PO PYTHON PROGRAMMEREN:

GANZENBORD

BOUW JE EIGEN BORDSPEL MET PYGAME

J. van Weert
Wt (@) sgdb.nl
Stedelijk Gymnasium 's-Hertogenbosch
2025

V3.0

1 INLEIDING

Gamen is leuk, maar zelf een game maken is nog een stuk leuker natuurlijk. Nu knutsel je niet op een vrijdagmiddag zomaar Call of Duty 86 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 weken van alles geleerd over Python en je hebt de belangrijkste programmeerconcepten zoals variabelen, loops, lijsten en allerlei handige constructies geleerd. Tot nu toe heb je ze alleen steeds in kleine oefeningen en opdrachten 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 library PyGame. Het is vervolgens aan jou om boven op 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 vooral je eigen ding van te maken.

Have fun!

Wt & Ev

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. Met het volgen van de instructies uit dit document haal je alvast een klein deel van de punten voor deze PO en heb je een basisversie van een bordspel. De overige punten zitten (zoals gewoonlijk) in je eigen uitbreidingen, 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...)

Naast het schrijven van de code, maak je ook een kort (Word) document waarin je de uitbreidingen/regels van je spel uitlegt en vertelt hoe je het hebt gemaakt (dus: hoe werkt je code? En: wat zijn de regels en systemen van het spel?). Deze uitleg, in combinatie met commentaar in je code, moeten Wt helpen begrijpen wat je precies gedaan hebt. Je hoeft niet elke stap toe te lichten (Wt spreekt zelf ook Python), maar beschrijf je codeopzet en geef uitleg over de regels en werking van het spel. *Belangrijk: Heb je externe bronnen gebruikt (tutorials, online hulp, etc): vermeld ook wat en hoe je hebt gebruikt.*

2.2 AI-GEBRUIK

2.2.1 DE SGDB AI-SCHAAL

Met de opkomst van generatieve AI (in de vorm van Chat-GPT en consorten) is er iets fundamenteel veranderd bij vrije praktische opdrachten zoals deze. AI kan goed overweg met (python)code en hij kan je goed helpen. Maar de verleiding kan dan groot zijn om het werk uit te besteden aan je favoriete chatbot en dat is nu ook weer niet de bedoeling. AI is een heel handige tool en kan heel leerzaam worden ingezet, maar daarvoor zijn er goede afspraken nodig. Hiervoor is de (nog in concept-versie) SGDB-AI schaal opgezet. Deze zul je de komende tijd vaker tegenkomen bij opdrachten en PO's (ook bij andere vakken dan informatica) en deze geeft de mate van toegestaan AI-gebruik overzichtelijk aan. De precieze details van wat is toegestaan vindt je altijd in de opdrachtomschrijving, dus deze blijft leidend.

2.2.2 AI-GEBRUIK BIJ DEZE OPDRACHT

Deze opdracht zit op trede 2 van de SGDB AI-schaal: "AI-ondersteund bewerken". Dit laat zich het beste samenvatten als:

"Op dit niveau mag de AI worden gebruikt om door de leerling gegenereerd materiaal aan te passen en te verbeteren, maar niet om hele nieuwe dingen te genereren."

Voor deze opdracht gaan we uit van drie "AI-principes":

AI-principe 1: Menselijke controle en begrip

Je bent als mens verantwoordelijk voor de code die je inlevert. AI mag gebruikt worden voor kleine aanpassingen en verbeteringen aan de code die je zelf schrijft, maar je wordt geacht te begrijpen wat je inlevert en je kunt desgevraagd mondeling toelichten wat elke regel code uit je programma doet.

AI-principe 2: Gebruik AI om van te leren

AI kan goed uitleggen wat bepaalde code doet en wat de achterliggende logica van zijn suggesties is. Als je AI gebruikt om je te helpen, vraag dan expliciet om deze toelichting. Dit zorgt ervoor dat je goed begrijpt wat de eventueel gegenereerde code doet en daar word je een betere programmeur van: win-win.

Denk bijvoorbeeld aan een prompt als:

"Ik krijg een foutmelding op regel 26. Kun je me uitleggen wat er mis gaat en hoe ik dit kan oplossen?"

AI-principe 3: Reflectie op AI-gebruik

Werken met AI is fundamenteel anders dan werken zonder AI. Je zult dus ook moeten leren hoe je het beste met AI om kunt gaan. Wanneer is het een goed idee en wanneer niet? Hoe stuur je een AI goed aan, waarbij je zelf de controle (en het begrip) houdt? Etc.
In je eindverslagje bij deze PO wordt verwacht dat je een korte reflectie hierop schrijft. Hoe heb je AI ingezet (of niet). Wat vond je ervan? Wat heb je geleerd over goed prompten?

Er zijn ook ethische bezwaren tegen AI denkbaar. Bijvoorbeeld het hoge energieverbruik met impact op het klimaat, het schenden van copyright bij het trainen van de AI, etc. Het staat je vrij om hier ook aandacht aan te besteden in de reflectie, maar dat is voor deze opdracht niet verplicht.

2.3 BEOORDELING

De beoordeling vindt plaats volgens dit beoordelingsmodel.

Een paar opmerkingen:

- Dit is een individuele opdracht, dus iedereen bouwt zijn eigen bordspel.
- Een flink deel van de beoordeling hangt samen met de complexiteit en diepgang van de uitbreidingen, maar er is zeker ook aandacht voor:
    - Nette code: goede layout, logische variabelenamen, zinvol commentaar etc.
    - Creativiteit en vormgeving
    - Reflectie (algemeen en AI-gebruik)

2.4 INLEVEREN

De deadline voor deze opdracht is vrijdag 9 januari.

LET OP: Deze PO is een SE onderdeel. Hiervoor gelden regels wat betreft het missen van de deadline. De procedure bij het niet voor de deadline inleveren van de opdracht is als volgt:

1. Examinator (Wt of Ev) constateert dat een leerling in gebreke blijft.
2. Examinator meldt dit z.s.m. aan de examensecretaris (dhr. Hegge), waarbij hij vermeldt om welke PO het gaat (nummer PTA) en wanneer het ingeleverd had moeten zijn.
3. Examensecretaris communiceert sanctie en een nieuwe deadline naar de leerling.
4. Herhaling stap 1 t/m 3 indien een leerling weer in gebreke blijft.

In de ELO staat een opdracht open. Hierin lever je een .zip file in met daarin:

- Je verslag
- Je Python bestand(en): *.py
- Gebruikte afbeeldingen en eventueel andere externe bestanden (audio, etc...) die bij de game horen

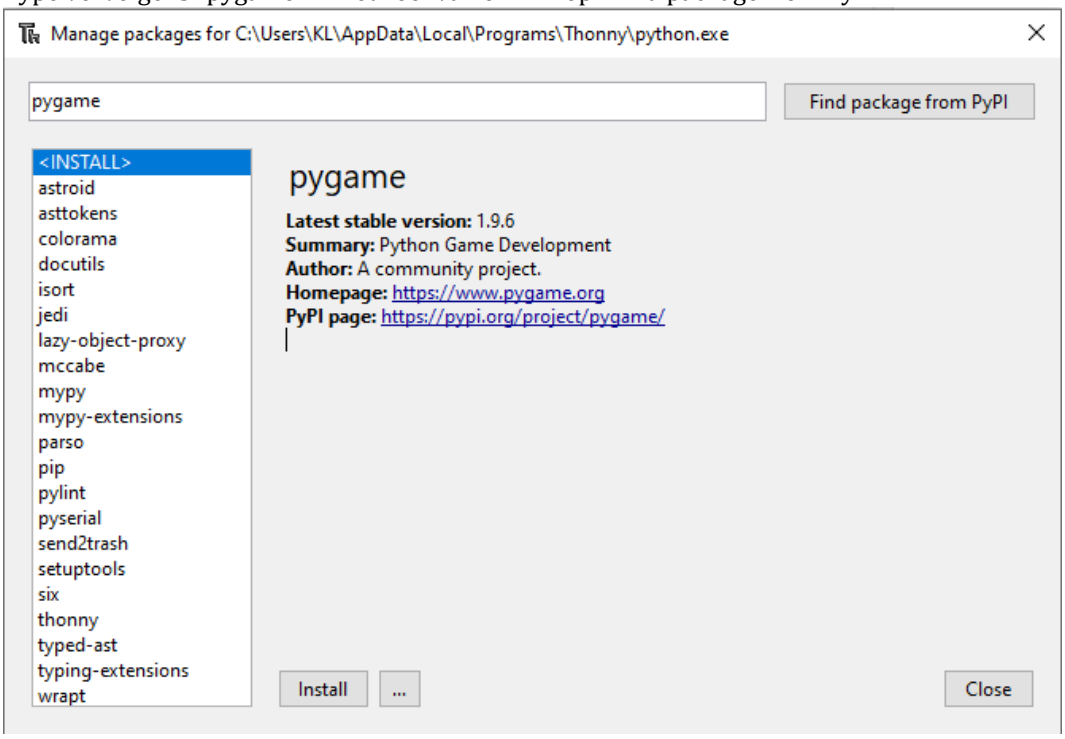
3 TOOLS

3.1 THONNY MET PYGAME

Een van de handige dingen aan Python is dat er ontzettend veel libraries zijn die een hoop functionaliteit aan Python toevoegen. Een library kun je zien als een verzameling Python code die door anderen geschreven is en centraal wordt bijgehouden. Je kunt libraries downloaden en importeren in je eigen projecten. Er zijn libraries voor alle denkbare dingen: voor data-analyse, AI, het ophalen van informatie uit webpagina's, etc. etc.

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 muiskliks en toetsaanslagen te detecteren en te verwerken. Pygame zit niet standaard in python geïnstalleerd (in tegenstelling tot standaard libraries, zoals bijvoorbeeld "random" en "math".) We zullen PyGame dus even moeten toevoegen aan onze Python installatie. Gelukkig is dat erg eenvoudig:

- Open Thonny
- Klik in het menu "Tools" op "Manage packages…"
- Type vervolgens "pygame" in het zoekvak en klik op "Find package from PyPI":



- Klik onderaan op "Install"
- Wacht even kort tot de installatie voltooid is en klaar! Je bent nu de trotse bezitter van de PyGame library
- *Denk eraan dat als je op een andere computer verder wilt werken (bijvoorbeeld thuis of in de mediatheek), je deze installatie even moet herhalen op die PC!*

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

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

3.2 PAINT.NET

Om de graphics van je game (je bord, pionnen etc.) te maken is een tekenprogramma handig. Paint.Net ligt voor de hand. Op school staat deze op alle computers. Voor thuis is hij gratis te downloaden: http://www.getpaint.net

Er zijn tegenwoordig ook veel handige editors die rechtstreeks in je browser werken:

- Piskelapp.com, speciaal voor pixel-art
- Photopea.com, een online kloon van photoshop

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):

- Eerst komen 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 (op welk vakje staat elke pion?)
    - Laatste dobbelsteenworp
    - De coördinaten van alle vakjes (nodig om pionnen op de goede plek te tekenen)
    - 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 eindeloze loop (gameloop) die voortdurend het volgende uitvoert:
    - Checken of er input is (toetsaanslagen, muiskliks, 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 eindeloze 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 BORDSPEL IN PYTHON MET PYGAME

In de volgende paragrafen wordt je stap voor stap geholpen met de basis van je game. Wt heeft 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.

Let op: *Je wordt geacht deze basisuitleg in enkele lessen te doorlopen. In het beoordelingsmodel krijg je punten voor dit onderdeel op basis van op welk moment je je docent hebt laten zien dat je de code hebt gemaakt (en hebt begrepen!)*

5.1 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

Open Thonny en maak een nieuw Python document. Neem de volgende code over. Sla je file meteen daarna vast even op.

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

Je ziet dat we met behulp van regels commentaar de code in een aantal blokken verdelen. Dit is gewoon handig om het overzicht te bewaren en structuur in je code aan te brengen. Later gaan we sommige blokken nog verder onderverdelen. Voel je vrij om als je later de code gaat uitbreiden meer van deze blokken toe te voegen. Als je code langer wordt is het steeds belangrijker om overzichtelijk te werken.

Voor nu gebeurt er alvast dit:

- 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 17 sluiten we alle pygame dingen netjes af met het commando `pygame.quit()`

5.2 PYGAME INITIALISATIE

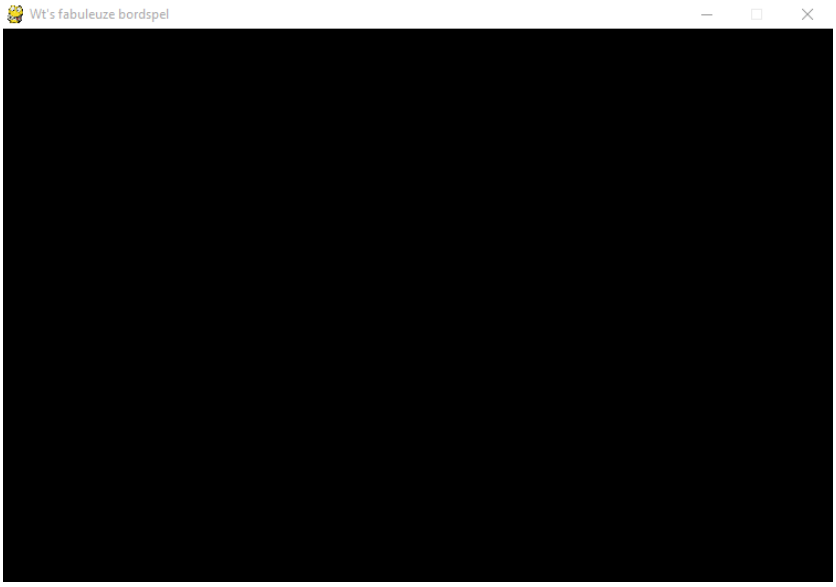
Als je met pygame werkt moeten er een aantal dingen worden “klaargezet”. Het belangrijkste is dat er een gamescherm gemaakt moet worden waarop straks onze graphics verschijnen. Ook is een “clock” nodig om straks het gamescherm steeds te verversen.

Dit doen we onder het kopje “Python initialisatie”. Neem de volgende code onder dat blokje over. Het commentaar legt uit wat de regels doen:

```
# -------- Pygame initialisatie ---------

# Pygame initialiseren (is altijd nodig bij begin van gebruik pygame)
pygame.init()

# Afmetingen van het spelscherm instellen (in pixels [breedte, hoogte])
# Daarna het spelscherm maken (en opslaan in een variabele screen)
WINDOW_SIZE = [1200, 800]
screen = pygame.display.set_mode(WINDOW_SIZE)

# Titel van spelscherm instellen:
pygame.display.set_caption("Wt's fabuleuze bordspel")

# Deze boolean laat ons spel straks in een oneindige loop lopen, totdat er op
# het kruisje wordt geklikt om af te sluiten (deze zet dan done op True)
done = False

# We maken een pygame Clock object. Deze is nodig om de verversingssnelheid
# (framerate) van het scherm te beheren
clock = pygame.time.Clock()

# -------- Hoofdloop van het programma -----------

```

Je kunt je programma om te testen vast eens runnen (Druk op de play knop of druk op F5).

Als het goed is zie je dan een scherm voorbij flitsen (hij sluit wel meteen af, want er is nog geen oneindige loop die hem in leven houdt. Dat fixen we verderop) met de opgegeven afmetingen en de juiste titel:



5.3 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:

```
31 # -------- Hoofdloop van het programma -----------
32
33 while not done:
34
35     # --- Check gebeurtenissen (zoals muiskliks e.d.) en werk de administatie bij ---
36
37
38     # --- Teken de graphics voor de volgende schermupdate (nog buiten beeld) ---
39
40
41     # --- Update het beeldscherm met de nieuwe graphics ---
42
43     clock.tick(60) # Zet de limiet op 60 frames per seconde
44     pygame.display.flip() # ververs het beeldscherm met de nieuwe versie van het scherm
45
46
47 # -------- Afsluiting -----------
48 pygame.quit() #dit sluit pygame netjes af en sluit het gamevenster
```

- De `while not done:` zorgt feitelijk voor een oneindige loop. Alle tijd dat ons spel runt, zal deze loop blijven lopen. Zolang de variabele `done` op `False` blijft staan (en dat hebben we in regel 24 zo ingesteld), blijft de loop herhalen. We voegen zo meteen nog code toe om hem met het afsluiten van het spel op `True` te zetten)

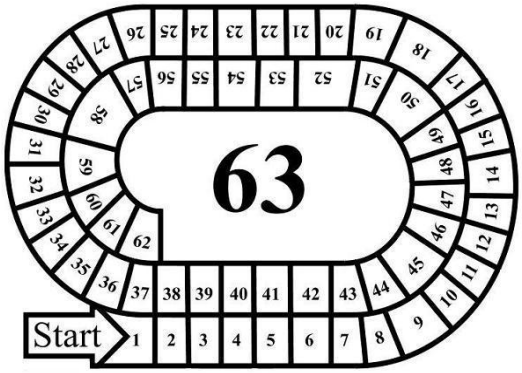
- Elke stap van deze oneindige loop bestaat uit 3 fasen:
    - Luister naar invoer (toetsenbord, muiskliks, etc.). Op basis van eventuele gedetecteerde invoer wordt het spel bediend en moet de toestand van het spel worden aangepast (bv. Het wijzigen van de positie van een pion na een worp). We koppelen later stukken code aan verschillende inputs.
    - Teken steeds een nieuwe versie van het spelscherm. Dit gebeurt nog "buiten beeld": Er wordt als het ware een nieuwe versie van het gamescherm opgebouwd en klaargezet.
    - Als het nieuwe scherm helemaal is opgebouwd, zet het dan 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 43 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.

NB: Het aantal flips per seconde noem je de framerate of FPS (frames per second). In een actiespel wil je dit zo hoog mogelijk hebben om het er vloeiend uit te laten zien. In ons bordspel is er wat minder actie en is een hoge framerate niet zo belangrijk. 60 is eigenlijk al "overkill".

5.4 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 kaal bord dat je goed als basis kunt gebruiken (in Paint.net kun je er natuurlijk een mooi "aangekleed" bord van maken), maar je mag ook een totaal eigen bord maken met een andere layout natuurlijk. Je hoeft ook helemaal niet 63 vakjes te hebben, meer of minder mag ook. Het is voor de basisversie wel handig om met dit standaard bord te werken. Je kunt later het bord gewoon vervangen.



Zet het plaatje van je bord in dezelfde map als waar je python script is opgeslagen, anders kan je programma het bestand niet vinden. Onder het stuk " Globale variabelen" bovenaan je code zet je vervolgens het volgende neer om het plaatje in te laden (we maken eigenlijk een variabele `bord` met het plaatje erin):

```
# -------- Globale variabelen --------

# bord afbeelding:
bord = pygame.image.load("ganzenbord.png")
```

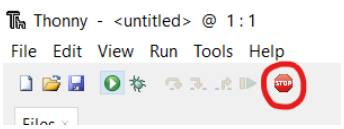
Hierbij is `ganzenbord.png` de bestandsnaam van je afbeelding. Als jouw plaatje anders heet, pas je dat hier aan.

Onder het kopje "Teken de graphics" in de hoofdloop zet je het volgende neer om het plaatje daadwerkelijk op het scherm te tekenen. Het commentaar licht toe wat er gebeurt:

```
    # --- Teken de graphics voor de volgende schermupdate (nog buiten beeld) ---

    screen.fill((255,255,255)) # begin met een witte achtergrond

    bordrect = bord.get_rect() #vraag afmetingen (rectangle) van het bordplaatje op
    screen.blit(bord, bordrect) # teken het bord op het volgende screen:
```

Let op dat als je het spel nu runt, dat hij vastloopt als je op het kruisje in het pygame venster klikt. Dat komt omdat we dat nog niet hebben geprogrammeerd. Dat gaan we hierna doen.

Je kunt je game toch stoppen door in Thonny op stop te klikken:



5.5 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:

```
33 # -------- Hoofdloop van het programma -----------
34
35 while not done:
36
37     # --- Check gebeurtenissen (zoals muiskliks e.d.) en werk de administatie bij ---
38
39     for event in pygame.event.get(): #doorloop alle gebeurtenissen sinds de vorige schermupdate
40
41         if event.type == pygame.QUIT:  # GEbeurtenis: het kruisje is aangeklikt
42             done = True  # Het spel moet eindigen dus we zetten done op True, zodat de loop straks stopt
43
```

Alle toetsaanslagen en muiskliks en andere events worden door pygame geregistreerd en aan een list met events toegevoegd. Daar wachten ze op “afhandeling”.

Met deze `for`-loop doorlopen we deze list met events elk frame en controleren van elk event het type (er zijn verschillende soorten events, zoals toetsaanslagen, muiskliks, etc). We checken of er een zogenaamd `QUIT`-event in voorkomt (met de `if` op regel 41). Als dat zo is, zetten we de `done` boolean op `True`. Hiermee laten we de “oneindige” hoofdloop eindigen en zal ons spel stoppen. Omdat als allerlaatste regel code na de loop de `pygame.quit()` staat, wordt het programma netjes beëindigd zonder te crashen.

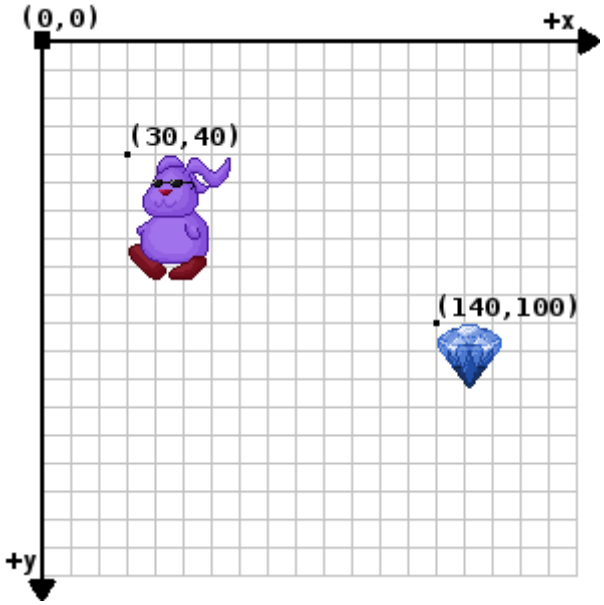
Later gaan we nog meer events detecteren (toetsaanslagen) en zullen we deze code nog uitbreiden.

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

5.6 COÖRDINATEN VAN DE VAKJES OPSLAAN

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

In computergames is de oorsprong altijd linksboven (in tegenstelling tot wat je gewend bent bij bijvoorbeeld wiskunde of natuurkunde). Dus pixel (0,0) is de pixel linksboven in het scherm, pixel (30, 40) is de pixel 30 stapjes rechts en 40 stapjes onder pixel (0,0), etc.



Computer graphics assenstelsel met sprites (bron: Wikipedia)

Als je graphics tekent die groter zijn dan een pixel, bijvoorbeeld een plaatje (sprite), dan gebruiken we de linkerbovenhoek van het plaatje als coördinaat (zoals in de afbeelding hierboven).

We moeten dus van elk van onze vakjes het x en y coördinaat opslaan.
Het handigste is om deze coördinaten in een python list op te slaan. 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. Dit is later makkelijk zoeken naar het juiste vakje

Hiervoor voeg je onder het kopje “Globale variabelen” in je code iets van deze strekking in:

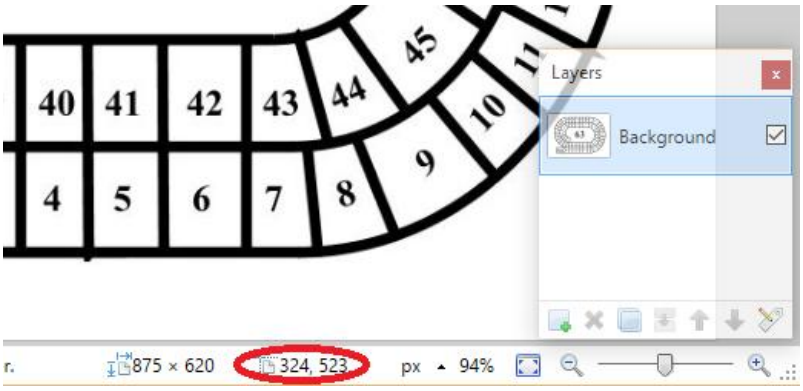
```
5  # -------- Globale variabelen --------
6
7  #coordinaten van de vakjes:
8  vakjes = [[160, 683], [286, 683], [356, 683]]
9
10 # bord afbeelding:
11 bord = pygame.image.load("ganzenbord.png")
```

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 160 en y-coördinaat 683, 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 Wt's standaard bord gebruikt, kan ik je wat werk besparen. Hier is een lijst met de coördinaten ervan *(Dank Wt op je knieën voor dit slavenwerk dat ik je bespaar…)*

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

Het handigste is om met je muis een beetje linksboven in elk vakje te gaan staan en dat als coördinaat voor dat vakje te gebruiken. Als we dan straks een plaatje tekenen met dat coördinaat als linker bovenhoek, staat het plaatje netjes middenin het vakje.)



5.7 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. Immers `vakjes[0]` geeft ons de coördinaten van vakje 0, `vakjes[2]` van vakje 2, etc. Dat was het voordeel van de coördinaten in een list zetten.

Onder het kopje "Globale variabelen" bovenin je code slaan we de speler posities ook op in een lijst. Dat maakt het straks namelijk makkelijk om code niet dubbel te hoeven schrijven voor elke speler. Ook is het uitbreiden naar meerdere spelers dan eenvoudiger.

Deze zetten we om te beginnen voor beide spelers op 0, want 0 is namelijk ons startvakje:

```
5  # -------- Globale variabelen --------
6
7  #coordinaten van de vakjes:
8  vakjes = [[160, 683], [286, 683], [356, 683]]
9
10 # pion posities
11 posities = [0,0]
12
13 # bord afbeelding:
14 bord = pygame.image.load("ganzenbord.png")
```

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

(NB: We nummeren de spelers vanaf 0, omdat dat handig is: Python begint met indexen van lijsten ook te tellen bij 0. Je kunt dat het speler nummer dan gebruiken om in de lijsten te zoeken)

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:

```
50     # --- Teken de graphics voor de volgende schermupdate (nog buiten beeld) ---
51
52     screen.fill((255,255,255)) # begin met een witte achtergrond
53
54     bordrect = bord.get_rect() #vraag afmetingen (rectangle) van het bordplaatje op
55     screen.blit(bord, bordrect) # teken het bord op het volgende screen:
56
57     #teken pionnen als gekleurde cirkels op de coordinaten van de vakjes waar ze staan:
58     speler0_x = vakjes[posities[0]][0]  #x-coordinaat pion speler 0
59     speler0_y = vakjes[posities[0]][1]  #y-coordinaat pion speler 0
60     kleur0 = (0,255,0) # groen
61     pygame.draw.circle(screen, kleur0, (speler0_x, speler0_y), 10) # teken cirkel als pion speler 0
62     speler1_x = vakjes[posities[1]][0] + 5  #x-coordinaat pion speler 1
63     speler1_y = vakjes[posities[1]][1] + 5  #y-coordinaat pion speler 1
64     kleur1 = (0, 0, 255) # blauw
65     pygame.draw.circle(screen, kleur1, (speler1_x, speler1_y), 10)  # teken cirkel als pion speler 1
```

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

- Regel 58: We willen het x-coördinaat opzoeken van het vakje van speler 0.
    - Hiervoor moeten we eerst weten op welk vakje speler 0 staat, dus dat zoeken we op met `posities[0]`. Dit geeft ons het vakjesnummer waar speler 0 op staat.
    - Het getal wat daar uit komt vullen we in als index voor onze lijst met vakjes-coördinaten. Dus daardoor wordt het `vakjes[posities[0]]`. Dit geeft ons de x- en y-coördinaten van het vakje waarop speler 0 staat.
    - Omdat we alleen het x-coördinaat van het vakje zoeken, moeten we het eerste van de twee getallen uit het coördinaten halen, deze heeft index 0. Daarom is de hele code: `vakjes[posities[0]][0]`. Dit geeft ons het x-coördinaat van het vakje waar speler 0 staat
- Regel 59: Exact hetzelfde als regel 58, maar dan zoeken we het y-coördinaat, dus moeten we het tweede getal uit de lijst met opgezochte coördinaten hebben: `vakjes[posities[0]][1]`
- Regel 60: De pion moet een kleur hebben. Zoals je weet, maak je op de computer kleuren 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, 255 (maximaal) Groen en 0 blauw om de kleur groen te maken.
- Regel 61: 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 60 gemaakt, `kleur0`) , een setje van 2 coördinaten (die hebben we in regel 58 en 59 gemaakt) en een radius in pixels die bepaalt hoe groot de cirkel moet worden. Hier kiezen we voor 10.

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 (blauw in dit geval)
- De opgezochte coördinaten verhogen we iets (met +5). 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!

Extra test:
We hebben de code op een slimme manier gebouwd, want het aanpassen van de waardes van `posities` is genoeg om de pionnen te verplaatsen. De code die we net geschreven hebben zoekt namelijk altijd de juiste coordinaten op op basis van deze list. Test het maar eens:

Als je in de variabele posities de [0,0] vervangt door andere getallen en het spel nog eens test, zie je als het goed is dat de pionnen nu op die vakjes beginnen.

Zet als je klaar bent met testen `posities` weer terug op [0,0]

5.8 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: `beurt` en `worp`.

```
7  #coordinaten van de vakjes:
8  vakjes = [[160, 683], [286, 683], [356, 683]]
9
10 # pion posities
11 posities = [0,0]
12
13 # wie is aan de beurt?
14 beurt = 0
15
16 # dobbelsteenworp:
17 worp = 0
18
19 # bord afbeelding:
20 bord = pygame.image.load("ganzenbord.png")
```

In de variabele `beurt` houden we bij welke speler er aan de beurt is. We gebruiken de waarde 0 voor speler 0 en de waarde 1 voor speler 1. We zetten `beurt` om te beginnen op 0, zodat speler 0 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 het checken op toetsaanslagen toevoegen aan onze loop die de events afhandelt:

```
46 while not done:
47
48     # --- Check gebeurtenissen (zoals muiskliks e.d.) en werk de administatie bij ---
49
50     for event in pygame.event.get(): #doorloop alle gebeurtenissen sinds de vorige schermupdate
51
52         if event.type == pygame.QUIT:  # Gebeurtenis: het kruisje is aangeklikt
53             done = True  # Het spel moet eindigen dus we zetten done op True, zodat de loop straks stopt
54         elif event.type == pygame.KEYDOWN:
55             # Er is een toets ingedrukt, we kijken welke en ondernemen actie
56             if event.key == pygame.K_SPACE: #spatie
57                 print("Knop: Spatie")
```

Naast kliks op het kruisje, vangen we nu ook toetsaanslagen af (regel 54). Als een toetsaanslag gedetecteerd is, kijken we of de ingedrukte de spatiebalk is (regel 56) . Voor nu drukken we om dat te testen even een melding af op de console. (Naast het PyGame gamescherm is de console in Thonny ook nog actief en kun je dus gewoon print() commando's geven. Deze zijn heel handig om dingen te testen. Het slim printen van meldingen kan je goed helpen dingen te testen en foutjes in je code op te sporen.

Probeer 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 3 dingen doen:

- We geven de dobbelsteen een nieuwe random waarde tussen 1 en 6.
- We verzetten de pion die aan de beurt is het aantal stappen van de worp.
- We geven de beurt door, zodat bij de volgende spatiebalk de andere speler zal worden verzet.

Hieronder staat de code met commentaar:

```
50      for event in pygame.event.get(): #doorloop alle gebeurtenissen sinds de vorige schermupdate
51
52          if event.type == pygame.QUIT:  # Gebeurtenis: het kruisje is aangeklikt
53              done = True  # Het spel moet eindigen dus we zetten done op True, zodat de loop straks stopt
54          elif event.type == pygame.KEYDOWN:
55              # Er is een toets ingedrukt, we kijken welke en ondernemen actie
56              if event.key == pygame.K_SPACE: #spatie
57                  print("Knop: Spatie")
58
59                  worp = random.randint(1,6) # kies een random getal tussen 1 en 6 als dobbelsteenworp
60                  posities[beurt] += worp # verzet de pion die op dit moment aan de beurt is
61
62                  # geef de beurt aan de andere speler, zodat die hierna aan de beurt is
63                  if beurt == 0:
64                      beurt = 1
65                  else:
66                      beurt = 0
```

- Regel 59 genereert een nieuwe random dobbelsteenworp
- Op regel 60 verhogen we de positie van de speler die aan de beurt is met de waarde van de dobbelsteenworp. (Dus in de lijst `posities`, wordt één van de twee waardes verhoogd). 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 die variabele wijzigen, worden ze automatisch ergens anders getekend. Handig!
- Regel 63 t/m 66 geven de beurt door aan de andere speler (van 0 naar 1 of van 1 naar 0)

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

5.9 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 (je pionnen zijn eigenlijk "van het bord gevallen"). Dit moeten we voorkomen.

Als er een speler op of voorbij het laatste vakje (bij het standaard bord is dat 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. Als hij er voorbij is, zetten we hem gewoon terug op vakje 63.

Vervolgens geven we de beurt alleen door als er geen spelers op vakje 63 staan (dan heeft er nog niemand gewonnen en moet het spel verder). Als iemand wel op 63 staat, zal de beurt niet meer worden doorgegeven en stopt het spel.

We moeten de code van net dus nog wat verder aanpassen:

```
50      for event in pygame.event.get(): #doorloop alle gebeurtenissen sinds de vorige schermupdate
51
52          if event.type == pygame.QUIT:  # Gebeurtenis: het kruisje is aangeklikt
53              done = True  # Het spel moet eindigen dus we zetten done op True, zodat de loop straks stopt
54          elif event.type == pygame.KEYDOWN:
55              # Er is een toets ingedrukt, we kijken welke en ondernemen actie
56              if event.key == pygame.K_SPACE: #spatie
57                  print("Knop: Spatie")
58
59                  worp = random.randint(1,6) # kies een random getal tussen 1 en 6 als dobbelsteenworp
60                  posities[beurt] += worp # verzet de pion die op dit moment aan de beurt is
61
62                  # Is de pion op of voorbij het laatste vakje? Dan valt hij van het bord af!
63                  # Zet hem dan precies op het laatste vakje om dat te voorkomen:
64                  if posities[beurt] >= 63:
65                      posities[beurt] = 63
66
67                  # heeft de speler nog niet gewonnen? geef dan de beurt door aan de volgende speler:
68                  else:
69                      # geef de beurt aan de andere speler, zodat die hierna aan de beurt is
70                      if beurt == 0:
71                          beurt = 1
72                      else:
73                          beurt = 0
```

Op regel 64 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 crash).

Het doorgeven van de beurt zit nu in de `else` (regel 68) en wordt dus alleen uitgevoerd als de speler niet op vakje 63 staat.

Test het maar weer eens.

5.10 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 aan onze lijst met keyboard events:

```
50      for event in pygame.event.get(): #doorloop alle gebeurtenissen sinds de vorige schermupdate
51
52          if event.type == pygame.QUIT:  # Gebeurtenis: het kruisje is aangeklikt
53              done = True  # Het spel moet eindigen dus we zetten done op True, zodat de loop straks stopt
54          elif event.type == pygame.KEYDOWN:
55              # Er is een toets ingedrukt, we kijken welke en ondernemen actie
56              if event.key == pygame.K_SPACE: #spatie
57                  print("Knop: Spatie")
58
59                  worp = random.randint(1,6) # kies een random getal tussen 1 en 6 als dobbelsteenworp
60                  posities[beurt] += worp # verzet de pion die op dit moment aan de beurt is
61
62                  # Is de pion op of voorbij het laatste vakje? Dan valt hij van het bord af!
63                  # Zet hem dan precies op het laatste vakje om dat te voorkomen:
64                  if posities[beurt] >= 63:
65                      posities[beurt] = 63
66
67                  # heeft de speler nog niet gewonnen? geef dan de beurt door aan de volgende speler:
68                  else:
69                      # geef de beurt aan de andere speler, zodat die hierna aan de beurt is
70                      if beurt == 0:
71                          beurt = 1
72                      else:
73                          beurt = 0
74
75              elif event.key == pygame.K_BACKSPACE: #backspace
76                  print("Knop: Backspace")
77                  posities = [0,0] # zet de spelers elk weer terug op vakje 0
78                  beurt = 0 # geef de beurt aan speler 0
```

Je ziet dat we naast het afvangen van de spatie nu ook de backspace (regel 75) kunnen detecteren. Als deze wordt ingedrukt zetten we 2 variabelen terug naar hun beginwaarden. De posities van beide pionnen worden op 0 gezet (ze worden dus terug op het startvakje gezet) en we geven de beurt aan speler 0. Hiermee hebben we het spel gereset.

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

5.11 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 hebben we al en staat 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 daarvoor wat code toevoegen aan het "Teken de graphics" deel van onze hoofdloop:

```
79      # --- Teken de graphics voor de volgende schermupdate (nog buiten beeld) ---
80
81      screen.fill((255,255,255)) # begin met een witte achtergrond
82
83      bordrect = bord.get_rect() #vraag afmetingen (rectangle) van het bordplaatje op
84      screen.blit(bord, bordrect) # teken het bord op het volgende screen:
85
86      #teken pionnen als gekleurde cirkels op de coordinaten van de vakjes waar ze staan:
87      speler0_x = vakjes[posities[0]][0]  #x-coordinaat pion speler 0
88      speler0_y = vakjes[posities[0]][1]  #y-coordinaat pion speler 0
89      kleur0 = (0,255,0) # groen
90      pygame.draw.circle(screen, kleur0, (speler0_x, speler0_y), 10) # teken cirkel als pion speler 0
91      speler1_x = vakjes[posities[1]][0] + 5  #x-coordinaat pion speler 0
92      speler1_y = vakjes[posities[1]][1] + 5  #y-coordinaat pion speler 0
93      kleur1 = (0, 0, 255) # blauw
94      pygame.draw.circle(screen, kleur1, (speler1_x, speler1_y), 10)  # teken cirkel als pion speler 1
95
96      # We tekenen wat tekst op het scherm
97      # Kies het standaardfont en een lettergrootte (30):
98      myfont = pygame.font.SysFont(None, 30)
99
100     #teken de laatste worp op het scherm:
101     text ="Laatste worp: " + str(worp)
102     label = myfont.render(text, 1, (0, 0, 0))
103     screen.blit(label, (350, 470))
104
105     #teken de speler die aan de beurt is op het scherm:
106     text ="Aan de beurt: " + str(beurt + 1)
107     label = myfont.render(text, 1, (0, 0, 0))
108     screen.blit(label, (350, 495))
```

- Op regel 98 stellen we een lettergrootte in door een font object te maken
- Op regel 101 maken we een string die we willen afdrukken om de worp weer te geven
- Op regel 102 maken we een plaatje van de string uit regel 101 en kiezen zwart (0, 0, 0) als tekstkleur
- Op regel 103 voegen we het gemaakte plaatje uit regel 102 aan het nieuwe scherm toe

Regel 106 t/m 108 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, hupsakee! 😉

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!

6.1 THEMA

Dit is meestal je eerste stap. Elk goed bordspel heeft een thema dat de sfeer van het spel en de regels bepaalt. Denk bijvoorbeeld eens aan:

- Je favoriete film, boek, game of serie
- Rome/Griekenlandreis
- Reis om de wereld
- Een levensloop van brugklasser tot bovenbouw op het SGDB
- Een sport
- Een appeltaart die zijn weg naar de oven moet zien te vinden
- …

Je zorgt natuurlijk dat de graphics van je bord en pionnen (en eventuele andere extra's) bij je thema aansluiten en je verzint regels/uitbreidingen die ook aansluiten bij het thema.

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)
- Animaties van lopende pionnen en rollende dobbelstenen
- Muisbesturing met interface knoppen die je kunt indrukken
- Een scherm met de spelregels en het verhaal
- Opslaan en laden van lopende spellen
- ...
- ...

De mogelijkheden zijn eindeloos. Wt weet ook niet alle code en mogelijkheden uit zijn hoofd natuurlijk, maar op internet is van alles te vinden. De uitdaging is om iets tofs te bedenken en uit te zoeken hoe je het kunt bouwen.

6.4 PLAATJES ALS PIONNEN GEBRUIKEN

Het ligt voor de hand dat plaatjes mooier zijn om als pion te gebruiken dan de standaard gekleurde cirkels, zeker als je eenmaal een thema voor je bord hebt. Dit is gelukkig eenvoudig om te doen.

Ten eerste moet je de afbeeldingen van de pionnen inladen. Zorg dat je plaatjes maakt die klein genoeg zijn om als pion te gebruiken. Als je transparante delen in je plaatje hebt, sla deze dan op als .png bestanden (anders krijg je witte vlakjes er omheen)

Het inladen gaat precies hetzelfde als het laden van het plaatje van het ganzenbord en het ligt voor de hand het daar bij te zetten bovenaan in de code:

```
# --------- Globale variabelen --------

# bord en pion afbeeldingen:
bord = pygame.image.load("ganzenbord.png")
pion1 = pygame.image.load("pion1.png")
pion2 = pygame.image.load("pion2.png")
```

Om nu de cirkels te vervangen door de afbeeldingen, ga je naar de plek in de code waar de cirkels worden getekend. Bijvoorbeeld voor speler 1 was de code dit:

```
#teken pionnen als gekleurde cirkels op de coordinaten van de vakjes waar ze staan:
speler0_x = vakjes[posities[0]][0]; #x-coordinaat pion speler 0
speler0_y = vakjes[posities[0]][1]; #y-coordinaat pion speler 0
kleur0 = (0,255,0) # groen
pygame.draw.circle(screen, kleur0, (speler0_x, speler0_y), 10)
```

En dat veranderen we hierin (de kleur is niet meer nodig en we tekenen nu de ingeladen pion in plaats van de cirkel. Hiervoor zijn de coördinaten nog steeds nodig natuurlijk):

```
speler0_x = vakjes[posities[0]][0]; #x-coordinaat pion speler 0
speler0_y = vakjes[posities[0]][1]; #y-coordinaat pion speler 0
screen.blit(pion1, (speler0_x, speler0_y)) #Teken de pion op het scherm
```

6.5 MEER SOORTEN INPUT

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

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

Je ziet hier codes voor het detecteren van verschillende toetsen op het toetsenbord (een lijst met alle toetscodes vind je in de pygame documentatie)

Verder worden muiskliks gedetecteerd en wordt opgeslagen wat de precieze coördinaten (mouseX en mouseY) van de aangeklikte pixel zijn.

6.6 VERPLAATSVAKJES EN BEURTEN OVERSLAAN

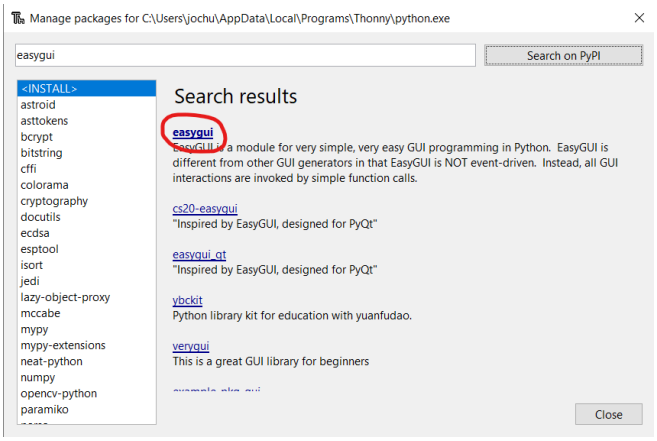
De twee meest gebruikte uitbreidingen zijn het toevoegen van “verplaatsvakjes” (dus vakjes die als je er op komt de pion vooruit of achteruit zetten naar een ander vakje) en “beurten overslaan”. Dit zijn gewoon hele voor de hand liggende uitbreidingen die in bijna elk thema wat je maar kiest voor je spel een plekje kunnen krijgen.

Omdat dit de meest gevraagd uitbreidingen zijn heeft Wt er een uitlegfilmpje over gemaakt.

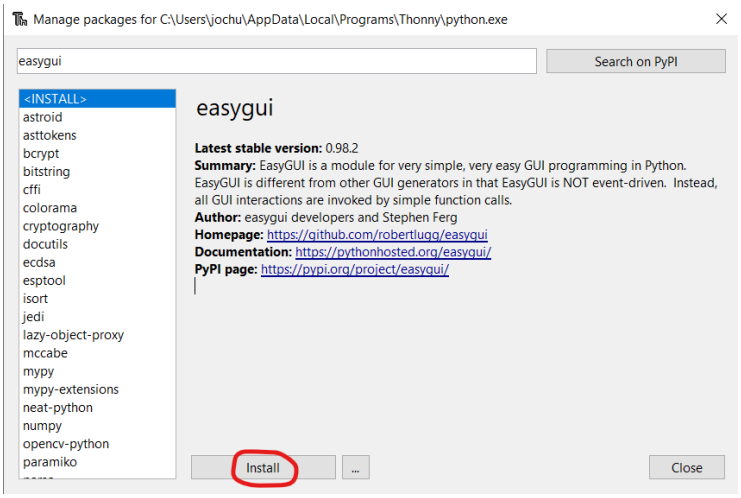
6.7 POP-UPS EN ANDERE VENSTERS

Nu we dankzij PyGame een grafisch venster hebben, is het niet meer zo handig om de gebruiker om invoer te vragen via het gebruikelijke `input()` commando. Hiervoor moet hij namelijk de console gebruiken en die zit waarschijnlijk verborgen achter het gamescherm en steeds wisselen is onhandig.

Gelukkig zijn er libraries (lang leve python, er zijn echt libraries voor alles) waarmee je eenvoudig popup vensters kunt maken voor het vragen van input, het aanbieden van keuzes, het doen van meldingen, etc. De gemakkelijkste library hiervoor is EasyGUI en die kunnen we even installeren via de Package Manager. Zoek hierin op EasyGUI en klik hem aan:



Klik op de volgende pagina op de “install” knop:



Om nu de easyGUI library te gebruiken in je code, moeten we hem bovenaan je code importeren met:

```
import easygui
```

In de volgende paragrafen laat ik een paar mogelijkheden van easyGUI zien, maar er zijn er een heleboel. Een overzicht van alle mogelijkheden vind je hier:

- Tutorial: https://easygui.readthedocs.io/en/master/tutorial.html
- Overzicht van alle functies: https://easygui.readthedocs.io/en/master/api.html

6.7.1 EEN MEDEDELING IN EEN POPUP

Als je de gebruiker op de hoogte wilt stellen van een gebeurtenis, kan een popup met een melding handig zijn. Dat kan met de messagebox functie van easyGUI. Deze heeft 3 strings als argument nodig om te werken:

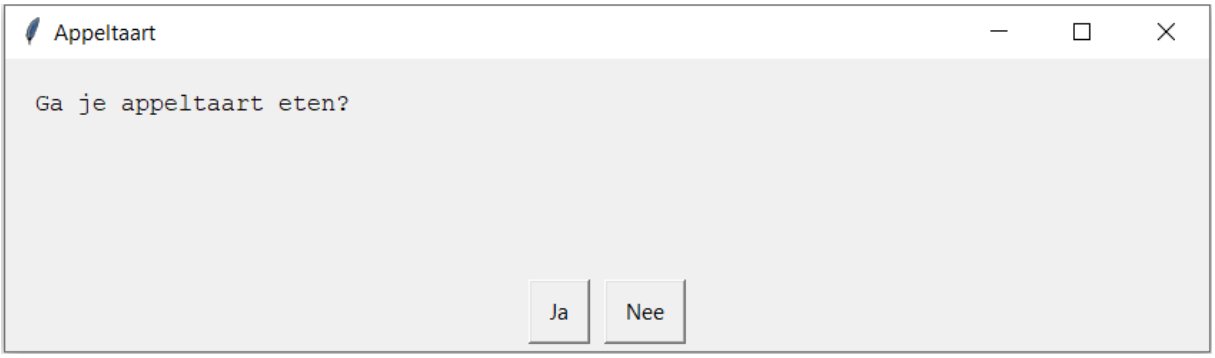
```
easygui.msgbox("Ik ben de tekst van het bericht", "Ik ben de titelbalk", "Knoptekst")
```

Dit resulteert in de volgende popup:

6.7.2 JA/NEE VRAGEN MET EASYGUI

Soms wil je de gebruiker een vraag stellen waarop je simpelweg een ja of nee als antwoord wilt. Dat kan met de ynbox functie. Deze heeft 3 argumenten nodig om te werken: Een string als vraag, een string als titel en een list met 2 strings als labels voor de knoppen. De popup geeft een returnwaarde van True (als het eerste knopje is ingedrukt) of False (als het tweede knopje is ingedrukt). Deze kun je opslaan in een variabele, zodat je op basis van de keuze van de gebruiker iets kunt doen:

```
antwoord = easygui.ynbox("Ga je appeltaart eten?", "Appeltaart", ["Ja", "Nee"])
```



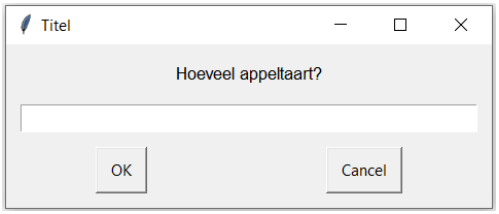
Je kunt nu bijvoorbeeld met een `if` beslissen wat je doet op basis van het antwoord:

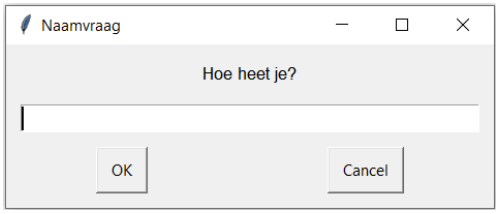
```
if antwoord == True:
    print("Jummie!")
else:
    print("Welke idioot eet er geen appeltaart als hij de kans heeft??!?")
```

6.7.3 DE GEBRUIKER IETS LATEN INTYPEN MET EASYGUI

Als je de gebruiker wat wilt laten invoeren, kun je de functies integerbox en enterbox gebruiken. Die geven je respectievelijk een integer en een string terug:

```
aantal = easygui.integerbox('Hoeveel appeltaart?', 'Titel')
naam = easygui.enterbox('Hoe heet je', 'Titel')
```





6.7.4 MULTIPLE CHOICE MET EASYGUI

Als je de gebruiker wilt laten kiezen uit meerdere opties is een buttonbox een goede optie. Deze heeft een 3 argumenten nodig; een string voor de vraag, een string voor de titel en een list met strings voor de verschillende knoppen. Als returnwaarde krijg je aangeklikte string terug:

```
antw = easygui.buttonbox('Favoriete snack?', 'Snack', ['Taart', 'Worst', 'Ei'])
```

6.8 MUZIEK EN GELUID

Pygame maakt het toevoegen van muziek en geluid gemakkelijk. Om geluid te gebruiken moet je de volgende init toevoegen (net na pygame.init):

```
pygame.mixer.init()
```

Voor muziek:

Omdat er maar 1 muziekje tegelijk kan spelen, is het starten van muziek simpel te doen door:

```
pygame.mixer.music.load("music.wav") # music.wav is bestand (.mp3 en .ogg kan ook)
pygame.mixer.music.play(-1) # -1 voor een oneindige herhaling
pygame.mixer.music.set_volume(0.2) #getal tussen 0.0 en 1.0 voor volume
```

Voor geluid:

Elk afzonderlijk geluid moet worden in geladen en apart worden gestart op het juiste moment (er kunnen meerdere geluiden door elkaar afspelen als dat nodig is):

```
#laad de geluiden (buiten de main loop, net als het laden van afbeeldingen):
sndDobbelsteen = pygame.mixer.Sound("dice.wav")
sndKlap = pygame.mixer.Sound("boom.wav")

Speel een geluid af (tijdens de game):
```

```
pygame.mixer.Sound.play(sndDobbelsteen)
```

Meer info in de documentatie van Pygame:

https://www.pygame.org/docs/ref/mixer.html

6.9 MEERDERE PADEN/ROUTES OVER HET BORD

Bij het standaardbord zitten alle vakjes netjes achter elkaar. Een veelgevraagde uitbreiding is de mogelijkheid om meerdere paden te kiezen over het bord. Dit is best complex, omdat je nu niet meer zomaar de pion kunt verzetten, maar steeds moet checken of je een kruising passeert en dan de gebruiker moet vragen wat hij wil. Het vergt ook wat slim rekenwerk om dan steeds op het juiste vakje uit te komen.

Wt heeft een YouTube filmpje gemaakt met uitleg hoe je dit aan zou kunnen pakken:

Ganzenbord meerdere paden - YouTube

Opmerking: in het filmpje gebruikt Wt een andere library voor de popups (dus niet de easyGUI library zoals hierboven beschreven. In een oudere versie van dit opdrachtboekje gebruikte Wt TkInter, dus dit filmpje is ook daarmee opgenomen). Je moet dus zelf even de relevante stukjes vervangen door easyGUI popups, maar dat is niet lastig.