PO PYTHON PROGRAMMEREN: GANZENBORD

BOUW JE EIGEN BORDSPEL MET PYGAME

V5 periode 8

Belangrijke concepten: lijsten, loops, functies

Dit lesmateriaal is gebaseerd op werk van J. van Weert, 2017

Inhoudsopgave

1 INLEIDING

Een spel spelen is leuk, maar zelf een spel maken is nog een stuk leuker natuurlijk. Nu knutsel je niet op een vrijdagmiddag zomaar Minecraft in elkaar, maar je zult zien dat je in een paar lessen al een aardige game kunt bouwen, die je vervolgens nog flink kunt uitbreiden en waar je een hoop creativiteit in kwijt kunt. Onderweg leer je nog allerlei toffe dingen over programmeren ook.
Ideaal!

We hebben de afgelopen tijd van alles geleerd over Python en programmeren. Met behulp van opgaven heb je de belangrijkste programmeerconcepten zoals variabelen, loops, lijsten en allerlei handige constructies geleerd. Maar tot nu toe heb je ze steeds in kleine oefeningen en voorbeelden toegepast.

In deze PO ga je je Python skills gebruiken om eens een wat groter project op te zetten. Alles wat je tot nu toe hebt geleerd is bruikbaar en ook nodig, maar doordat het een groter programma wordt, is structuur en logische (stap voor stap) opbouw extra belangrijk.

In dit document vind je een uitgebreide uitleg hoe je de basis van een eenvoudig Ganzenbord spel kunt maken in Python met behulp van de PyGame library. Het is vervolgens aan jou om bovenop deze basis een uitgebreider (bord)spel te bouwen. Of je nu het originele ganzenbord zo goed mogelijk probeert na te maken of een volledig eigen thema en spelconcept gaat bedenken, dat is helemaal aan jou. Je wordt aangemoedigd er een tof eigen project ervan te maken.

Veel plezier!

2 OPZET, BEOORDELING EN INLEVEREN/DEADLINE

2.1 OPZET

Op de volgende pagina's staat uitleg over het opbouwen van de basiscode om een werkend bordspel te maken. Mat het volgen van de instructies uit dit document behaal je 1 punt voor deze PO. De hogere cijfers zitten in je eigen uitbreidingen, elegante code, verbeteringen en toevoegingen. Met deze uitbreidingen kun je alle kanten op (benader het originele ganzenbord, maak een ander bordspel na of verzin je hele eigen variant op het ganzenbord met een eigen thema en regels, of nog iets heel anders...)

2.2 INLEVEREN

Lever in op **Montiplaza** in een **.zip** bestand voor 14-06 om 23:59:

1. Je **code** voorzien van **commentaar**. Voeg ook gebruikte afbeeldingen en eventueel andere externe bestanden (audio, etc..) toe. Als je code van iemand anders gebruikt moet bronvermelding in je commentaar zetten, geef daarbij duidelijk aan om welk deel van de code het gaat.
2. Verslag met:
   - **a.** **Logboek**: Geef per keer aan, hoe lang heb je er aan gewerkt, waar heb je toen aan gewerkt.
   - **b.** Korte uitleg van de **uitbreiding**/regels van je spel: omschrijf welke functionaliteit je zelf hebt toegevoegd.
   - **c.** **Algoritme**: Selecteer een stukje programma code dat een algoritme implementeert en dat van fundamenteel belang is voor je programma om het beoogde doel te bereiken . Dit moet een codesegment zijn die je zelf ontwikkeld hebt, moet twee of meer deelalgoritmen bevatten en moeten wiskundige en/of logische concepten integreren. Beschrijf hoe elk deelalgoritme binnen de geselecteerde algoritme onafhankelijk van elkaar werkt, en hoe ze in combinatie met anderen, een nieuw algoritme vormen dat helpt om het beoogde doel van de programmering te bereiken. Plak de code (of een screenshot) in je verslag en geef met een **ovaal** aan waar je het precies over hebt.
   - **d.** **Abstractie**: Selecteer een stukje programma code dat je zelf hebt gemaakt dat gebruik maakt van een abstractie (bv. een methode of datastructuur). Deze abstractie moet wiskundige en logische concepten integreren. Leg uit hoe je abstractie heeft bijgedragen aan het beheren van de complexiteit van jouw programma. Plak de code in je verslag en geef met een **rechthoek** aan waar je het precies over hebt.
   - **e.** **Reflectie**: blik terug op een bepaald moment in het ontwikkeltraject. Beschrijf de moeilijkheden en/of kansen die je bent tegengekomen en hoe ze werden opgelost of hoe je daar gebruik van hebt kunnen maken.

2.3 BEOORDELING

Dit is een individuele opdracht, dus iedereen bouwt zijn eigen bordspel en levert hun eigen verslag in.

Je **proces** wordt beoordeeld op (zie ook kopje 'Inleveren'):	
Voortgang *Logboek* *Reflectie* *(20%)*	Je werkt zelfstandig en lost zelf fouten in de code op. Je logboek omschrijft hoe lang je waaraan gewerkt hebt. Je reflectie omschrijft de leereffect van moeilijkheden/kansen die je bent tegengekomen.
Je **product** wordt beoordeeld op (zie ook kopje 'Inleveren'):	
Uitbreidingen *Correctheid* *Stijl* *Constructie* *(65%)*	Hoe verder je komt qua functionaliteit (uitbreidingen, originele invulling), hoe hoger je cijfer. Alle code werkt precies zoals verwacht. Code is makkelijk te lezen en begrijpen, voorzien van uitstekende commentaar en logische naamgeving (conventies). De kwaliteit van de code: het is duidelijk, efficiënt, elegant, logisch en goed gestructureerd. Er wordt gebruik gemaakt van loops, condities, functies en parameters. Gebruik van 'harde' waardes en globale variabelen worden zo veel mogelijk vermeden.
Algoritme en Abstractie *(15%)*	Omschrijving van zowel een eigen ontwikkelde algoritme en abstractie.

3 TOOLS

3.1 PYCHARM

Voor de PO maak je gebruik van Pycharm. Als je wat uitgebreidere code gaat schrijven zoals nu, is een uitgebreidere editor ook wel handig. Er zijn er veel verkrijgbaar, maar voor deze opdracht gaan we werken met PyCharm.

3.2 PYGAME

Een van de handige dingen aan Python is dat er ontzettend veel libraries zijn die een hoop functionaliteit aan Python toevoegen. Voor het maken van (2D) games is de library PyGame beschikbaar. Tot nu toe hebben onze Python scripts alleen maar tekst-uitvoer gegenereerd. PyGame maakt het mogelijk om met graphics te werken en om gebruikersinvoer zoals muisklikken en toetsaanslagen te detecteren en te verwerken. Als je een school PC gebruikt, dan zit de PyGame library er al voor je in.

Om PyGame in Pycharm werkend te krijgen moet je de volgende stappen volgen:

1. Start Pycharm op
2. Ga in het menu naar Settings -> Project -> Project Interpreter
3. Klik rechts op de +
4. Zoek op Pygame
5. Kies install package

Alle documentatie van PyGame (handig dingen opzoeken): https://www.pygame.org/docs/

Meer info over PyGame vind je hier: http://pygame.org/

4 HOE WERKT PYGAME?

Voordat we in de code duiken, is het handig om globaal wat te vertellen over de opzet van een spel in PyGame. De opzet kun je als volgt samenvatten (op volgorde zoals ze in de code staan):

- Alle imports van de libraries die we gaan gebruiken (o.a. pygame zelf)
- Een lijst met variabelen die de toestand van je spel bijhouden zoals:
    - Speler posities
    - Laatste dobbelsteenworp
    - De coördinaten van alle vakjes
    - etc..
- PyGame initialisatie. Het gereedmaken van het spel voordat het begint:
    - Opstarten van de PyGame bibliotheek
    - De afmetingen en eigenschappen van het spelscherm opgeven
    - Nog wat andere PyGame administratie
- Een loop die voortdurend het volgende uitvoert:
    - Checken of er input is (toetsaanslagen, muisklikken, etc.)
        - Zo ja: werk de toestand van de spelvariabelen bij o.b.v. de input bijvoorbeeld het verplaatsen van een pion
    - Het bijwerken van het beeldscherm

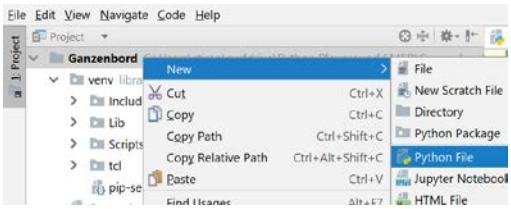
De loop zorgt ervoor dat python steeds naar invoer blijft luisteren en het spel (en het scherm) aanpast als er iets gebeurt. Op het kruisje van het spelscherm klikken zorgt voor beëindiging van de loop en het netjes afsluiten van PyGame.

5 TUTORIAL: DE BASIS VAN EEN BORD SPEL IN PYTHON MET PYGAME

In de volgende paragrafen wordt je stap voor stap geholpen met de basis van je game. Er wordt expres screenshots in plaats van rechtstreekse code gebruikt. Je zult de code dus zelf moeten invoeren. Dit is niet om je te pesten, maar daardoor heb je een veel beter inzicht in je code als je aan je eigen uitbreidingen begint. Wees ook niet zuinig met (extra) commentaar in je code.

5.1 OPZETTEN OMGEVING

1. Maak een mapje 'Ganzenbord' in je **Montessori OneDrive** aan waar je gaat werken. Hier zet je al je bestanden. Zo kun je zowel vanuit thuis als op school bij, en even belangrijk, als je laptop kapot valt dan heb je je werk nog.
2. Start PyCharm op. Kies voor New Project en blader naar het mapje 'Ganzenbord' in je Montessori OneDrive dat je net hebt aangemaakt. Dit kan eventjes duren.
3. Maak een Python bestand aan voor je code. Met je rechtermuisknop klik je op linksboven op je 'Ganzenbord' map en kies voor een New -> Python File. Noem je bestand Ganzenbord.



4. Als je op een eigen PC werkt moet je de Pygame module importeren in PyCharm. Volg daarvoor de volgende stappen (en vraag hulp als het niet lukt):
   a) Open PyCharm
   b) Ga naar Settings -> Project -> project interpreter -> +
   c) Zoek op Pygame
   d) Kies 'install package'

5.2 CODE INDELING EN IMPORTS

Zoals je in hoofdstuk 4 hebt kunnen lezen, zal onze code een specifieke structuur aannemen. We beginnen met het duidelijk aangeven van deze structuur in onze code. Hierdoor blijft je code mooi overzichtelijk.

```
1  # Vul hier je naam en de naam van je spel in
2  import pygame, random
3
4
5  # -------- Globale variabelen --------
6
7
8
9  # -------- Pygame initialisatie ---------
10
11
12
13 # -------- Hoofdloop van het programma -----------
14
15
16
17 # --------- Afsluiting ---------
18 pygame.quit()
```

- Op regel 2 importeren we de pygame library en de random library (Die hebben we straks nodig om random getallen te maken voor onze dobbelsteen)
- Op regel 18 sluiten we alle pygame dingen netjes af met het commando pygame.quit()
- De groene strepen in commentaar maken de verschillende secties in de code duidelijk. Dit is gewoon handig om het overzicht te bewaren.

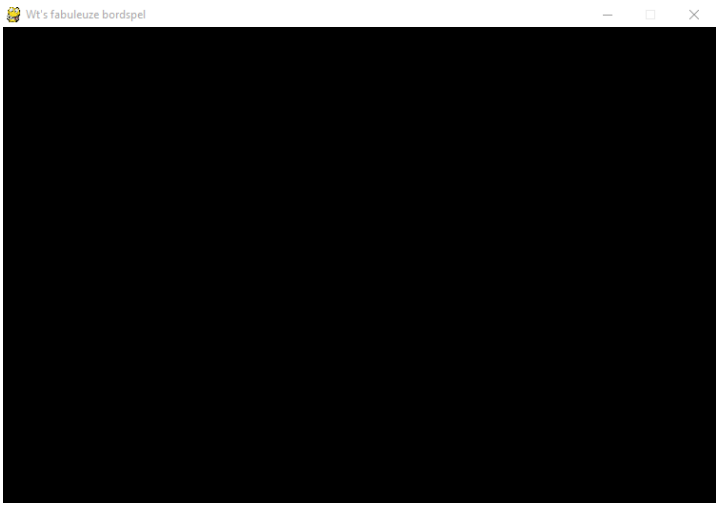
5.3 PYGAME INITIALISATIE

Als je met pygame werkt moeten er een aantal dingen worden “klaargezet”. Dit doen we onder het kopje “Python initialisatie”. Neem de volgende code onder dat blokje over. Het commentaar legt uit wat de regels doen:

```
9  # -------- Pygame initialisatie ---------
10
11 # Pygame initialiseren (is altijd nodig bij begin van gebruik pygame)
12 pygame.init()
13
14 # Afmetingen van het spelscherm instellen (in pixels [breedte, hoogte])
15 # En het spelscherm maken (en opslaan in een variabele screen)
16 WINDOW_SIZE = [1050, 750]
17 screen = pygame.display.set_mode(WINDOW_SIZE)
18
19 # Titel van spelscherm instellen:
20 pygame.display.set_caption("Wt's fabuleuze bordspel")
21
22 # Deze boolean laat ons spel straks in een oneindige loop lopen, totdat er op
23 # het kruisje wordt geklikt om af te sluiten (deze zet dan done op True)
24 done = False
25
26 # We maken een pygame Clock object. Deze is nodig om de verversingssnelheid
27 # (framerate) van het scherm te beheren
28 clock = pygame.time.Clock()
29
30
31
32 # -------- Hoofdloop van het programma -----------
33
```

Je kunt je programma om te testen vast eens runnen (Druk op de groene play knop bovenaan PyCharm of druk op Ctrl+Shift+F10).

Als het goed is zie je dan een scherm voorbij flitsen (hij sluit wel meteen af, dat fixen we verderop) met de opgegeven afmetingen en een titel:



5.4 OPZET VAN DE HOOFDLOOP

We gaan nu de basis van de oneindige hoofdloop van het programma invoeren. Omdat deze eigenlijk in drie delen is in te delen, geven we dat ook aan met commentaarlijnen. Neem de volgende code over:

```
32 # -------- Hoofdloop van het programma -----------
33
34 while not done:
35
36
37     # --- Check gebeurtenissen (zoals muiskliks e.d.) ---
38
39
40     # --- Teken de graphics (nog buiten beeld) ---
41
42
43     # --- Ververs het beeldscherm met de nieuwe graphics ---
44
45     clock.tick(60) # Zet de limiet op 60 frames per seconde en
46     pygame.display.flip() # ververs het beeldscherm met de bijgewerkte versie
47
48
49
50 # --------- Afsluiting ---------
51 pygame.quit()
```

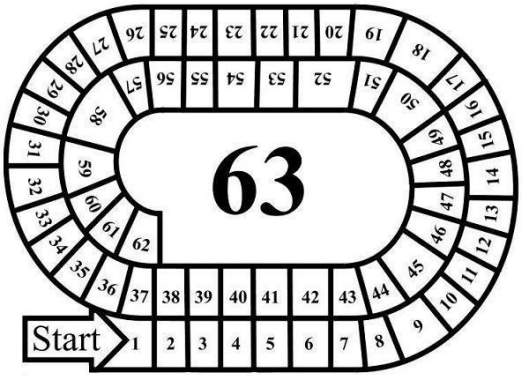
- De `while not done:` zorgt feitelijk voor de oneindige loop. Zolang de variabele `done` op `False` blijft staan, blijf de loop herhalen. (we voegen later code toe om hem met het afsluiten van het spel op `True` te zetten)

- Elke stap van de loop bestaat uit 3 fasen:
    - Luister naar invoer (toetsenbord, muiskliks, etc.). Op basis van gedetecteerde invoer wordt het spel bediend. We koppelen later stukken code aan verschillende inputs.
    - Teken een nieuwe versie van het spelscherm. Als er invoer is verwerkt is waarschijnlijk de toestand van je spel veranderd (pionnen zijn verzet, dobbelstenen gegooid, etc.). Er moet dus een nieuwe versie van het scherm worden "klaargezet"
    - Als het nieuwe scherm klaar is, zet het daadwerkelijk ook op het beeldscherm. Dat noem je "flippen". Dat wil zeggen dat het huidige scherm wordt gewist en opnieuw wordt gevuld met alles wat we in de vorige stap hebben getekend.

- De `clock.tick(60)` op regel 45 zorgt ervoor dat deze loop maximaal 60 keer per seconde wordt uitgevoerd. Hierdoor is er een maximale framerate van 60 (beeldjes per seconde). Dit is meer dan genoeg om het spel vloeiend te laten lopen en bespaart rekenkracht, omdat vaker verversen toch niet zichtbaar is.

5.5 HET BORD TEKENEN

We gaan het bord op het scherm tekenen. Hiervoor heb je natuurlijk een plaatje van een spelbord nodig. Het is het leukste om je eigen bord te ontwerpen (en deze af te stemmen op je thema en spelregels). Hier is een simpele bord dat je goed als basis kunt gebruiken (in Paint of Paint.net kun je er natuurlijk een mooi "aangekleed" bord van maken), maar je mag ook een totaal eigen bord maken natuurlijk.



Zet het plaatje van je bord in dezelfde map als waar je python script is opgeslagen. Onder het stuk " Globale variabelen" zet je vervolgens het volgende neer om het plaatje in te laden:

```
5 # -------- Globale variabelen --------
6
7 # bord afbeelding:
8 bord = pygame.image.load("bord.jpg")
9
```

Hierbij is `bord.jpg` de naam van je afbeelding.

Onder het kopje “Teken de graphics” in de hoofdloop zet je het volgende neer om het plaatje daadwerkelijk op het scherm te tekenen:

```
42      # --- Teken de graphics (nog buiten beeld) ---
43
44      screen.fill((255,255,255)) # begin met een witte achtergrond
45
46      bordrect = bord.get_rect() # vraag afmetingen (rectangle) van het bordplaatje op
47      screen.blit(bord, bordrect) # teken het bord
48
```

5.6 HET SPEL NETJES SLUITEN

We willen dat het spel netjes afsluit als we op het “kruisje” klikken. Het klikken op het kruisje is een “gebeurtenis” (event). Die moeten we detecteren.

Onder het kopje “Check gebeurtenissen” in de hoofdloop zet je het volgende:

```
39      # --- Check gebeurtenissen (zoals muiskliks e.d.) ---
40
41      for event in pygame.event.get():
42          if event.type == pygame.QUIT:  # Het kruisje is aangeklikt
43              done = True  # Het spel moet eindigen dus we zetten done op True
44
45
46      # --- Teken de graphics (nog buiten beeld) ---
```

Alle toetsaanslagen en muiskliks en andere events worden door pygame geregistreerd en aan een lijst met events toegevoegd. Moet deze `for`-loop doorlopen we deze lijst en checken of er een zogenaamd `QUIT`-event in voorkomt (met de `if` op regel 42). Als dat zo is, zetten we de `done` boolean op True. Hiermee laten we de oneindige hoofdloop toch eindigen.

Later gaan we nog meer events detecteren en zullen we deze code nog uitbreiden.

Test nu je spel nog eens. Je bord zou zichtbaar moeten zijn en het boeltje sluit netjes af als je op het kruisje klikt.

5.7 COÖRDINATEN VAN DE VAKJES OPSLAAN

Nu komt er een vervelend, maar noodzakelijk klusje...

Omdat we straks pionnen willen gaan tekenen en deze pionnen alle vakjes van ons bord moeten kunnen doorlopen, zullen we alle locaties (coördinaten) van de vakjes op het scherm moeten opslaan in ons programma. Als we dan weten op welk vakjesnummer een pion staat, kunnen opzoeken wat de coördinaten van dat vakje zijn en zo de pion op de juiste plek op het scherm tekenen.

Omdat alle vakjes een x-coördinaat en een y-coördinaat hebben, moeten we dus van alle vakjes die 2 waarden (het zijn simpelweg gehele getallen) opslaan.

Het handigste is om dat in een python list te doen. Beter nog is om elk coördinaat als een list van 2 getallen op te slaan al deze coördinaten samen

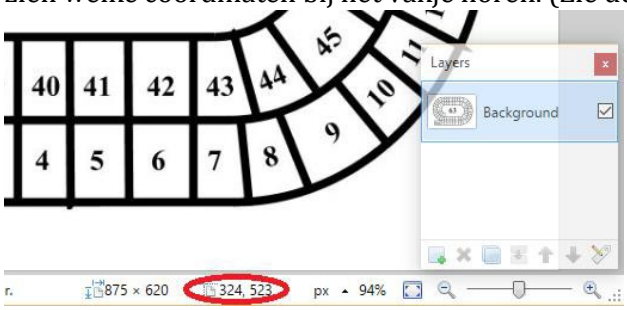
weer in een list te stoppen. Je krijgt dan een list met lists. Hiervoor voeg je onder het kopje “Globale variabelen” in je code iets van deze strekking in:

```
5  # -------- Globale variabelen --------
6
7  # bord afbeelding:
8  bord = pygame.image.load("bord.jpg")
9
10 #coordinaten van de vakjes:
11 vakjes = [[80, 385], [152, 385], [192, 385]]
12
```

Hier maken we een list met de naam `vakjes`. Deze list bestaat (in ons voorbeeld) weer uit 3 lists van elk 2 getallen. Elk van deze sublists zijn de x- en y- coördinaten van een vakje. (dus vakje 0 heeft x-coördinaat 80 en y-coördinaat 385, etc.)
Jouw bord zal uit meer dan 3 vakjes bestaan. Je moet voor elk vakje de coördinaten aan de list toevoegen. Dat is even een klusje, maar daar ontkomen we niet aan.

Als je het standaard bord gebruikt, kan ik je wat werk besparen. Hier is een lijst met de coördinaten ervan.

Als je een eigen bord gebruikt, is het handigst om je bord even in Paint (of Paint.net) te openen. Je kunt dan je muis over een vakje houden en rechtsonder zien welke coördinaten bij het vakje horen. (Zie de screenshot hieronder).



Het handigste is om met je muis een beetje linksboven in het vakje te gaan staan. Het punt wat je aanwijst geeft de linkerbovenhoek van de pion aan straks. Slim is om ze vanuit Paint even op een briefje te schrijven. Als je lijstje klaar is, typ je ze over in je list in Python.

5.8 POSITIES VAN DE PIONNEN

Nu we weten wat de coördinaten van alle vakjes zijn, kunnen we de pionnen gaan tekenen. We gaan hierbij uit van 2 spelers, je kunt dit later uitbreiden natuurlijk...

Om de pionnen te kunnen tekenen (en later ook te kunnen verzetten) moeten we opslaan op welk vakje ze staan. Hierbij slaan we het vakjesnummer op en niet de coördinaten. Als we het vakjesnummer weten, kunnen we namelijk in onze lijst met coördinaten opzoeken waar dat vakje op het bord is en onze pion erop tekenen.

Onder het kopje “Globale variabelen” maken we een lijst met speler posities. Deze zetten we om te beginnen voor beide spelers op 0. 0 is namelijk ons startvakje:

```
5  # -------- Globale variabelen --------
6
7  # bord afbeelding:
8  bord = pygame.image.load("bord.jpg")
9
10 #coordinaten van de vakjes:
11 vakjes = [[80, 385], [152, 385], [192, 385]]
12
13 # pion posities
14 posities = [0,0]
```

Hierbij is het eerste getal in de lijst de positie van speler 1 en het tweede getal is de positie van speler 2. Met de code `posities[0]`kunnen we nu de positie van speler 1 opvragen en met `posities[1]` de positie van speler 2.

We gaan de pionnen tekenen als gekleurde cirkels. (Je kunt dit later vervangen door een afbeelding of een complexere vorm als je wilt natuurlijk).

Hiervoor voeg je de volgende code toe aan het kopje “Teken de graphics” in de hoofdloop:

```
52     # --- Teken de graphics (nog buiten beeld) ---
53
54     screen.fill((255,255,255)) # begin met een witte achtergrond
55
56     bordrect = bord.get_rect() # vraag afmetingen (rectangle) van het bordplaatje op
57     screen.blit(bord, bordrect) # teken het bord
58
59     #teken pionnen als gekleurde cirkels op de coordinaten van de vakjes waar ze staan:
60     speler1_x = vakjes[posities[0]][0]
61     speler1_y = vakjes[posities[0]][1]
62     speler1_kleur = (0, 255, 0) #groen
63     pygame.draw.circle(screen, speler1_kleur, (speler1_x, speler1_y), 10)
64
65     speler2_x = vakjes[posities[1]][0] + 3
66     speler2_y = vakjes[posities[1]][1] + 3
67     speler2_kleur = (0, 0, 255) #blauw
68     pygame.draw.circle(screen, speler2_kleur, (speler2_x, speler2_y), 10)
69
70
71     # --- Ververs het beeldscherm met de nieuwe graphics ---
```

Dit is de meest ingewikkelde code tot nu toe, maar met wat uitleg is hij prima te begrijpen. We doen eigenlijk 2 keer hetzelfde (namelijk voor beide pionnen):

- Regel 60: We willen het x-coördinaat opzoeken van het vakje van speler 1. Hiervoor moeten we eerst weten op welk vakje speler 1 staat, dus dat zoeken we op met `posities[0]`.
  Het getal wat daar uit komt vullen we in als index voor onze lijst met vakjes-coördinaten.
  Dus daardoor wordt het `vakjes[posities[0]]`.
  We hebben nu de coördinaten van het juiste vakje opgezocht. Omdat we alleen het x- coördinaat van het vakje zoeken, moeten we het eerste van de twee getallen uit het coördinaten halen. Daarom is de hele code: `vakjes[posities[0]][0]`
- Regel 61: Hetzelfde als regel 60, maar dan zoeken we het y-coördinaat, dus moeten we het tweede getal uit de lijst hebben: `vakjes[posities[0]][1]`
- Regel 62: De pion moet een kleur hebben. Computerkleuren maak je door Rood, Groen en Blauw (RGB) te mengen. Een kleur kun je dus maken door deze 3 RGB-kleuren een cijfer tussen de 0 en de 255 geven. In dit geval 0 rood, 0 Groen en 255 (maximaal) blauw om de kleur blauw te maken.
- Regel 63: met de functie `pygame.draw.circle()` kunnen we gekleurde cirkels tekenen. Deze functie heeft 4 argumenten: het scherm waarop getekend moet worden (in ons geval `screen`), de kleur (die hebben we in regel 62 gemaakt) , een setje van 2 coördinaten (die hebben we in regel 60 en 61 gemaakt) en een radius in pixels die bepaalt hoe groot de cirkel moet worden.

Regel 65 t/m 68 doen nog eens (bijna) hetzelfde, maar dan voor de pion van speler 2. Een paar kleine verschillen:

- We zoeken natuurlijk de positie van speler 2 op met `posities[1]`.
- Een andere kleur natuurlijk (groen in dit geval)
- De opgezochte coördinaten verhogen we iets (met +3). Dit is zodat de pionnen niet precies bovenop elkaar staan. Als ze op het zelfde vakje staan (zoals nu) zou je anders maar 1 pion zien, omdat de andere er precies bovenop staat...

Test je spel. Je zou nu 2 pionnen moeten zien op het startvakje!

5.9 DOBBELEN EN PIONNEN VERZETTEN

Het bord en de pionnen staan klaar, tijd voor wat interactie. Om het gooien en verzetten goed te laten werken, moeten we eerst 2 globale variabelen toevoegen:

```
11 vakjes = [[80, 385], [152, 385], [192, 385]]
12
13 # pion posities
14 posities = [0,0]
15
16 # wie is aan de beurt?
17 beurt = 0
18
19 # dobbelsteenworp:
20 worp = 0
21
22
23 # -------- Pygame initialisatie ---------
```

In de variabele `beurt`, houden we bij welke speler er aan de beurt is. We gebruiken 0 voor speler 1 en 1 voor speler 2 (Dit is handiger, omdat een lijst in Python ook met element 0 begint en we de waarde van `beurt` willen gebruiken om info van de spelers uit andere lijsten te halen). We zetten `beurt` op 0, zodat speler 1 mag beginnen.

In de variabele `worp` gaan we de dobbelsteenworp opslaan. Deze zetten we voor nu op 0, omdat er nog niet gegooid is, later vullen we deze variabele steeds met een random nummer tussen 1 en 6.

Nu gaan we regelen dat als de speler op de spatie drukt, het volgende gebeurt:

- Er wordt een dobbelsteen gegooid (dus een random getal tussen 1 en 6 gegenereerd)
- De speler die aan de beurt is, wordt verzet
- De beurt wordt doorgegeven aan de volgende speler

Eerst moeten we zorgen dat een druk op de spatie wordt gedetecteerd. Dit is een event, dus we moeten de lijst met opgevangen events in onze hoofdloop uitbreiden:

```
51     # --- Check gebeurtenissen (zoals muiskliks e.d.) ---
52
53     for event in pygame.event.get():
54         if event.type == pygame.QUIT:  # Het kruisje is aangeklikt
55             done = True  # Het spel moet eindigen dus we zetten done op True
56
57         elif event.type == pygame.KEYDOWN:
58             # Er is een toets ingedrukt, we kijken welke en ondernemen actie
59             if event.key == pygame.K_SPACE:
60                 print "Knop: Spatie"
61
```

Naast kliks op het kruisje, vangen we nu ook toetsaanslagen af (regel 57). Vervolgens kijken we welke toets precies is ingedrukt en als dat de spatie is (regel 59) drukken we voor nu even een melding af op de console (die staat in

PyCharm onderaan het editorscherm).

Test je programma eens uit. Als het goed is verschijnt er elke keer dat je op de spatie drukt een melding in de console.

We willen natuurlijk dat er nog meer gebeurt dan alleen maar een melding als de spatie is ingedrukt.

Als de spatie is gedetecteerd gaan we de dobbelsteen een random waarde tussen 1 en 6 geven. We verzetten de pion die aan de beurt is en we geven de beurt door. Hieronder staat een deel van de code, de rest (de stukken met ...) zou je zelf moeten kunnen invullen:

```
57          elif event.type == pygame.KEYDOWN:
58              # Er is een toets ingedrukt, we kijken welke en ondernemen actie
59              if event.key == pygame.K_SPACE:
60                  print "Knop: Spatie"
61
62                  worp = ...  # kies een random getal tussen 1 en 6 als dobbelsteenworp
63                  posities[beurt] += worp # verzet pion
64
65                  #verzin iets slims om de beurt door te geven aan de volgende speler:
66                  ...
67                  ...
68
69      # --- Teken de graphics (nog buiten beeld) ---
```

Op regel 63 verhogen we de positie van de speler die aan de beurt is met de waarde van de dobbelsteenworp. Hiermee verzetten we effectief de pion. Dit wordt ook direct op het scherm weergegeven, omdat verderop in het “Teken de graphics” deel van de code de pionnen altijd worden getekend op de positie waar ze volgens de variabele `posities` staan. Dus als we de positie wijzigen, worden ze automatisch ergens anders getekend. Handig!

Op regel 66 moet je dus iets verzinnen waardoor de beurt word doorgegeven. Dit zorgt ervoor dat als de volgende keer op de spatie wordt gedrukt de andere speler wordt verzet (en daarna weer de eerste, enzovoort).

Test je spel. Je zou nu rennende pionnen moeten zien! ☺

5.10 SPELEINDE

We hebben nu netjes rondrennende pionnen, maar het zal je vast al opgevallen zijn dat als ze voorbij het laatste vakje lopen er een foutmelding verschijnt. Dat komt omdat python dan probeert de niet-bestaande coördinaten op te zoeken van een niet-bestaand vakje. Dit moeten we voorkomen.

Als er een speler op of voorbij het laatste vakje (in dit voorbeeld vakje 63) komt, heeft deze speler gewonnen en willen we het spel beëindigen. Om dat voor elkaar te krijgen, moeten we elke keer dat we een speler verzet hebben meteen checken of deze toevallig op of over vakje 63 gekomen is.

Het verzetten van de pionnen gebeurt bij de events, om precies te zijn op de plek waar we de spatiebalk detecteren. Ddus daar moeten we de code toevoegen:

```
57          elif event.type == pygame.KEYDOWN:
58              # Er is een toets ingedrukt, we kijken welke en ondernemen actie
59              if event.key == pygame.K_SPACE:
60                  print "Knop: Spatie"
61
62                  worp = .. # kies een random getal tussen 1 en 6 als dobbelsteenworp
63                  posities[beurt] += worp # verzet pion
64
65                  # is de pion op of voorbij het laatste vakje?
66                  # zet hem dan precies op het laatste vakje:
67                  if posities[beurt] >= 63:
68                      posities[beurt] = 63
69
70                  # heeft de speler nog niet gewonnen? geef dan gewoon de beurt door:
71                  else:
72                      # verzin iets slims om de beurt door te geven aan de volgende speler
73                      # deze code heb je in een vorige stap al geschreven als het goed is...
74                      ...
75                      ...
76
77      # --- Teken de graphics (nog buiten beeld) ---
```

Op regel 76 checken we of de speler (die we in de regels daarvoor nog verzet hebben) op of over vakje 63 staat. Zo ja, zetten we hem precies op 63 (dit voorkomt alvast de niet-bestaande coördinaten).
We geven de beurt nu alleen nog maar door, als de if niet waar is, dus als de speler nog niet gewonnen heeft. Hierdoor blijven als er een speler wel gewonnen heeft de pionnen gewoon staan.

5.11 OPNIEUW BEGINNEN

Het zou handig zijn om als het spel afgelopen is (of zelfs tussendoor) een nieuw spel te kunnen beginnen. We gaan dit doen als de speler op de backspace knop drukt. Dit event moeten we even toevoegen:

```
57          elif event.type == pygame.KEYDOWN:
58              # Er is een toets ingedrukt, we kijken welke en ondernemen actie
59              if event.key == pygame.K_SPACE: #spatie
60                  print "Knop: Spatie"
61
62                  worp = .. # kies een random getal tussen 1 en 6 als dobbelsteenworp
63                  posities[beurt] += worp # verzet pion
64
65                  # is de pion op of voorbij het laatste vakje?
66                  # zet hem dan precies op het laatste vakje:
67                  if posities[beurt] >= 63:
68                      posities[beurt] = 63
69
70                  # heeft de speler nog niet gewonnen? geef dan gewoon de beurt door:
71                  else:
72                      # verzin iets slims om de beurt door te geven aan de volgende speler
73                      # deze code heb je in een vorige stap al geschreven als het goed is...
74                      ...
75                      ...
76
77              elif event.key == pygame.K_BACKSPACE: #backspace
78                  print "Knop: Backspace"
79                  # bedenk zelf hoe je de spelerposities weer op 0 kunt zetten
80                  # en hoe je de beurt weer aan speler 1 geeft
81
82      # --- Teken de graphics (nog buiten beeld) ---
```

Je ziet dat we naast het afvangen van de spatie nu ook de backspace (regel 77) kunnen detecteren. Als deze wordt ingedrukt, moeten een aantal globale variabelen worden aangepast. Je moet nu zelf even bedenken welke dat zijn en welke waarde ze moeten krijgen…

Test je spel om te checken of je nu met een druk op de backspace een nieuw spel kunt starten.

5.12 INFORMATIE OP HET SCHERM

In principe is de basis van ons spel klaar. Het laatste wat we nog willen doen, is wat informatie aan de speler geven. Hiervoor willen we wat tekst afdrukken op het scherm.

We gaan afdrukken wie er aan de beurt is en wat de laatste dobbelsteenworp is geweest. Deze informatie zit al in onze globale variabelen beurt en worp. We hoeven deze info alleen nog maar om te zetten in iets wat voor de gebruiker leesbaar is. Dit doen we door de tekst op het scherm te tekenen. We moeten dan ook code toevoegen aan het "Teken de graphics" deel van onze hoofdloop:

```
82    # --- Teken de graphics (nog buiten beeld) ---
83
84    screen.fill((255,255,255)) # begin met een witte achtergrond
85
86    bordrect = bord.get_rect() # vraag afmetingen (rectangle) van het bordplaatje op
87    screen.blit(bord, bordrect) # teken het bord
88
89    #teken pionnen als gekleurde cirkels op de coordinaten van de vakjes waar ze staan:
90    speler1_x = vakjes[posities[0]][0]
91    speler1_y = vakjes[posities[0]][1]
92    speler1_kleur = (0, 255, 0) #groen
93    pygame.draw.circle(screen, speler1_kleur, (speler1_x, speler1_y), 10)
94
95    speler2_x = vakjes[posities[1]][0] + 3
96    speler2_y = vakjes[posities[1]][1] + 3
97    speler2_kleur = (0, 0, 255) #blauw
98    pygame.draw.circle(screen, speler2_kleur, (speler2_x, speler2_y), 10)
99
100
101   #Kies een lettergrootte:
102   myfont = pygame.font.SysFont(None, 25)
103
104   #teken de laatste worp op het scherm:
105   text ="Laatste worp: " + str(worp)
106   label = myfont.render(text, 1, (0, 0, 0))
107   screen.blit(label, (450, 460))
108
109   #teken de speler die aan de beurt is op het scherm:
110   text ="Aan de beurt: " + str(beurt + 1)
111   label = myfont.render(text, 1, (0, 0, 0))
112   screen.blit(label, (450, 480))
113
114
115   # --- Ververs het beeldscherm met de nieuwe graphics ---
```

- Op regel 102 stellen we een lettergrootte in door een font object te maken
- Op regel 105 maken we een string die we willen afdrukken om de worp weer te geven
- Op regel 106 maken we een plaatje van de string uit regel 105 en kiezen zwart (0, 0, 0) als tekstkleur
- Op regel 107 voegen we het gemaakte plaatje uit regel 106 aan het nieuwe scherm toe
  Regel 110 t/m 112 doen iets vergelijkbaars voor de speler die aan de beurt is.

Test je game om te zien af alles werkt. Ja? Gefeliciteerd, je basisbordspel is klaar! Nu uitbreiden dat ding!

6 TIPS VOOR UITBREIDINGEN EN VERBETERINGEN

Je hebt een werkend, maar nog erg primitief bordspel gemaakt, maar wat nu? Hier vind je enkele tips en ideeën om verder te gaan. Je kunt alle kanten op, dus wees zelf creatief, kies een tof thema en maak een gave game!

Belangrijk! Denk na over wat je wilt doen, waarop je beoordeeld wordt en bedenk eerst een plan.

- Denk er aan dat je ook een **eigen algoritme** en een **abstractie** moet maken (zie hoofdstuk Inleveren). Tip: Het ligt voor de hand om hiervoor een aantal regels uit te programmeren (zie Regels hieronder).
- Je programma wordt beoordeeld op **codekwaliteit** en **elegantie**. Het is aan te raden om **alle** code verbeteringen die voorgesteld worden (zie hoofdstuk code verbetering hieronder) door te voeren.
- Tip: maak eerst 6.1 en 6.2 af, zorg dat je je algoritme en abstractie beschreven hebt in je verslag (zie hoofdstuk Beoordeling) en maak daarna een keuze uit een van de overige verbetervoorstellen.

6.1 CODE VERBETERINGEN

De code die je nu hebt is niet overzichtelijk en voldoet ook niet aan code afspraken die we hebben gemaakt. Voer de volgende wijzigingen door:

- Geef de venster een nieuwe (eigen naam).
- Voeg commentaar toe. Minimaal per functie en bij het gebruik van nieuwe Pygame methoden (bv. flip)
- Gebruik functies en parameters en zorg voor juiste naamgeving (functie namen in camelCase). Verhuis delen van de code nu alvast naar eigen functies en test je programma goed voordat je verder gaat.
- Globale constanten, dingen die nooit veranderen in het spel moet je bovenaan je code definiëren en kies een naam bestaand uit alleen hoofdletters.
- Gebruik zo min mogelijk globale variabelen. Dit kun je voorkomen door slim gebruik te maken van parameters. Gebruik je wel globale variabelen, doe dat dan boven aan je programma.
- Variabelen: Zorg voor juiste naamgeving (zinvol_en_correct). Bijvoorbeeld, de variabele ‘done’ zegt niet zo veel (done…. met wat?).

6.2 REGELS

Het bedenken en coderen van toffe regels zijn de belangrijkste uitdaging van deze PO. Bijvoorbeeld:

- Je kunt de officiële ganzenbordregels gebruiken, zoals:
  - Beurten overslaan op bepaalde vakjes
  - Nog eens gooien
  - De put
  - Achteruit lopen als je niet precies op 63 uitkomt
  - ….
- Je kunt ook je eigen regels verzinnen:

- Inventory met items die je op bepaalde vakjes krijgt waar je in het spel wat mee kan om je een voordeel (of je tegenspeler een nadeel!) te geven,
- Score
- Geld dat je kunt verdienen om items/upgrades te kopen
- Elkaar slaan als je op hetzelfde vakje komt
- Een quiz of minigame op bepaalde vakjes
- …

6.3 GRAFISCHE EN TECHNISCHE VERBETERINGEN

Los van thema's en regels kun je natuurlijk allerlei technische en/of grafische verbeteringen aanbrengen, zoals:

- Kiezen met hoeveel spelers je speelt
- Namen en pionnen laten kiezen (misschien met andere skills/eigenschappen zelfs)
- Muisbesturing met interface knoppen die je kunt indrukken
- Een scherm met de spelregels en het verhaal
- Saven en laden van lopende spellen (stand opslaan in een tekst bestand)
- Animaties van lopende pionnen en rollende dobbelstenen
- …
- …

De mogelijkheden zijn eindeloos. Op internet is van alles te vinden. De uitdaging is om iets tofs te bedenken en uit te zoeken hoe je het kunt bouwen.

Je kunt de volgende handleiding bekijken voor inspiratie: Tutorial GUIs met PyGame

6.4 MEER SOORTEN INPUT

Hier nog een handig stukje code voor als je meerdere toetsen en muiskliks wilt gebruiken:

```
58      # --- Check gebeurtenissen (zoals muiskliks e.d.) ---
59      for event in pygame.event.get():
60          if event.type == pygame.QUIT:  # Het kruisje is aangeklikt
61              done = True  # Het spel moet eindigen dus we zetten done op True
62
63          elif event.type == pygame.MOUSEBUTTONDOWN:
64              # Er is met de muis geklikt. Vraag nu de coordinaten op:
65              pos = pygame.mouse.get_pos()
66              mouseX = pos[0]
67              mouseY = pos[1]
68              print "Click: ", mouseX, mouseY
69              # deze code doet nog niets anders dan muiskliks detecteren
70              # deze kun je later gebruiken voor interactie
71
72          elif event.type == pygame.KEYDOWN:
73              # Er is een toets ingedrukt, we kijken welke en ondernemen actie
74              if event.key == pygame.K_LEFT:
75                  print "Knop: Links"
76              elif event.key == pygame.K_RIGHT:
77                  print "Knop: Rechts"
78              elif event.key == pygame.K_UP:
79                  print "Knop: Omhoog"
80              elif event.key == pygame.K_DOWN:
81                  print "Knop: Omlaag"
82              elif event.key == pygame.K_BACKSPACE:
83                  print "Knop: Backspace"
84              elif event.key == pygame.K_SPACE:
85                  print "Knop: Spatie"
86
```

Je kunt de volgende handleiding bekijken voor inspiratie: Tutorial GUIs met PyGame

6.5 POP-UPS

Je kunt makkelijk simpele pop-ups toevoegen. Je moet de volgende import toevoegen bovenaan je code:

```
import ctypes
```

Met de volgende code kun je een pop-up venster tonen zoals hieronder:

```
# toont een popup venster met tekst "gewonnen!", OK en Cancel knop. Title is "Einde".
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "Je hebt gewonnen!", "Einde", 1)
```

Als je nog iets speciaals wilt doen als er op een knop wordt gedrukt, dan kun je het volgende code gebruiken:

```
# toont een popup venster met tekst “gewonnen!”, OK en Cancel knop. Title is “Einde”.
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 1)

if gebruikersKeuze == 1: #er is op OK geklikt
    print("Pressed OK")
elif gebruikersKeuze == 2: #er is op cancel geklikt
    print("Pressed Cancel")
```

Met de volgende code krijg je er nog een icoontje en geluidje bij:

```
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 0x40 | 0x1
```

Hier een overzicht van andere knopjes die je kunt laten zien:

```
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 0 ) # OK
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 1 ) # OK, Cancel
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 2 ) # Abort, Retry, Ignore
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 3 ) # Yes, No, Cancel
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 4 ) # Yes, NO
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 5 ) # Retry, Cancel

# knopjes “Cancel”, “Try Again” en “Continue:
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 6 )

# icoontje met uitroepteken en met knoppen: OK en cancel:
gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 0x40 | 0x1 )

# icoontje met uitroepteken en met knoppen: Yes, no, en cancel:
 gebruikersKeuze = ctypes.windll.user32.MessageBoxW( 0, "gewonnen!", "Einde", 0x40 | 0x3 )
```

Je kunt de volgende handleiding bekijken voor inspiratie: Tutorial GUIs met PyGame

6.6 THEMA

Elk bordspel heeft een thema dat de sfeer van het spel en de regels bepaalt.
Denk bijvoorbeeld eens aan:

- Een film of serie
- Rome/Griekenlandreis
- Reis om de wereld
- Een levensloop van brugklasser tot bovenbouw op het Montessori College
- Een sport
- …

Je zorgt natuurlijk dat je bord en pionnen (en eventuele andere extra’s) bij je thema aansluiten.