenwerking bij opdrachten zoals deze.
De oplossing die je voorlegt, moet volledig het resultaat zijn van het werk dat jij zelf gepresteerd hebt. Je mag je werk uiteraard bespreken met andere medestudenten, in de zin dat je praat over algemene oplossingsmethoden of algoritmen, maar de bespreking mag niet gaan over specifieke code of een specifieke oplossing die je aan het schrijven bent.
Als je het met anderen over je practicum hebt, mag dit er dus NOOIT toe leiden, dat je op om het even welk moment in het bezit bent van een geheel of gedeeltelijke kopie van het opgeloste practicum van anderen, onafhankelijk van of die code nu op papier staat of in elektronische vorm beschikbaar is, en onafhankelijk van wie die code geschreven heeft (medestudenten, eventueel uit andere studiejaren, volledige buitenstaanders, e.d.). Dit houdt tevens ook in dat er geen enkele geldige reden is om de code van je practicum door te geven aan medestudenten of te posten op welk forum dan ook.
Elke student is verantwoordelijk voor de code en het werk dat hij/zij indient. Indien je Python code verkrijgt op een dubieuze wijze en die in het practicum inbrengt, word je zelf verantwoordelijk geacht. Als tijdens de demonstratie van het practicum de assistent twijfels heeft over het feit of het practicum zelf gemaakt is (bijv. gelijkaardige code met andere studenten), zal de docent worden ingelicht en de quotering in beraad gehouden worden. Dit zal ook gebeuren als achteraf blijkt dat jouw code te hard lijkt op die van andere studenten (bv. bij automatische vergelijking). Na het horen van de studenten, en indien dit de twijfels niet wegwerkt, zal er overgegaan worden tot het melden van een onregelmatigheid (fraude), zoals voorzien in het examenreglement.

Examenpracticum Methodiek van de Informatica
1010!

Inleiding

1010! is een uitdagend puzzelspel. In dit practicum zullen jullie zelf een versie uitwerken in Python. Deze opgave beschrijft het spel in detail, maar het kan nuttig zijn om het bestaande spel eerst even online te spelen om de regels in de vingers te krijgen[1].

1010!

Het spel wordt gespeeld op een bord van 10 horizontale rijen en 10 verticale kolommen. *Kolommen* worden van links naar rechts genummerd vanaf 1; *rijen* worden van onder naar boven genummerd vanaf 1. In de code die je uitwerkt, moet het wel makkelijk zijn om over te stappen naar een andere dimensie, i.e. een bord van N rijen en N kolommen.

Bij elke zet verschijnen drie puzzelblokjes die op het bord moeten worden geplaatst. Het spel voorziet in blokjes met verschillende vormen en lengtes. Bij het plaatsen van een blokje op het bord mogen vanzelfsprekend enkel lege cellen bedekt worden. Indien door het plaatsen van een blokje rijen en/of kolommen volledig opgevuld worden, dan worden die rijen en kolommen terug helemaal leeg gemaakt. Elk vakje dat je vult met een blokje telt voor 1 punt. Wanneer je een aantal rijen en/of kolommen tegelijkertijd volmaakt krijg je 10+20+30+40+... punten extra.

Het doel van het spel is om zoveel mogelijk blokjes te verwijderen en daarbij zoveel mogelijk combo's te scoren totdat het onmogelijk is geworden om nog een blokje te plaatsen.

Het spel

Het spel wordt gespeeld op een *speelbord* dat bestaat uit 10x10 *posities* waarop *blokjes* geplaatst moeten worden. In deze sectie krijg je meer informatie over al deze concepten.

1. Posities

Elke cel op het bord heeft een positie in de vorm van een tuppel (k,r), waarbij k het nummer is van de kolom waarin de cel zich bevindt, en r het nummer van de rij.
Elke stip op een blok heeft eveneens een positie in de vorm van een tuppel (k,r), waarbij k de horizontale afstand is van de stip tot het ankerpunt van het blok, en r de verticale afstand tot dat ankerpunt.

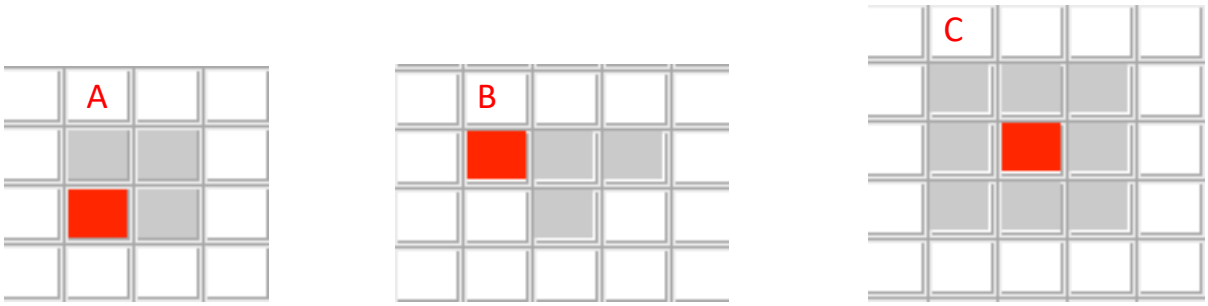
[1] https://poki.com/en/g/1010-deluxe (of zoek even op Google voor een andere versie of een Android of iOS app)

2. Blokjes

Een blok is een niet-lege verzameling van stippen (dots). Elke stip past precies op 1 cel van het spelbord. Elk blok heeft steeds precies 1 ankerpunt, dat gebruikt wordt bij het plaatsen van het blok op het bord. Het ankerpunt heeft steeds `(0,0)` als coördinaten. Het kan, maar moet niet één van de stippen van het blok zijn. Een blok waarvan het ankerpunt deel uit maakt van de stippen noemen we een *genormaliseerd blok*. In de voorbeelden in dit document (en in de *Grafische weergave*), is het ankerpunt telkens weergegeven in het rood.

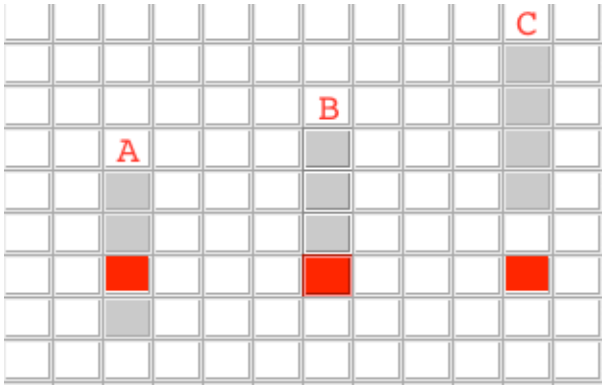
De positie van alle stippen binnen een blok zijn relatief ten opzichte van het ankerpunt. De stippen van een blok vormen een ketting ("*chain*"). Als gevolg daarvan is elke stip van een blok direct of indirect (via andere stippen) verbonden met elke andere stip. Bijvoorbeeld:

- Een blok met stippen op posities `(0,0)`, `(0,1)`, `(1,0)` en `(1,1)` is een vierkant blok. *(Blok A in* FIGUUR 1*)*
- Een blok met stippen op posities `(0,0)`, `(1,0)`, `(1,-1)`, en `(2,0)`, neemt de vorm aan van de letter T. *(Blok B in* FIGUUR 1*)*
- Een blok met stippen `(-1,-1)`, `(0,-1)`, `(1,-1)`, `(-1,0)`, `(1,0)`, `(-1,1)`, `(0,1)` en `(1,1)` omvat alle stippen die direct grenzen aan het ankerpunt. Het ankerpunt zelf is niet één van de stippen, waardoor dit een voorbeeld is van een niet-genormaliseerd blok. *(Blok C* in FIGUUR 1*)*



Figuur 1: Blok voorbeelden (Blok A – vierkant, Blok B – letter T, Blok C – niet-genormaliseerd)

Merk op dat hetzelfde blok met meerdere verzamelingen van stippen kan gedefinieerd worden. Dergelijke blokken noemen we *equivalente blokken.* FIGUUR 2 toont een aantal equivalente blokken. De rode stippen tonen het ankerpunt. A en B zijn genormaliseerde blokken (en bevatten dus zelf het ankerpunt), C is een niet-genormaliseerde blok. Alle drie de blokken zijn equivalent want ze bestaan uit 4 boven elkaar gestapelde stippen.



Figuur 2: Equivalente blokken

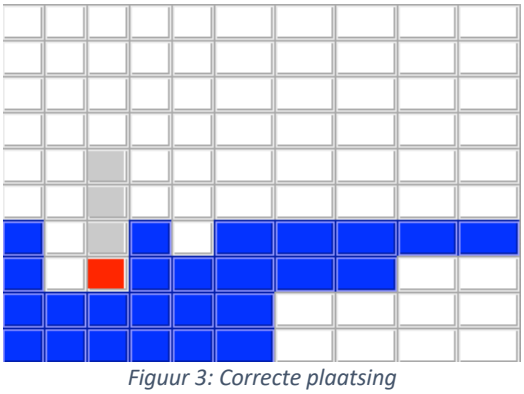
3. Het bord

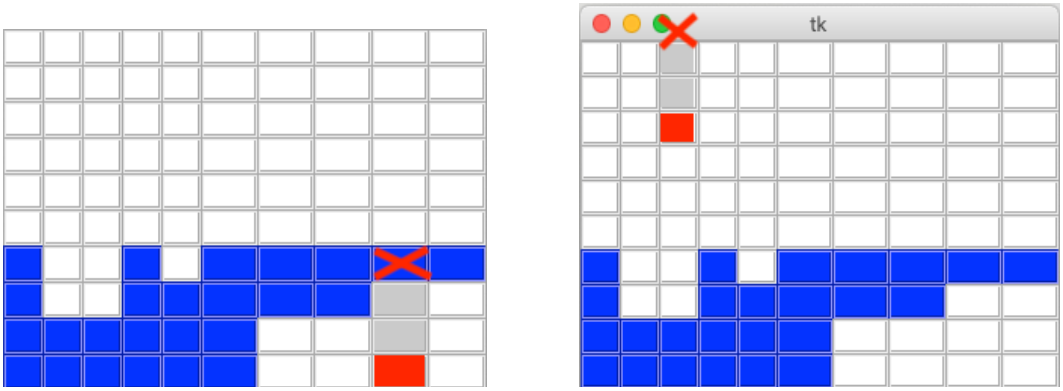
Het bord is een vierkant bord van individuele cellen. Het spel moet kunnen gespeeld worden op borden van verschillende dimensies. Het bord is verdeeld in horizontale rijen en verticale kolommen. Rijen worden van onder naar boven genummerd vanaf 1. Kolommen worden van links naar rechts genummerd vanaf 1. De cel links onderaan het bord bevindt zich dus op de kruising van de eerste rij en de eerste kolom. Elke cel heeft slechts 2 mogelijke toestanden: de cel is leeg of de cel is gevuld.

Blokken plaatsen

Cellen kunnen gevuld worden door een blok te laten vallen op het bord. Daarbij wordt de positie meegegeven voor het anker van het te plaatsen blok. De plaatsing van het blok zorgt voor het vullen van elke cel op het bord op dezelfde relatieve afstand van de positie voor het anker als een dot positie van het gegeven blok. Het blok kan enkel geplaatst worden op de gegeven positie, als al de cellen die zullen gevuld worden binnen de grenzen van het bord liggen en als ze allemaal initieel leeg zijn. Merk op dat de positie voor het ankerpunt van het te plaatsen blok niet noodzakelijk binnen de grenzen van het bord moet liggen. Dat kan met name het geval zijn voor een niet-genormaliseerd blok.

FIGUUR 3 toont een correcte plaatsing van een blok (equivalent aan de blokken in FIGUUR 2). FIGUUR 4 toont een foutieve plaatsing. In het eerste voorbeeld wordt het blokje geplaatst op reeds gevulde cellen. In het tweede voorbeeld wordt het blokje buiten de grenzen geplaatst.


Figuur 3: Correcte plaatsing



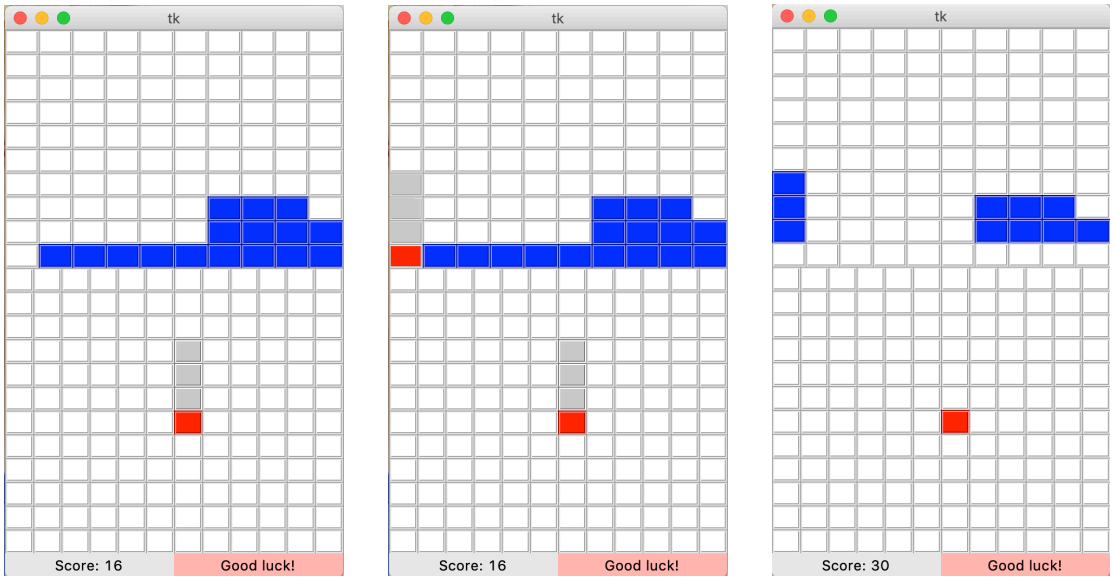
Figuur 4: Foutieve plaatsing (gevulde cellen - buiten de grenzen)

Blokken in het bord

Het laten vallen van een blok op een bord heeft enkel voor gevolg dat een aantal cellen van het betrokken bord gevuld worden. Een bord houdt geen informatie bij omtrent welke blokken allemaal gebruikt werden om haar cellen tot hiertoe te vullen. Het concept 'blok' bestaat dus enkel voor plaatsing (in de GUI betekent dit: onderaan in het venster wordt telkens een blok getoond, waarbij de rode stip het ankerpunt voorstelt. Van zodra dit blok geplaatst wordt in het bord bovenaan verdwijnt de kennis over blokken aangezien het bord enkel bijhoudt welke cellen gevuld zijn en welke cellen leeg zijn.)

Rijen en kolommen leegmaken

Wanneer een blokje geplaatst wordt dat een rij of kolom volledig opvult, wordt deze hele rij (of kolom) leeggemaakt. FIGUUR 5 toont de verschillende stappen die leiden tot het leegmaken van een rij. De middelste stap toont wat er achter de schermen gebeurt (deze stap wordt niet getoond in de GUI aangezien na het selecteren van een plaats voor het ankerpunt de volle rijen en kolommen worden leeggemaakt voordat de GUI het geüpdatet bord toont).
Merk op dat het leegmaken van rijen in dit spel wat atypisch is, in die zin dat bovenliggende stippen niet naar onder 'vallen' zoals in typische bordspelletjes. Een volle rij wordt weer volledig leeggemaakt, maar aan de andere cellen in het bord wordt niets gewijzigd.



Figuur 5: Leegmaken van een rij (voor plaatsing - tijdens plaatsing - na plaatsing)

Richtlijnen

De hoeveelheid geheugen die in beslag genomen wordt door een leeg bord moet onafhankelijk zijn van de dimensie van het bord. Daarenboven moet de functie om na te gaan of een cel op een gegeven positie op een bord al dan niet gevuld is, een (haast) constante uitvoeringstijd hebben.

Belangrijk: Beperk het aantal functies dat moet aangepast worden indien overgestapt zou worden naar een nieuwe gegevensstructuur (set, sequentie, map, ...) voor het bijhouden van de inhoud van het bord. Dat kan door zoveel mogelijk gebruik te maken van de functies uit het skelet. Zo gebruik je beter de functie `is_filled_at` als je wil weten of een cel op een gegeven positie gevuld is, dan rechtstreeks de huidige gegevensstructuur van het bord te inspecteren. Gelijkaardige opmerkingen gelden voor andere functies uit het skelet. Het verbetert daarenboven ook aanzienlijk de leesbaarheid van je code.

4. Score

Voor elke volledig gevulde rij of kolom zijn er punten te verdienen. Elke cel van het bord die je vult met een stip van een blok telt voor 1 punt. Zo heeft het eerste scherm in FIGUUR 5 16 punten aangezien het bord bestaat uit 16 stippen (en er nog geen volle rijen of kolommen zijn leeggemaakt).

Wanneer je een aantal rijen en/of kolommen tegelijkertijd vol maakt krijg je 10+20+30+40+... punten extra. Zo levert in FIGUUR 5 het blokje met 4 stippen ook 4 punten op en levert bovendien het leegmaken van de volle rij nog 10 punten extra op wat de score van 16 op 30 brengt.

Gevraagd

We vragen jou om dit spel uit te werken in Python. Om je op weg te helpen, krijg je een skelet waarin de hoofdingen van de meest essentiële functies reeds gegeven zijn. Het is de bedoeling dat je deze functies implementeert volgens de beschreven documentatie. *Indien nodig mag je de parameters van gegeven functies verder uitbreiden, op voorwaarde dat verzuimwaarden voorzien worden voor elk van die extra parameters. Je mag uiteraard ook extra hulpfuncties toevoegen als je die denkt nodig te hebben.*

De documentatie beschrijft wat er in elke functie geïmplementeerd moet worden. Je mag er vanuit gaan dat de *assumption*s steeds gelden. Je hoeft deze m.a.w. niet uitdrukkelijk te checken (maar je moet er natuurlijk wel voor zorgen dat, wanneer je deze functies zelf oproept, de parameters die je meegeeft voldoen aan de gegeven *assumptions*).

Grafische weergave

Je kan gebruik maken van de door ons aangeboden **GUI** (grafische user interface) die een grafische weergave mogelijk maakt van het spel dat jij implementeert. Deze is geïmplementeerd in GUI.py. Hierin zit eigenlijk het volledige spelverloop vervat. Jij zal zorgen voor de ondersteunende functies (zoals hieronder beschreven). In Game.py zit ook een functie play_game om het spel te spelen vanaf het toetsenbord. Merk op dat de GUI enkel gebruikt maakt van genormaliseerde blokjes en dat er slechts 1 blokje per zet getoond (en dus geplaatst) zal worden.

Basisfuncties

De basisfuncties van 1010! kan je eigenlijk structureren volgens een aantal elementaire concepten in het spel: het spel, het bord en de posities. Deze functies zijn dan ook verdeeld over respectievelijke python-bestanden (Game.py, Board.py, Block.py en Position.py).
<u>Tip:</u> Denk eraan dat je deze bestanden moet importeren vooraleer je hun functies kan oproepen in andere bestanden.

Complexere functies

In Board.py wordt verwacht dat je de functies free_all_cells en are_chainable **recursief** uitwerkt.

In Position.py zal je de functie are_chained zowel iteratief als recursief (are_chained_rec) moeten uitwerken.

In Game.py zal je de essentiële functies voor het spelverloop moeten implementeren. Hiervoor kan je uiteraard gebruik maken van de hulpfuncties die je reeds hebt gemaakt in Board.py, Block.py en Position.py. De functie game_move moet 1 stap in het spel ondersteunen, met name het laten vallen van een blokje op een welbepaalde plaats. Het eigenlijke spelen wordt, zoals reeds vermeld, aangestuurd vanuit de GUI of vanuit de functie play_game in Game.py.

Verder wordt er in Game.py ook verwacht dat je 2 algoritmes uitwerkt om de hoogste score te bepalen die je kan bekomen met een gegeven sequentie blokjes. In play_greedy zal je de hoogste score bepalen door in elke zet de 3 gegeven blokjes te plaatsen in de volgorde die op dat moment de hoogste score oplevert. In highest_score zal je de meest optimale score voor de volledige verzameling gegeven blokjes bepalen. Dit is dus niet noodzakelijk de aaneenschakeling van de hoogst mogelijke score per zet. De functie highest_score is een typisch voorbeeld van een backtracking probleem en dient verplicht **recursief** uitgewerkt te worden.

Merk op dat bepaalde (complexere) functies niet essentieel zijn voor het verloop van het spel. Deze zijn echter opgenomen in de opgave om te pijlen naar jouw inzicht en begrip van de leerstof.

Testen

Samen met het skelet voor de gevraagde functies en de GUI, krijg je alle testen die zullen gebruikt worden om de correctheid van je code te controleren. Die testen zijn verdeeld over de bestanden Game_Test.py, Board_Test.py, Block_Test.py en Position_Test.py. Alle testen kunnen worden uitgevoerd door het bestand Test_Suite.py uit te voeren. Testen voor functies die je nog niet hebt uitgewerkt, zet je best voorlopig in commentaar.
In Board.py vind je bovendien een functie om de inhoud van een bord uit te schrijven op de standaard uitvoerstroom.

Praktische informatie

Hieronder geven we alle praktische informatie i.v.m. dit examenpracticum wat betreft deadline, de uiteindelijke verdediging en quotering, enz.

Alleen of in groep?

Het practicum zal **individueel** worden gemaakt.

De verdedigingen van het practicum vinden plaats tussen 13 mei en 24 mei. Per reeks zijn er een aantal slots voorzien voor de verdediging van het practicum (zie uurrooster en toledo).
Zorg dus zeker dat je jezelf op deze momenten kan vrijmaken! Als er een gegronde reden is waarom een bepaald practicumslot in jouw uurrooster niet past (bijvoorbeeld een vak uit het 2e jaar op hetzelfde moment), laat je dit zo snel mogelijk weten aan Koen Paes (koen.paes@mirw.kuleuven.be). Het precieze moment van jouw verdediging wordt op 8 mei op Toledo beschikbaar gemaakt.

Lees ook zeker de gedragscode na op de eerste pagina van deze opgave en op Toledo. Op plagiaat staan zware sancties. Je mag over de opgave discussiëren met andere studenten, maar het is absoluut not done om code, op welke manier ook, door te geven.

Wat in te dienen?

Er wordt verwacht dat je een werkend programma schrijft. Je moet de oplossing van je practicum elektronisch indienen tegen **dinsdag 7 mei 2019 om 13u** (harde deadline; niet op tijd ingediend is geen verdediging). Stel het indienen niet uit tot de laatste minuut. Er kunnen steeds dingen foutlopen ...! Met oplossing bedoelen we alle Python-bestanden die je gemaakt hebt gebundeld in 1 zip-bestand (zelfs al heb je maar 1 python-bestand). Bekijk hiervoor de instructies op Toledo en volg deze strikt, zo niet zal je niet gepland worden voor ondervraging.

Verwachte inspanning

We verwachten dat je 25 uur nodig hebt om dit practicum tot een goed einde te brengen. In die schatting gaan we uit van een perfecte kennis van het cursusmateriaal. Het is met andere woorden de tijd die een ervaren Python programmeur nodig zou hebben om deze opgave uit te werken.

Waar word je op beoordeeld?

Je code zal door ons gedraaid worden tegen de verzameling testen die je van ons gekregen hebt. Naast functionaliteit, zal je oplossing ook (en vooral) worden beoordeeld op leesbaarheid, structuur, aanpasbaarheid (zie de richtlijnen omtrent het gebruik van functies versus het direct manipuleren van de gegevensstructuur), …

Wat te doen bij onduidelijkheden?

Je kan steeds uitleg vragen over de opgave via het forum op Toledo. Indien je vragen stelt op Toledo word je verondersteld te controleren of de vraag nog niet aan bod kwam. Geef daarom een zinnig onderwerp aan je bericht.

Wat wel en niet gebruiken?

Dit practicum zal toetsen of je de leerstof tot en met module 14 beheerst. Alles wat in latere modules aan bod komt (o.a. functioneel programmeren en objectgericht programmeren) mag niet gebruikt worden. Je mag ook geen gebruik maken van list comprehension of set comprehension, behalve bij het initialiseren van gegevensstructuren, zoals dat ook in de cursus gebeurde.

Het practicum moet uitgewerkt worden in versie 3.6 van Python. Je mag gebruik maken van de standaard Python modules, de Python-bestanden die door ons gegeven worden en natuurlijk de zelf geschreven bestanden. Het gebruik van andere modules en/of packages is niet toegelaten. Bij twijfel, kan je het forum van Toledo raadplegen.

Heel veel succes!
het MI-team