Pong
Python Spill – Prosjektrapport

Marcus Allen Denslow

February 3, 2026

Contents

Chapter 1

Introduksjon

Denne rapporten dokumenterer utviklingen av et Pong-spill skrevet i Python med PyGame-biblioteket. Pong er et klassisk arkadespill fra 1972 der to spillere styrer hver sin paddle for å slå en ball frem og tilbake. Spillet egner seg godt som programmeringsprosjekt fordi det krever håndtering av brukerinput, kollisjonsdeteksjon, poengberegning og sanntids grafikkrendering.

Prosjektet er bygget med objektorientert programmering (OOP), der spillobjektene er modellert som egne klasser med tydelig ansvarsfordeling. Koden er fordelt på flere filer for å oppnå god modulstruktur, og alle klasser og metoder er dokumentert med docstrings etter PEP 257-standarden. I tillegg er det skrevet enhetstester med pytest for å verifisere at spillogikken fungerer korrekt.

Rapporten er strukturert som følger: kapittel 2 beskriver kravspesifikasjonen, klassediagrammet og flytskjemaet. Kapittel 3 diskuterer implementasjonen, testingen og mulige forbedringer. Kapittel 4 oppsummerer prosjektet.

Chapter 2

Beskrivelse

2.1 Kravspesifikasjon

Følgende funksjonelle krav ble satt for spillet:

- **To spillere:** Spiller 1 styrer venstre paddle med tastene W og S. Spiller 2 styrer høyre paddle med piltastene opp og ned.
- **Ball:** En ball beveger seg over skjermen og spretter av topp- og bunnveggene.
- **Kollisjon:** Når ballen treffer en paddle, snur den horisontal retning.
- **Poeng:** Dersom ballen passerer en spillers side, får motstanderen ett poeng.
- **Vinner:** Den første spilleren som når 10 poeng vinner runden.
- **Restart:** Etter at en spiller har vunnet, kan man trykke SPACE for å starte en ny runde.
- **Avslutt:** Spillet kan avsluttes med ESC eller ved å lukke vinduet.

I tillegg ble det stilt følgende ikke-funksjonelle krav:

- Spillet skal kjøre i 60 bilder per sekund (FPS).
- Skjermoppløsningen er 800×600 piksler.
- Alle spillkonstanter (hastigheter, størrelser, farger) skal være konfigurerbare fra én sentral fil.

2.2 Klassediagram (UML)

Figur 2.1 viser UML-klassediagrammet for programmet. Programmet består av tre hovedklasser: `Game`, `Paddle` og `Ball`. `Game`-klassen har en komposisjonsrelasjon til de to andre klassene – den oppretter og eier to `Paddle`-objekter og ett `Ball`-objekt.

2.3 Flytskjema

Figur 2.3 viser flyten i programmet fra oppstart til avslutning. Hovedspilløkken kjører kontinuerlig og håndterer hendelser, oppdaterer spilltilstand og tegner alle elementer til skjermen for hvert bilde.

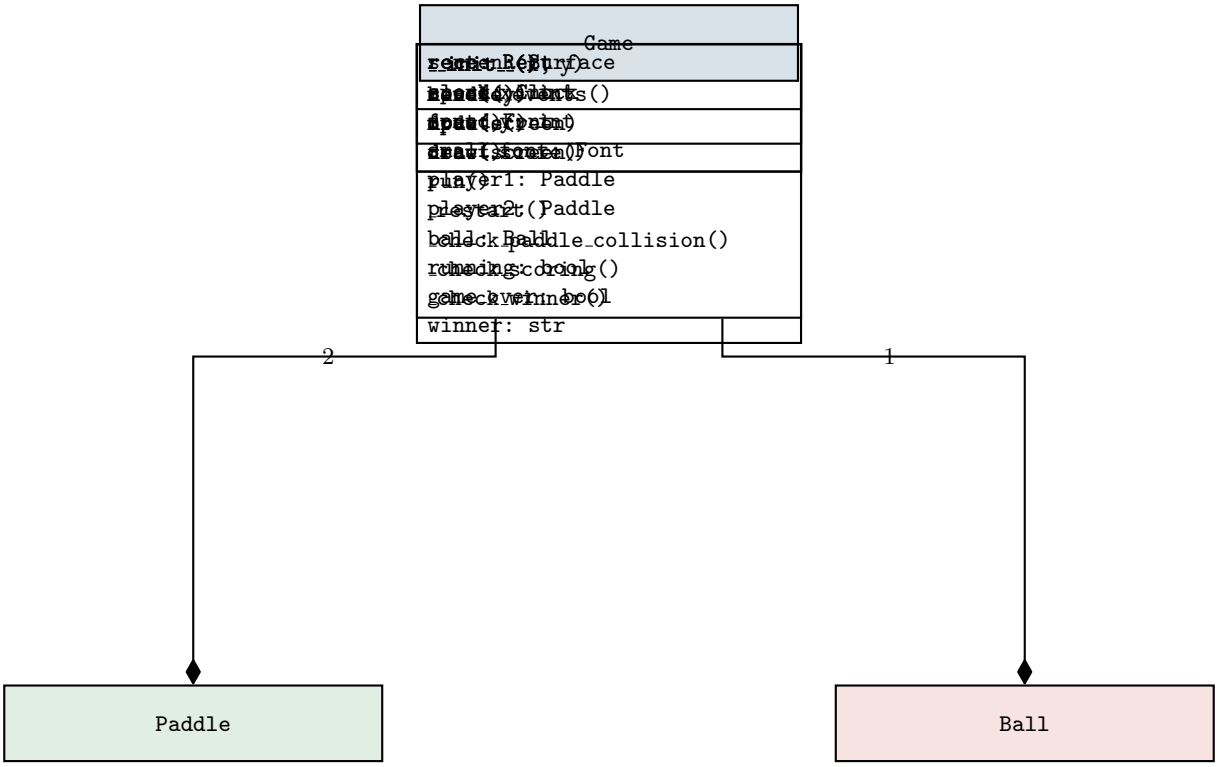

Figure 2.1: UML-klassediagram for Pong. Diamantpilene indikerer komposisjon.

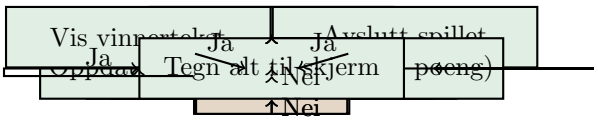

Figure 2.2: Flytskjema for hovedspilløkken i Pong.

Chapter 3

Diskusjon og Resultater

3.1 Programstruktur

Prosjektet er organisert i seks filer, der hver fil har et tydelig ansvarsområde:

Fil	Beskrivelse
`main.py`	Inngangspunkt. Oppretter et `Game`-objekt og kaller `run()`.
`settings.py`	Sentral konfigurasjonsfil med alle konstanter: skjermstørrelse, farger, hastigheter, paddle- og ballstørrelser, vinnerpoeng og FPS.
`paddle.py`	Inneholder `Paddle`-klassen som håndterer posisjon, bevegelse, tegning og poeng.
`ball.py`	Inneholder `Ball`-klassen som håndterer posisjon, bevegelse, spretting og tilbakestilling.
`game.py`	Inneholder `Game`-klassen med spilløkke, hendelseshåndtering, kollisjonslogikk, poengberegning og rendering.
`test_game.py`	Enhetstester for `Paddle` og `Ball` med pytest.

Table 3.1: Oversikt over filstrukturen.

Denne strukturen gjør at hver klasse kan utvikles og testes uavhengig. Dersom man ønsker å endre spillkonstanter, trenger man bare å redigere `settings.py` – resten av koden henter verdiene derfra.

3.2 Objektorientert design

Programmet er bygget rundt tre klasser som samarbeider gjennom komposisjon:

Paddle-klassen representerer en spillerpaddle. Den har et `pygame.Rect`-objekt for posisjon og størrelse, samt en `score`-variabel. Metoden `move(dy)` flytter padlen vertikalt og sørger for at den ikke beveger seg utenfor skjermen. Metoden `draw(screen)` tegner padlen, og `reset_score()` nullstiller poengsummen.

Ball-klassen representerer ballen. Den har et `rect`-objekt samt `speed_x` og `speed_y` for hastighet i begge retninger. Metoden `move()` flytter ballen og inverterer `speed_y` dersom ballen treffer topp- eller bunnkanten. Metoden `reset()` plasserer ballen tilbake i midten av skjermen.

Game-klassen er hovedklassen som binder alt sammen. Den oppretter to `Paddle`-objekter og ett `Ball`-objekt i konstruktøren. Spilløkken i `run()` kaller tre metoder i sekvens for hvert bilde:

1. `handle_events()` – leser tastaturinput og vindushendelser.
2. `update()` – oppdaterer padleposisjon basert på tastene som holdes inne, flytter ballen, sjekker kollisjoner og poeng.
3. `draw()` – tegner bakgrunn, midtlinje, poengsum, padler, ball og eventuelt vinnertekst.

Kollisjonsdeteksjon gjøres med Pygames innebygde `colliderect()`-metode, som sjekker om ballens rektangel overlapper med en paddles rektangel. Det sjekkes også at ballen beveger seg mot den aktuelle padlen (via `speed_x`), slik at ballen ikke fester seg i padlen.

3.3 Enhetstesting

Prosjektet inneholder 11 enhetstester skrevet med pytest, fordelt på to testklasser: `TestPaddle` (6 tester) og `TestBall` (5 tester).

For `Paddle` testes:

- At padlen opprettes med riktig posisjon og poengsum lik 0.
- At `move()` flytter padlen korrekt opp og ned.
- At padlen stoppes ved toppen og bunnen av skjermen (grenseverdier).
- At `reset_score()` nullstiller poengsummen.

For `Ball` testes:

- At ballen starter i midten av skjermen.
- At `reset()` plasserer ballen tilbake i sentrum.
- At `move()` endrer posisjon korrekt basert på hastighet.
- At ballen spretter av toppen (inverterer `speed_y`).
- At ballen spretter av bunnen (inverterer `speed_y`).

Et eksempel på en test:

```
def test_stopp_ved_toppen(self):
    paddle = Paddle(30, 5)
    paddle.move(-20)
    assert paddle.rect.top == 0
```

Listing 3.1: Test for at padlen stoppes ved toppen av skjermen.

Alle 11 tester bestod uten feil.

Hvorfor er testing viktig?

Testing er viktig fordi det gjør det mulig å oppdage feil tidlig i utviklingsprosessen, før de påvirker resten av programmet. Enhetstester verifiserer at hver enkelt komponent fungerer isolert, noe som gjør det tryggere å gjøre endringer uten å introdusere nye feil. I dette prosjektet sikrer testene for eksempel at padlene aldri beveger seg utenfor skjermen og at ballen alltid tilbakestilles til riktig posisjon. Uten tester måtte man manuelt teste all funksjonalitet ved å spille gjennom spillet hver gang man endrer koden, noe som er tidkrevende og upålitelig.

3.4 Alternative løsninger og forbedringer

Under utviklingen ble det vurdert noen alternative løsninger:

- **AI-motstander:** Spillet kunne hatt en enspillermodus der datamaskinen styrer den ene padlen. Dette kunne implementeres ved å la padlen følge ballens y-posisjon med en viss forsinkelse for å gjøre det utfordrende.
- **Vinkelbasert spretting:** I nåværende implementasjon spretter ballen alltid med samme vinkel. En forbedring ville vært å endre utgangsvinkel basert på hvor på padlen ballen treffer – nær kanten gir skarpere vinkel, nær midten gir rettere bane.
- **Lydeffekter:** Pygame støtter avspilling av lyd. Man kunne lagt til lyder for kollisjoner og poeng for en mer engasjerende spillopplevelse.
- **Menysystem:** En startmeny med valg for å starte spill, endre innstillinger eller avslutte ville forbedret brukervennligheten.

Disse forbedringene ble ikke implementert da hovedfokuset var å lage et fungerende spill med god objektorientert struktur og kodeorganisering.

Chapter 4

Konklusjon

Prosjektet resulterte i et fungerende Pong-spill for to spillere, skrevet i Python med PyGame. Spillet oppfyller alle kravene i kravspesifikasjonen: spillerne kan styre padlene, ballen spretter korrekt, poeng telles og en vinner kåres ved 10 poeng.

Koden er organisert med objektorientert design fordelt på seks filer, der hver klasse har tydelig ansvar. Alle klasser og metoder er dokumentert med docstrings etter PEP 257. Enhetstestene med pytest bekrefter at kjernelogikken i `Paddle` og `Ball` fungerer som forventet.

Gjennom prosjektet er det oppnådd praktisk erfaring med objektorientert programmering, modulær filstruktur, GUI-utvikling med PyGame, UML-modellering og automatisert testing.

Referanser

- Pygame dokumentasjon: https://www.pygame.org/docs/
- Python dokumentasjon: https://docs.python.org/3/
- PEP 257 – Docstring Conventions: https://peps.python.org/pep-0257/
- pytest dokumentasjon: https://docs.pytest.org/

Appendix A

Vedlegg – Kildekode

A.1 settings.py

```
"""Spillinnstillinger og konstanter for Pong."""

# Skjermstorrelse
WIDTH = 800
HEIGHT = 600

# Farger
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# Paddle-innstillinger
PADDLE_WIDTH = 10
PADDLE_HEIGHT = 100
PADDLE_SPEED = 6

# Ball-innstillinger
BALL_SIZE = 15
BALL_SPEED_X = 5
BALL_SPEED_Y = 5

# Spillregler
WINNING_SCORE = 10

# FPS
FPS = 60
```

Listing A.1: settings.py – Spillkonstanter.

A.2 paddle.py

```
"""Modul for Paddle-klassen."""

import pygame
from settings import PADDLE_WIDTH, PADDLE_HEIGHT, HEIGHT, WHITE


class Paddle:
    """Representerer en spiller-paddle i Pong.

    paddle kan bevege seg oppover innenfor grensene
```

```
    av skjermen og teller spillerens poeng

    Attributes:
        rect: Pygame-rektangel som definerer padlens
              posisjon og storrelse.
        score: Spillerens naavaerende poengsum.
    """

    def __init__(self, x, y):
        """Opprett en ny paddle.

        Args:
            x: Horisontal startposisjon (piksler fra venstre).
            y: Vertikal startposisjon (piksler fra toppen).
        """
        self.rect = pygame.Rect(x, y, PADDLE_WIDTH,
                                PADDLE_HEIGHT)
        self.score = 0

    def move(self, dy):
        """Flytt padlen vertikalt.

        Padlen holdes innenfor skjermens grenser.

        Args:
            dy: Antall piksler aa flytte
                (negativt = opp, positivt = ned).
        """
        self.rect.y += dy
        if self.rect.top < 0:
            self.rect.top = 0
        if self.rect.bottom > HEIGHT:
            self.rect.bottom = HEIGHT

    def draw(self, screen):
        """Tegn padlen paa skjermen.

        Args:
            screen: Pygame-overflaten aa tegne paa.
        """
        pygame.draw.rect(screen, WHITE, self.rect)

    def reset_score(self):
        """Nullstill spillerens poengsum."""
        self.score = 0
```

Listing A.2: paddle.py – Paddle-klassen.

A.3 ball.py

```
"""Modul for Ball-klassen."""

import pygame
from settings import (WIDTH, HEIGHT, BALL_SIZE,
                      BALL_SPEED_X, BALL_SPEED_Y, WHITE)


class Ball:
    """Representerer ballen i Pong.
```

```
10
11      Ballen beveger seg over skjermen og spretter av
12      vegger og padler.
13
14      Attributes:
15          rect: Pygame-rektangel som definerer ballens
16                posisjon og storrelse.
17          speed_x: Horisontal hastighet (piksler per frame).
18          speed_y: Vertikal hastighet (piksler per frame).
19      """
20
21      def __init__(self):
22          """Opprett en ny ball i midten av skjermen."""
23          self.rect = None
24          self.speed_x = 0
25          self.speed_y = 0
26          self.reset()
27
28      def reset(self):
29          """Tilbakestill ballen til midten av skjermen."""
30          self.rect = pygame.Rect(
31              WIDTH // 2 - BALL_SIZE // 2,
32              HEIGHT // 2 - BALL_SIZE // 2,
33              BALL_SIZE,
34              BALL_SIZE,
35          )
36          self.speed_x = BALL_SPEED_X
37          self.speed_y = BALL_SPEED_Y
38
39      def move(self):
40          """Flytt ballen basert paa naavarende hastighet.
41
42          Ballen spretter av toppen og bunnen av skjermen
43          ved aa invertere den vertikale hastigheten.
44          """
45          self.rect.x += self.speed_x
46          self.rect.y += self.speed_y
47
48          if self.rect.top <= 0 or self.rect.bottom >= HEIGHT:
49              self.speed_y *= -1
50
51      def draw(self, screen):
52          """Tegn ballen paa skjermen.
53
54          Args:
55              screen: Pygame-overflaten aa tegne paa.
56          """
57          pygame.draw.rect(screen, WHITE, self.rect)
```

Listing A.3: ball.py – Ball-klassen.

A.4 game.py

```
1  """Modul for Game-klassen som styrer hovedspillokken."""
2
3  import pygame
4  import sys
5  from settings import (
6      WIDTH, HEIGHT, WHITE, BLACK,
```

```
    PADDLE_WIDTH, PADDLE_HEIGHT, PADDLE_SPEED,
    BALL_SPEED_X, WINNING_SCORE, FPS,
)
from paddle import Paddle
from ball import Ball


class Game:
    """Hovedklassen som styrer Pong-spillet.

    Haandterer spillokke, input, kollisjon, poengberegning
    og rendering av alle spillelementer.

    Attributes:
        screen: Pygame-overflaten som spillet tegnes paa.
        clock: Pygame-klokke for aa styre fps.
        player1: Venstre spillers paddle.
        player2: Hoyre spillers paddle.
        game_over: Om spillet er ferdig.
        winner: Navnet paa vinneren, eller None.
    """

    def __init__(self):
        """Initialiser Pygame og opprett alle spillobjekter."""
        pygame.init()
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        pygame.display.set_caption("Pong - 2 Spillere")
        self.clock = pygame.time.Clock()
        self.font = pygame.font.Font(None, 74)
        self.small_font = pygame.font.Font(None, 36)

        self.player1 = Paddle(
            30, HEIGHT // 2 - PADDLE_HEIGHT // 2)
        self.player2 = Paddle(
            WIDTH - 30 - PADDLE_WIDTH,
            HEIGHT // 2 - PADDLE_HEIGHT // 2)
        self.ball = Ball()

        self.running = True
        self.game_over = False
        self.winner = None

    def handle_events(self):
        """Haandter brukerinput fra tastatur."""
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    self.running = False
                if self.game_over \
                        and event.key == pygame.K_SPACE:
                    self._restart()

    def _restart(self):
        """Tilbakestill spillet for en ny runde."""
        self.player1.reset_score()
        self.player2.reset_score()
        self.ball.reset()
        self.game_over = False
        self.winner = None
```

```
68
69      def update(self):
70          """Oppdater spilltilstand."""
71          if self.game_over:
72              return
73
74          keys = pygame.key.get_pressed()
75          if keys[pygame.K_w]:
76              self.player1.move(-PADDLE_SPEED)
77          if keys[pygame.K_s]:
78              self.player1.move(PADDLE_SPEED)
79          if keys[pygame.K_UP]:
80              self.player2.move(-PADDLE_SPEED)
81          if keys[pygame.K_DOWN]:
82              self.player2.move(PADDLE_SPEED)
83
84          self.ball.move()
85          self._check_paddle_collision()
86          self._check_scoring()
87          self._check_winner()
88
89      def _check_paddle_collision(self):
90          """Sjekk kollisjon mellom ball og padler."""
91          if self.ball.rect.colliderect(self.player1.rect) \
92                  and self.ball.speed_x < 0:
93              self.ball.speed_x *= -1
94              self.ball.rect.left = self.player1.rect.right
95          if self.ball.rect.colliderect(self.player2.rect) \
96                  and self.ball.speed_x > 0:
97              self.ball.speed_x *= -1
98              self.ball.rect.right = self.player2.rect.left
99
100     def _check_scoring(self):
101         """Sjekk om ballen har passert en paddle."""
102         if self.ball.rect.left <= 0:
103             self.player2.score += 1
104             self.ball.reset()
105             self.ball.speed_x = -BALL_SPEED_X
106         if self.ball.rect.right >= WIDTH:
107             self.player1.score += 1
108             self.ball.reset()
109
110     def _check_winner(self):
111         """Sjekk om en spiller har vunnet."""
112         if self.player1.score >= WINNING_SCORE:
113             self.game_over = True
114             self.winner = "Spiller 1"
115         elif self.player2.score >= WINNING_SCORE:
116             self.game_over = True
117             self.winner = "Spiller 2"
118
119     def draw(self):
120         """Tegn alle spillelementer paa skjermen."""
121         self.screen.fill(BLACK)
122         self.player1.draw(self.screen)
123         self.player2.draw(self.screen)
124         self.ball.draw(self.screen)
125         if self.game_over:
126             self._draw_game_over()
127         pygame.display.flip()
128
```

```
    def run(self):
        """Kjoer hovedspillokken."""
        while self.running:
            self.handle_events()
            self.update()
            self.draw()
            self.clock.tick(FPS)
        pygame.quit()
        sys.exit()
```

Listing A.4: game.py – Game-klassen (hovedspillokke).

A.5 main.py

```
"""Inngangspunkt for Pong-spillet.

Kjoer dette skriptet for aa starte spillet:
    python main.py
"""

from game import Game


def main():
    """Opprett og start Pong-spillet."""
    game = Game()
    game.run()


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

Listing A.5: main.py – Inngangspunkt.

A.6 test_game.py

```
"""Enhetstester for Pong-spillet.

Kjoer testene med:
    python -m pytest test_game.py
"""

import pygame
import pytest
from settings import (HEIGHT, PADDLE_HEIGHT,
                      WIDTH, BALL_SIZE, BALL_SPEED_Y)
from paddle import Paddle
from ball import Ball


@pytest.fixture(autouse=True)
def init_pygame():
    """Initialiser Pygame foer hver test."""
    pygame.init()
    yield
    pygame.quit()

```

```
22
23 class TestPaddle:
24     """Tester for Paddle-klassen."""
25
26     def test_opprett_paddle(self):
27         paddle = Paddle(30, 200)
28         assert paddle.rect.x == 30
29         assert paddle.rect.y == 200
30         assert paddle.score == 0
31
32     def test_flytt_opp(self):
33         paddle = Paddle(30, 200)
34         paddle.move(-10)
35         assert paddle.rect.y == 190
36
37     def test_flytt_ned(self):
38         paddle = Paddle(30, 200)
39         paddle.move(10)
40         assert paddle.rect.y == 210
41
42     def test_stopp_ved_toppen(self):
43         paddle = Paddle(30, 5)
44         paddle.move(-20)
45         assert paddle.rect.top == 0
46
47     def test_stopp_ved_bunnen(self):
48         paddle = Paddle(30, HEIGHT - PADDLE_HEIGHT - 5)
49         paddle.move(20)
50         assert paddle.rect.bottom == HEIGHT
51
52     def test_reset_score(self):
53         paddle = Paddle(30, 200)
54         paddle.score = 5
55         paddle.reset_score()
56         assert paddle.score == 0
57
58
59 class TestBall:
60     """Tester for Ball-klassen."""
61
62     def test_opprett_ball(self):
63         ball = Ball()
64         assert ball.rect.centerx == WIDTH // 2
65         assert ball.rect.centery == HEIGHT // 2
66
67     def test_reset(self):
68         ball = Ball()
69         ball.rect.x = 100
70         ball.rect.y = 100
71         ball.reset()
72         assert ball.rect.centerx == WIDTH // 2
73         assert ball.rect.centery == HEIGHT // 2
74
75     def test_move(self):
76         ball = Ball()
77         start_x = ball.rect.x
78         start_y = ball.rect.y
79         ball.move()
80         assert ball.rect.x == start_x + ball.speed_x
81         assert ball.rect.y == start_y + ball.speed_y
82
```

```
83      def test_sprett_av_topp(self):
84          ball = Ball()
85          ball.rect.top = 1
86          ball.speed_y = -BALL_SPEED_Y
87          ball.move()
88          assert ball.speed_y == BALL_SPEED_Y
89
90      def test_sprett_av_bunn(self):
91          ball = Ball()
92          ball.rect.bottom = HEIGHT - 1
93          ball.speed_y = BALL_SPEED_Y
94          ball.move()
95          assert ball.speed_y == -BALL_SPEED_Y
```

Listing A.6: test_game.py – Enhetstester.