Polymorphism

Bas Terwijn

February 16, 2026

1 Introductie

Dit document behandelt polymorphism binnen Object Oriented Programming (OOP).

1.1 Doelen

Bekend raken met:

- gebruik van inheritance voor polymorphism
- gebruik van compositie voor polymorphism

2 Computerspel met polymorphism

In de `game_composition.py` opdracht maken we gebruik van de `pygame` package om een eenvoudig computer spel te maken. Installeer `pygame` als je dat nog niet gedaan hebt met:

```
$ pip install gtts pygame
```

In onderstaande `assignments/game.py` file wordt eem scherm aangemaakt waarin units worden gecreëerd. Vervolgens wordt een while-loop gestart welke in iedere tijdstap van het spel:

- de events afhandelt (bijv. sluiten van het scherm)
- het scherm leeg gemaakt
- elke unit een stap laat maken en tekent
- het scherm update

Elke unit is een object van de `Unit` class. Deze class heeft verschillende methode om een unit te initializeren, tekenen en bewegen.

assignments/game.py

```python
import pygame
import random

FRAMES_PER_SECOND = 60

def main():
    # initialize pygame window
    pygame.init()
    screen = pygame.display.set_mode((800, 600), pygame.RESIZABLE)
    pygame.display.set_caption("Polymorphism Game Example")

    # create units
    nr_units = 12
    units = [Unit(screen) for _ in range(nr_units)]
```

```
    running = True
    while running:  # run until the user closes the window

        # handle events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:  # user closed the window
                running = False

        # clear the screen
        screen = pygame.display.get_surface()
        screen.fill((0, 0, 0))

        # step and plot each unit
        for unit in units:
            unit.step(screen)
            unit.plot(screen)

        pygame.display.flip()  # update the screen
        pygame.time.Clock().tick(FRAMES_PER_SECOND) # limit frame rate


class Unit:

    def __init__(self, screen):
        """ Initialize a unit """
        self.init_position(screen)
        maxspeed = 5
        self.dx = random.random() * maxspeed * 2 - maxspeed
        self.dy = random.random() * maxspeed * 2 - maxspeed
        self.color = (0, 255, 0) # RGB: Red, Green, Blue
        self.radius = 10

    def init_position(self, screen):
        """ Initialize position of a unit on the screen. """
        self.x = random.randint(0, screen.get_width())
        self.y = random.randint(0, screen.get_height())

    def plot(self, screen):
        """ Plot a unit """
        pygame.draw.circle(screen, self.color, (self.x, self.y), self.radius, 0)

    def step(self, screen):
        """ Take a step: move and possibly other actions. """
        self.move(screen)
        self.handle_border(screen)

    def move(self, screen):
        """ Move a unit """
        self.x += self.dx
        self.y += self.dy

    def handle_border(self, screen):
        """ Handle a unit's movement at the edges of the screen. """
        if self.x < 0 or self.x > screen.get_width():
            self.dx = -self.dx
        self.x = max(0, min(self.x, screen.get_width()))
        if self.y < 0 or self.y > screen.get_height():
```

```
            self.dy = -self.dy
        self.y = max(0, min(self.y, screen.get_height()))


if __name__ == "__main__":
    main()
```

Als we deze code uitvoeren zien we een scherm met Twaalf groene units die een random beginpositie (x,y) en random snelheid (dx, dy) krijgen, zoals weergegeven in figuur 1, en die bewegen en tegen de randen van het scherm botsen.

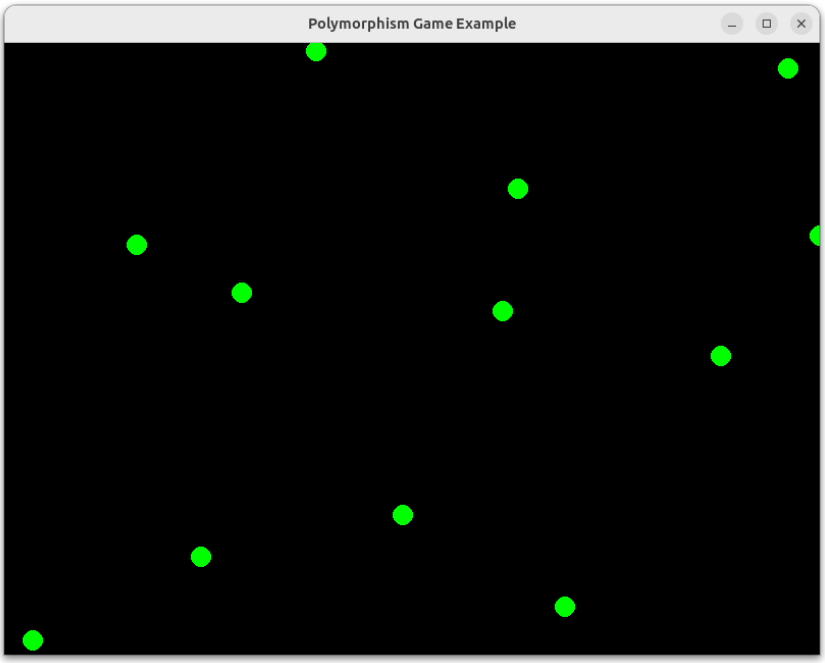

Figure 1: Twaalf groene units.

Lees deze code door zodat je deze begrijpt. Experimenteer eventueel door kleine aanpassingen te maken en de code opnieuw uit te voeren.

2.1 Verschillende soorten units

We willen nu verschillende soorten units maken welke zich verschillend gedragen terwijl we deze units wel op dezelfde manier willen gebruiken. Voor dit doel maken we gebruik van polymorhisme. Voor de implementatie van polymorphime maken we eerst gebruik van inheritance zoals weergeven in figuur 2.

We maken hierbij eerst onderscheid tussen een unit welke een random beginpositie heeft, `Unit_Random_Start`, en een units welke het midden van het scherm als beginpositie heeft, `Unit_Center_Start`.

Voor elk van deze soorten units maken we vervolgens weer een onderscheid tussen een unit welke van de randen van het scherm botst, eindigend op `_Bounce`, en een unit welke door de randen van het scherm heen gaat en aan de andere kant weer verschijnt, eindigend op `_Wraparound`.

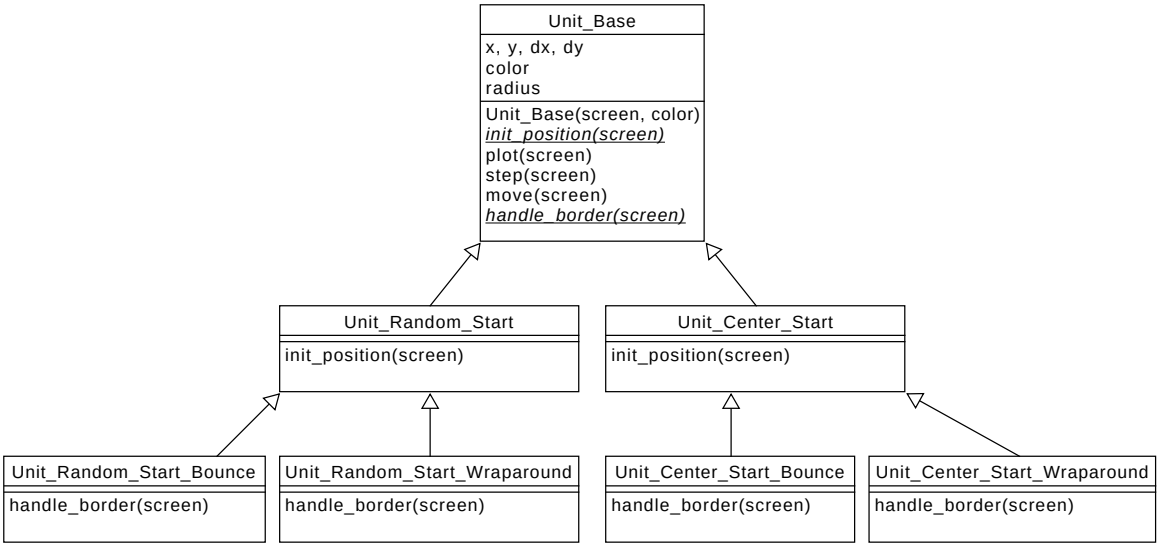

Figure 2: UML class diagram van verschillende Units classes.

We kunnen nu verschillende soorten units maken door deze verschillende classes te gebruiken met hetzelfde interface. Deze code is al geimplementeerd in `assignments/game_inheritance.py`. Voer deze code uit en lees deze code door zodat je deze begrijpt. Experimenteer eventueel door kleine aanpassingen te maken en de code opnieuw uit te voeren.

3 Opdracht `game_composition.py`

We zijn achteraf gezien toch niet blij met polymorphime door gebruik van inheritance. Het is nu lastig om nieuwe soorten units toe te voegen omdat we dan steeds nieuwe classes moeten maken. Ook hebben we nu de `Bounce` en `Wraparound` functionaliteit twee keer geïmplementeerd, één keer voor de `Unit_Start_Random` en één keer voor de `Unit_Start_Center`.

We gaan daarom nu polymorphism op een moderne manier door gebruik van compositie implementeren zoals weergegeven in figuur 3. We hebben nu nog maar één `Unit` class welke een `Position_Strategy` object en een `Border_Strategy` object krijgt. In de subclasses van deze strategie objecten moet dan de specifieke functionaliteit komen te staan voor het bepalen van de beginpositie en het gedrag bij de randen van het scherm.

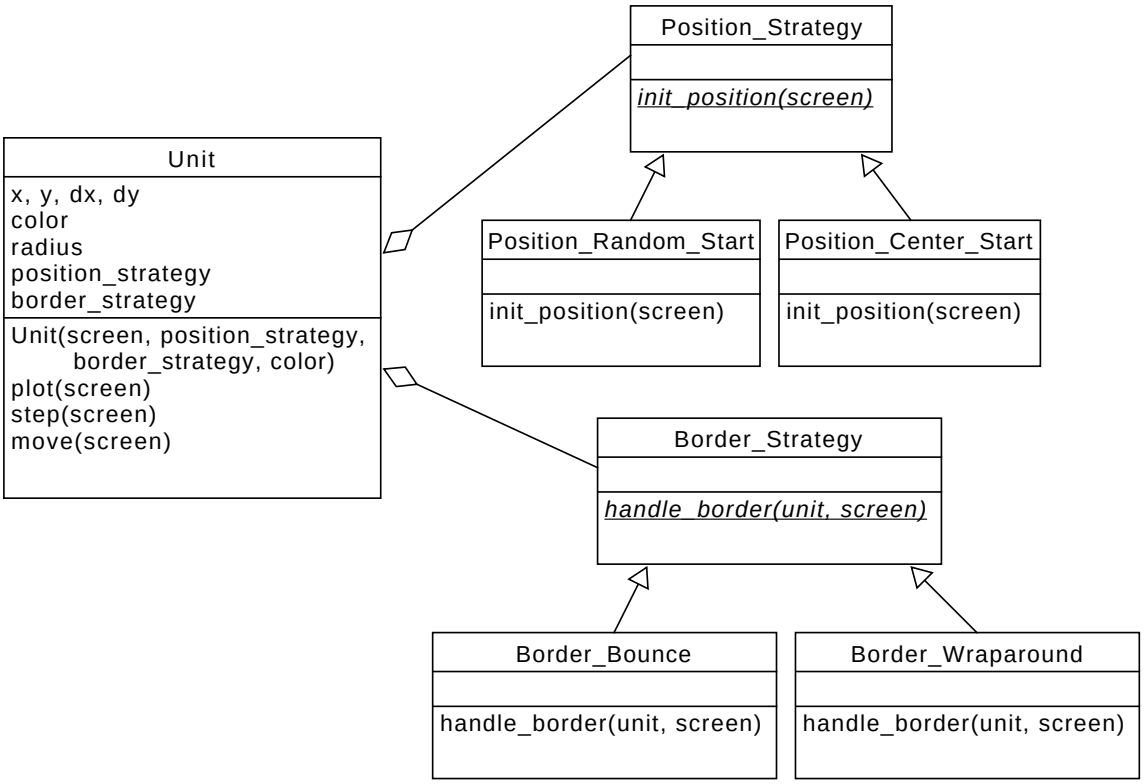

Figure 3: UML class diagram van de Units met verschillende Strategy classes.

Implementeer nu de classes:

- `Position_Random_Start` welke de beginpositie op een random positie binnen het scherm zet.
- `Position_Center_Start` welke de beginpositie in het midden van het scherm zet.
- `Border_Bounce` welke ervoor zorgt dat de unit van de randen van het scherm afstuitert.
- `Border_Wraparound` welke ervoor zorgt dat de unit door de randen van het scherm heen gaat en aan de andere kant weer verschijnt.

en pas de `Unit` class aan zodat het deze strategie objecten opslaat en op de juiste manier gebruikt zodat het resultaat hetzelfde is als bij de `game_inheritance.py` code.

Test deze opdracht met:

```
$ pytest test_game_composition.py
```