



Gegevens student:

Naam: Lodewijk (L.D.C.) Aris

Student nummer: 530159

Adres:

Delfshavenseweg 23
3043CB
Rotterdam

Opleiding:

Software developer [25998]

Kwalificatiedossier:

ICT (ICT support, IT Systems and Devices, Software development)

Gegevens examinatoren/trainers:

Bart (B.J.E.) Wijenberg
Mark (M.J.M.) Dackus
Vista College

Henri Dunantstraat 2
6419PB
Heerlen

Inhoudsopgave

CHANGE RECORD	3
1 INLEIDING	4
2 TEST CASES	6
3 TEST PLAN.....	10
3.1 Testomgeving.....	10
3.1.1 Aannames en beperkingen.....	11
3.1.2 Gebruikte python modules:	11
3.1.3 Testopzet en testprocedure.....	12
3.2 Testdata, ontwerp en scenario's	14
3.2.1 Code onder Test TC1	14
3.2.2 De Testcode voor TC1.....	14
3.2.3 Code onder Test TC2.....	15
3.2.4 De Testcode voor TC2	15
3.2.5 Code onder Test TC3.....	16
3.2.6 De Testcode voor TC3	16
3.2.7 Code onder Test TC4.....	17
3.2.8 De Testcode voor TC4	18
3.2.9 Code onder Test TC5.....	19
3.2.10 De Testcode voor TC5	19
3.2.11 Code onder Test TC6.....	20
3.2.12 De Testcode voor TC6	20
3.2.13 Code onder Test TC7.....	21
3.2.14 De Testcode voor TC7	21
3.2.15 Code onder Test TC8	22
3.2.16 De Testcode voor TC8.....	23
3.2.17 Code onder Test TC9.....	25
3.2.18 De Testcode voor TC9	25
3.2.19 Code onder Test TC10.....	25
3.2.20 De Testcode voor TC10	25
3.2.21 Code onder Test TC11	25
3.2.22 De Testcode voor TC11.....	25
4 TESTRAPPORT	26
4.1 TC1 Spel initialisatie Test resultaten	26
4.1.1 Testprocedure TC1 Spel initialisatie:	26
4.2 TC2 Spel Start Test resultaten.....	27
4.2.1 Testprocedure TC2 Spel Start:.....	27
4.3 TC3 Hit Test resultaten	28
4.3.1 Testprocedure TC3 Hit:	28
4.4 TC4 Stand Test resultaten	29
4.4.1 Testprocedure TC4 Stand:	29
4.5 TC5 count hands Test resultaten.....	30
4.5.1 Testprocedure TC5 count hands:.....	30
4.6 TC6 BlackJack Test resultaten	31
4.6.1 Testprocedure TC6 BlackJack:	31
4.7 TC7 kapot Test resultaten.....	32
4.7.1 Testprocedure TC7 kapot:.....	32
4.8 TC8 Dealer Turn Test resultaten.....	33
4.8.1 Testprocedure TC8 Dealer Turn:.....	33
4.9 TC9 Split Test resultaten	34

4.9.1 Testprocedure TC9 Split:	34
4.10TC10 Game Outcome Test resultaten.....	35
4.10.1 Testprocedure TC10 Game Outcome:	35
4.11 TC11 Edge Case Test resultaten	39
4.11.1 Testprocedure TC11 Edge Case:.....	39
4.12 Overzicht testresultaten en conclusies van de uitgevoerde tests	41
5 VERBETERVOORSTELLEN	43
5.1 De verbeteringsvoorstellen a.d.h.v. gebruikerservaringen	43
5.2 De verbeteringsvoorstellen a.d.h.v. testervaringen	44
ANNEX A BLACKJACK CODE.....	47
ANNEX B CODE VERBETERINGEN.....	49
ANNEX C BEVINDINGEN.....	54

CHANGE RECORD

Versie	Datum	Change Log
1.0	16-02-2025	Eerste (initiële) versie.
2.0	10-03-2025	Na interne reviews zijn de volgende veranderingen aangebracht. Hoofdstuk 1 Definities en typos. Hoofdstuk 2: Enkele beschrijvingen in Test Cases Hoofdstuk 4.6 TC6 BlackJack Test resultaten Hoofdstuk 5 Verbetervoorstellen Annex C Bevindingen: Tina Slot en Corné Brouwer

1 INLEIDING

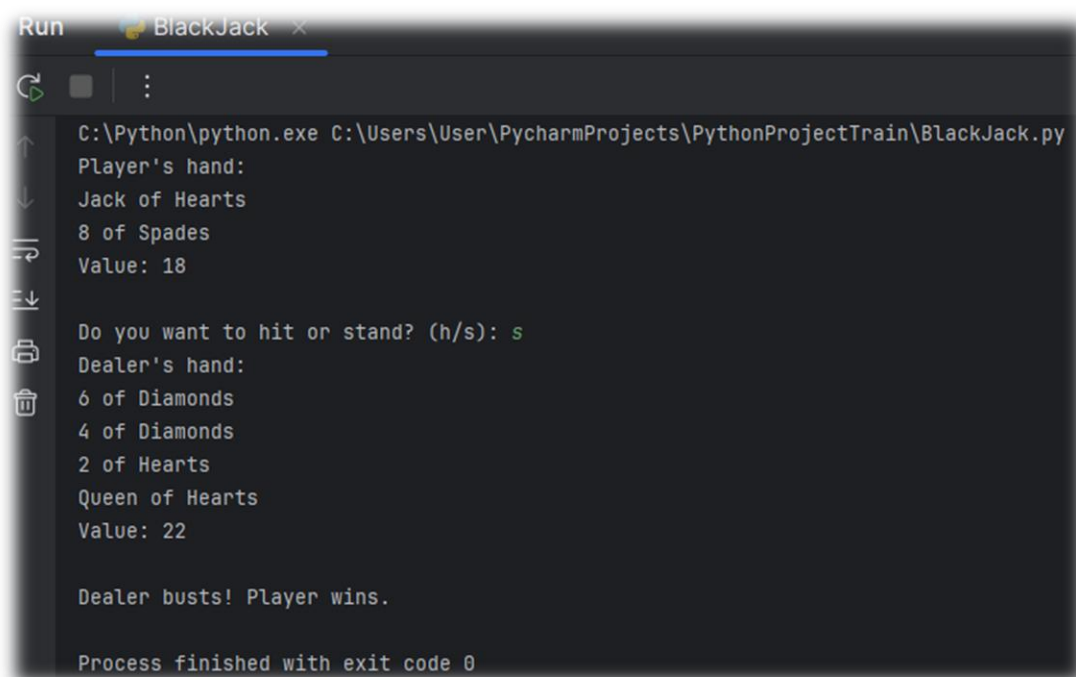
In november 2024 kreeg ik via het UWV de kans om een werkervaringsplaats bij Flanderijn in Rotterdam te vervullen. Hier biedt een IT-straat, georganiseerd in samenwerking met Stichting BEE-Ideas, een uitdagende omgeving waar Services & Operations-taken worden uitgevoerd. Deze IT-gerelateerde werkzaamheden omvatten het beheren van laptops, smartphones en iPads, het aanmaken van accounts, het verwerken van invoergegevens van opdrachtgevers en het autoriseren van toegang tot diverse applicaties. Daarnaast ondersteun ik de helpdesk bij het oplossen van storingen en het afhandelen van aanvragen via Topdesk binnen Flanderijn.

Voor mijn opdracht heb ik gekozen voor softwareontwikkeling en functioneel testen. Tijdens de cursus van het Vista College besloot ik mij verder te verdiepen in de programmeertaal Python, omdat deze mij aanspreekt.

In dit examenportfolio ben ik aan de slag gegaan om een klein programma op internet te vinden dat ik kan gebruiken voor mijn test- en verbeteringsopdracht. Om de opdracht behapbaar te houden heb ik de twee criteria voor het programma gesteld en zijn:

- a) Minder dan 200 regels code
- b) Minder dan 10 functionaliteiten

Omdat ik in het verleden wel eens blackjack heb gespeeld, zocht ik naar een simpel blackjackprogramma geschreven in Python. Op GitHub met behulp van GPT-4o (Azure OpenAI) vond ik dit programmaatje van een eenvoudige blackjackcode dat ik kan testen en bovendien voldoet aan de criteria. Ik kopieerde de code naar de PyCharm-omgeving om het blackjackspel te spelen en voerde deze uit. Het werkt, zie Figuur 1.



```
Run BlackJack x
C:\Python\python.exe C:\Users\User\PycharmProjects\PythonProjectTrain\BlackJack.py
Player's hand:
Jack of Hearts
8 of Spades
Value: 18

Do you want to hit or stand? (h/s): s
Dealer's hand:
6 of Diamonds
4 of Diamonds
2 of Hearts
Queen of Hearts
Value: 22

Dealer busts! Player wins.

Process finished with exit code 0
```

Figuur 1 Voorbeeld van BlackJack.py spelverloop

Dit programmaatje BlackJack.py dient nu als uitgangspunt voor mijn test- en verbeteropdrachten. Het testplan en test cases zijn te vinden in de onderstaande paragrafen. Ook bevat dit document een testrapportage (Testrapport), waarin elke testcase uitvoering wordt besproken. Tot slot is er een hoofdstuk met verbetervoorstellen (zie Verbetervoorstellen) en zijn sommige verbetervoorstellen verder uitgewerkt in Annex B Code Verbeteringen.

Het programmaatje BlackJack.py werkt dus naar behoren. Bij het inspecteren van de code ontdekte ik dat de code van het programmaatje redelijk overzichtelijk is. De code is 100 regels en is te vinden in de bijlage Annex A BlackJack Code en bestaat uit de volgende vijf (hoofd)functionaliteiten:

- | | |
|-------------------------|------------------------------------|
| a) create_deck | (creëer een dek kaarten) |
| b) calculate_hand_value | (reken de waarde uit van een hand) |
| c) display_hand | (laat de hand zien) |
| d) deal_card | (deel een kaart uit) |
| e) play_black_jack | (speel het blackjack spel) |

Ik heb ervoor gekozen om testcases toe te passen op de BlackJack.py code (zie Test Cases), de functionaliteiten te testen en te bepalen welke aanbevelingen of verbeteringen ik op basis van deze testen kan doen.

De code en testcode zijn in GitHub te vinden.

<https://github.com/lodewijkaris/PythonProjectTrain/tree/master> en zijn openbaar.

Definities:

A = Ace	Aas
h = hit	geef nog een kaart aan speler.
J = Jack	Boer
K = King	Heer
Patch	Het aanpassen van een object in Python. Vaak gebruikt om iets te simuleren voor testuitvoer.
Q= Queen	Dame
s = stand	de speler past.
Test-driven	Test-driven ontwikkeling is een software-ontwikkelmethode waarbij je testen schrijft voordat je de daadwerkelijke code implementeert.
unittest	ingebouwde module in Python die ondersteuning biedt voor het schrijven en uitvoeren van code testen.

2 TEST CASES

Voor het bepalen van de testcases is geanalyseerd waar het BlackJack spel aan zou moeten voldoen. Belangrijk opmerking is dat tijdens deze analyse er een aantal beperkingen van toepassing zijn voor het uitvoeren van de functionaliteiten die bij een completer blackjack spel van toepassing zouden moeten zijn, zie hiervoor 3.1.1 Aannames en beperkingen. Aan de volgende criteria moet het spel voldoen:

- Het blackjack spel wordt correct opgestart/geïnitieerd.
- De kaarten worden correct uitgedeeld aan speler en deler.
- Het blackjack spel handelt de actie “nog een kaart aan speler” goed af.
- Het blackjack spel handelt de actie “pass” goed af.
- Het blackjack spel rekent de handen van speler en deler goed uit.
- Het blackjack spel wordt gespeeld volgens de spelregels:
 - Het spel herkent blackjack
 - Deler neemt kaart indien <17 punten
 - Speler kan splitsen
- Het blackjack spel detecteert als de speler of deler “dood” gaat.
- Het blackjack spel herkent de uitkomst van het spel (wie er gewonnen of verloren heeft of dat het spel is geëindigd in een gelijk spel).
- Het spel is robuust.

Hieruit zijn de volgende test cases gedefinieerd voor het testen van het spel BlackJack.py

Tabel 1 Test Cases definitie

Test Case	Test Code Functie(s)	Beschrijving	Stappen	Verwacht resultaat
1 Spel initialisatie	create_deck	Genereer het dek kaarten. Een dek kaarten bestaat uit 52 kaarten.	Check dat de dek kaarten is gecreëerd en geschud. Check dat de speler en dealer met een lege hand beginnen.	Het spel start met een dek kaarten Het dek is geschud Lege handen voor speler en deler

Test Case	Test Code Functie(s)	Beschrijving	Stappen	Verwacht resultaat
2 Spel Start	play_blackjack	Uitdelen van de eerste 2 kaarten	Deel kaarten uit aan speler en dealer	Speler en deler hebben 2 kaarten Tweede kaart van de deler is verborgen
3 Hit	deal_card	Speler heeft twee kaarten in de hand en speler kan een kaart nemen	Deel 1 kaart uit aan speler in de test Deel nog twee kaarten uit aan speler test	Kaart wordt toegevoegd aan de spelers hand en weggehaald van het dek: Speler heeft 3 kaarten, dek nog 47 Speler heeft nu 5 kaarten en dek nog 45
4 Stand	play_blackjack: choice loop	Speler kiest om te passen na 2 hits	Deel geen kaart meer uit aan speler na hit, hit, stand (h,h,s)	Speler is in staat om te passen Aantal kaarten aan speler is 4 en er wordt geen nieuwe kaart toegevoegd
5 Count Hand(s)	calculate_hand_value	Spel berekent de waarde van het aantal punten op de hand	Bereken de waarde van de kaarten: a) 2,3,4 (no_aces_no_tens) b) J,3,K (with_face_cards) c) A,3,A (with_aces) d) A,A,10 (with_ace_adjustment) e) A,A,9,A,10 (busting_with_aces)	De waarde van de handen zijn correct uitgerekend (ook met in achtname van een Aas, deze is 1 of 11 punten) a) 9 b) 23 c) 15 d) 12 e) 22
6 BlackJack		Het spel herkent blackjack	Speler heeft een aas en een plaatje of een tien (blackjack)	Check dat het spel blackjack herkent (speler wint meteen)

Test Case	Test Code Functie(s)	Beschrijving	Stappen	Verwacht resultaat
7 Kapot	calculate_hand_value	Het spel herkent als de speler of deler meer dan 21 punten heeft	Speler heeft 3 kaarten samen meer dan 21 punten Speler heeft 4 kaarten samen meer dan 21 punten Deler heeft 3 kaarten samen meer dan 21 punten Deler heeft 5 kaarten samen meer dan 21 punten Deler heeft 4 kaarten en 1 plaatje meer dan 21 punten	Het spel herkent dat de waarde van de hand > 21 is: 22 22 22 23 28
8 Dealer Turn	create_deck deal_card calculate_hand_value	Het spel wordt uitgevoerd volgens de regels; de deler laat zijn twee kaarten zien en de deler neemt kaarten zolang <17 punten	Deler heeft 17 punten met 2 kaarten Deler heeft 19 punten met 2 kaarten Deler heeft 14 punten met 2 kaarten Deler heeft 16 punten met 2 kaarten en neemt kaart (7 harten) Deler heeft 15 punten met 2 kaarten en neemt kaart (10 ruiten) Deler heeft 2 kaarten met 1 aas en 17 punten Deler heeft 2 kaarten met 1 aas en 16 punten en neemt 1 kaart (klaver Aas)	De deler past en blijft op 17 De deler past en blijft op 19 De deler neemt nog 1 willekeurige kaart De deler neemt 1 specifieke kaart (en gaat kapot) De deler neemt 1 specifieke kaart (en gaat kapot) De deler past De deler neemt 1 specifieke kaart en past op 17
9 Split		Het spel laat splitsen toe	Speler heeft twee 7-s Speler heeft twee 2-en	Het spel laat splitsen toe en gaat verder met de eerste 7 Het spel laat splitsen toe en gaat verder met de eerste 2

Test Case	Test Code Functie(s)	Beschrijving	Stappen	Verwacht resultaat
			en ontvangt weer een 2 op de eerste 2 Deler heeft 2 azen	Het spel laat weer splitsen toe en gaat verder met de eerste 2 De speler kan de azen splitsen en krijgt dan op beide azen 1 kaart
10 Game Outcome	Alle functies: create_deck deal_card calculate_hand_value display_hand play_blackjack	Het spel concludeert wie er heeft gewonnen of verloren in verschillende scenario's. Dit is een systeem test	Speel net zoveel verschillende spellen dat 1) de speler wint 2) de deler wint 3) het gelijk spel is	De uitkomst van ieder spel wordt correct geconcludeerd
11 Edge Case	play_blackjack	Het spel kan met uitzonderlijk geval omgaan waarbij h en s worden opgegeven	Speler kiest om "hs" in te typen en enter	Spel vraagt om opnieuw in te vullen, hs is geen goede input

De Test Cases zijn verder in PyCharm ontwikkeld en zijn te vinden in de subhoofdstukken van het Testdata hoofdstuk ("Code onder Test TC#" en "De Testcode voor TC#"). Voor het testen van de code is voornamelijk gebruik gemaakt van de "assertequal" or "assertgreater" functies in de unittests. Er is specifiek voor test TC 4 een patch functie gebruikt. De resultaten van de test cases staan in het hoofdstuk Testrapport.

3 TEST PLAN

Voor het testen van de code is gebruik gemaakt van de Python-module unittest. Het is belangrijk om de testconfiguraties en de specifieke beperkingen van het BlackJack-spel te begrijpen voordat de testen worden uitgevoerd.

Er is gekozen om twee Python-versies te gebruiken voor de testen: Python 3.8.10 en Python 3.13. De keuze voor deze versies is arbitrair. Python 3.8.10 is de laatste onderhoudsversie van Python 3.8, en Python 3.13.1 was de nieuwste versie op het moment dat de cursus begon.

Python 3.8.10 is gekoppeld aan het Wing Personal-platform ([Wing Python IDE - Designed for Python](#)), terwijl Python 3.13 is gekoppeld aan het PyCharm-platform ([PyCharm: the Python IDE for data science and web development](#)). Beide platforms voldoen aan de testvereisten, maar PyCharm biedt een meer aanpasbare en uitgebreide ontwikkelomgeving, met geavanceerde code-analyse en ondersteuning voor meerdere programmeertalen. PyCharm is daarom gekozen voor de softwareontwikkeling. WingIDE richt zich meer op eenvoud en efficiëntie, biedt een goede debugger en heeft toegewijde ondersteuning voor Python, waardoor het geschikt is voor testen.

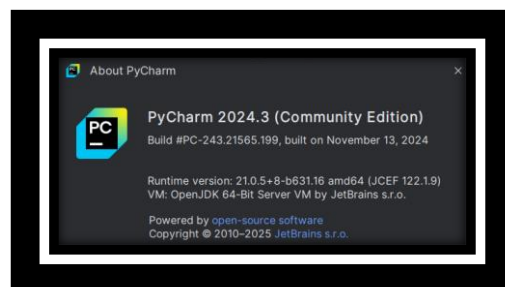
3.1 Testomgeving

De testscenario's worden uitgevoerd op de volgende configuraties en tools:

- **Besturingssysteem:** Windows 11
- **Python-versies:**
 - Python 3.8.10 (tags/v3.8.10:3d8993a, 3 mei 2021, 11:48:03)
 - Python 3.13.1 (tags/v3.13.1:0671451, 3 december 2024, 19:06:28)
- **Testtools: unittest van Python**

De testtools worden uitgevoerd binnen twee geïntegreerde ontwikkelomgevingen (IDE's):

- **Platform 1:** PyCharm voor softwareontwikkeling en testen.



De Python interpreter moet worden gezet op: Python 3.13

- **Platform 2:** Wing Python IDE voor softwaretesten (alleen).



- **Python-versie kan worden gecontroleerd met de volgende code:**

```
import sys
print(sys.version)
```

Commands execute without debug. Use arrow keys for history.

```
Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license()" for more information.
```

3.1.1 Aannames en beperkingen

De volgende aannames en beperkingen zijn van toepassing voor het definiëren van geldige testcases:

- a) Er bestaan verschillende varianten van het blackjackspel. Voor het opstellen van de tests zijn de regels van Meneer Casino gekozen:
<https://meneercasino.com/online-casino-tips/blackjack-regels-en-uitleg>.
- b) De code is ontworpen voor één enkele speler. Testscenario's voor meerdere spelers worden niet uitgevoerd, aangezien dit niet relevant is voor het spel in deze opdracht.
- c) Er wordt gewerkt met slechts één deck van kaarten, dus er zijn 52 kaarten in totaal.
- d) Bij de start van een nieuw spel wordt het volledige deck opnieuw geschud, en wordt er geen deel van het deck bewaard voor volgende rondes. Deze feature wordt niet getest in de scenario's.
- e) De speler heeft geen beginsaldo, en het spel wordt in één enkele ronde gespeeld.

3.1.2 Gebruikte python modules:

Voor het doen van de testen zijn twee python modules gebruikt:

- Module gebruikt in spel is *random* voor het willekeurig kiezen van kaarten. Deze moet worden geïmporteerd: Zie bijvoorbeeld Line 1 in code (Annex A BlackJack Code) :

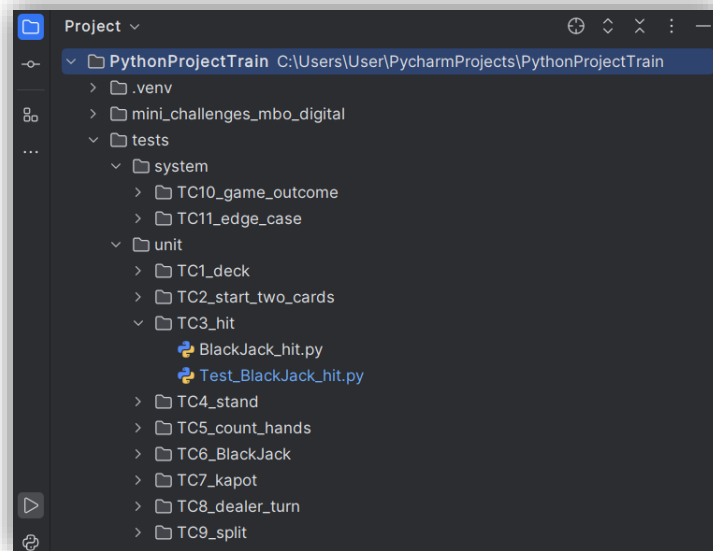
```
import random
```

- Module voor testen in python wordt *unittest* gebruikt en wordt geïmporteerd in de test code:

```
import unittest
```

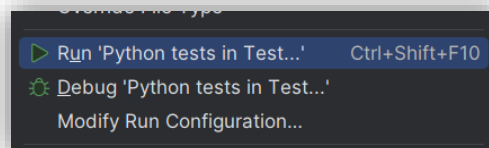
3.1.3 Testopzet en testprocedure

De ontwikkeling van de testcode is gedaan in PyCharm. Er is voor gekozen om voor iedere testcase een aparte directory te maken waarin de testcode en het te testen deel (van de code) in staat, zie Figuur 2. Iedere directory bevat twee files: de `BlackJack_{test_case}.py` file en de `Test_BlackJack_{test_case}.py` file. Het merendeel van de testen zijn unittests, deze bevinden zich in folder `tests/unit/`. Voor iedere testcase (TC#) is een subdirectory gemaakt. Ditzelfde is van toepassing op de system tests.



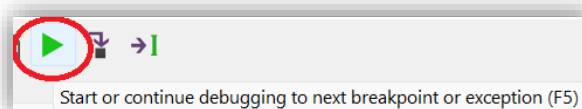
Figuur 2 opzet van de testcases in de project directory

De testcase kan dan in de testcase directory automatisch worden uitgevoerd. Voor het uitvoeren van de test kan dan in PyCharm gebruik worden gemaakt van de (test)Run functie, zie Figuur 3.



Figuur 3 Run Test in PyCharm

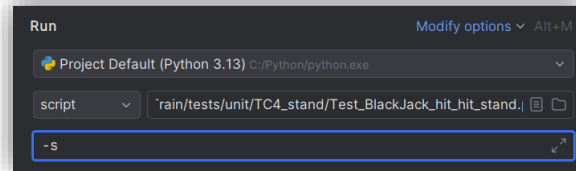
Zo'n zelfde Run functie is ook ter beschikking in Wing Python, zie Figuur 4.



Figuur 4 Run Test in Wing

Voor sommige testen was het nodig delen van de BlackJack.py code over te nemen om de testcases te kunnen uitvoeren. In de uitvoering is er deelcode geplaatst in dezelfde directory als de `test_` code file om de functies te kunnen testen.

Voor het slagen van sommige testen was het bovendien nodig de optie `-s` te gebruiken om gebruikers inputs toe te staan. Dit kan in PyCharm worden ingesteld met de `-s` optie, zie Figuur 5.



Figuur 5 Optie om te test uit te voeren met `-s` functie

Voor welke test case dit noodzakelijk is zal in de testprocedure worden aangegeven. Nadat de testen goed zijn verlopen in PyCharm, zijn de testen herhaald met Wing Python IDE.

3.2 Testdata, ontwerp en scenario's

Voor het doen van de testen zijn er altijd twee files nodig. De code met de functionaliteiten en de testcode. Voor iedere testcase is een stukje code geschreven, die afgeleid is van de hele blackjack.py code en een specifieke testcode, die de functie(s) test(en).

3.2.1 Code onder Test TC1

Het stukje code onder test is BlackJack_spel_initilisatie.py en bevat de functie create_deck() en is voldoende om het spel te initialiseren:

```
#####
#Lodewijk Aris version 1, date 31 jan 2025                                     #
#This sourcecode is part of the code BlackJack and covers Test Case Spel initialisatie #
#####

import random
# Function to create a deck of cards
def create_deck():
    deck = []
    suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']
    for suit in suits:
        for rank in ranks:
            deck.append((rank, suit))
    return deck
```

3.2.2 De Testcode voor TC1

De code om de functionaliteit te testen heet Test_BlackJack_spel_initilisatie.py en bevat scenario's waarin wordt gekeken dat:

- 1) Alle kaarten in het dek zijn,
- 2) De kaarten zijn geschud
- 3) Speler en deler lege handen hebben

```
#####
#Lodewijk Aris version 1, date 31 jan 2025                                     #
#This Test code belongs to Test Case Spel initialisatie                         #
#####

import unittest

from BlackJack_spel_initilisatie import create_deck

import random

class TestBlackjackTC1(unittest.TestCase):

    def test_create_deck(self):
        deck = create_deck()
        self.assertEqual(len(deck), 52) # Check dat het dek 52 kaarten heeft
        suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
        ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']
        expected_deck = [(rank, suit) for suit in suits for rank in ranks]
        self.assertEqual(sorted(deck), sorted(expected_deck)) # Check dat alle kaarten in het
dek zijn

    def test_deck_shuffled(self):
        deck = create_deck()
        shuffled_deck = create_deck()
        random.shuffle(shuffled_deck)
        self.assertNotEqual(deck, shuffled_deck, "Het dek moet zijn geschud.")

    def test_initial_hands_empty(self):
```

```

        player_hand = []
        dealer_hand = []

        self.assertEqual(len(player_hand), 0, "Spelers hand is leeg bij aanvang.")
        self.assertEqual(len(dealer_hand), 0, "Delers hand is leeg bij aanvang.")

if __name__ == '__main__':
    unittest.main()

```

3.2.3 Code onder Test TC2

Het stukje code onder test is BlackJack_Spel_Start.py en bevat de functies create_deck(), deal_card() en play_blackjack() en is voldoende om het spel te starten:

```

#####
#Lodewijk Aris version 1, date 1 feb 2025
#This sourcecode is part of the code BlackJack and covers Test Case Spel Start
#####
import random

# Function to create a deck of cards
def create_deck():
    deck = []
    suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']
    for suit in suits:
        for rank in ranks:
            deck.append((rank, suit))
    return deck

# Function to deal a card
def deal_card(deck, hand):
    card = random.choice(deck)
    deck.remove(card)
    hand.append(card)

def play_blackjack():
    deck = create_deck()
    random.shuffle(deck)

    player_hand = []
    dealer_hand = []
    # number_of_items_player = []
    # number_of_items_dealer = []
    for _ in range(2):
        deal_card(deck, player_hand)
        deal_card(deck, dealer_hand)
        continue
    # Return the hands for testing
    return player_hand, dealer_hand
# Run the game
play_blackjack()

```

3.2.4 De Testcode voor TC2

De code om de functionaliteiten te testen heet

Test_BlackJack_Test_Spel_Start_only_two_cards.py en bevat scenario's waarin wordt gekeken dat:

- 1) Speler en deler 2 kaarten op de hand hebben
- 2) De tweede kaart onzichtbaar is voor de speler:

```

#####
#Lodewijk Aris version 1, date 1 feb 2025
#This Test code belongs to Test Case Spel Start
#####
import unittest

```

```

from BlackJack_Spel_Start import play_blackjack

class TestBlackJackTC2(unittest.TestCase):

    def test_two_cards_dealt(self):
        # Call the function to test
        player_hand, dealer_hand = play_blackjack()

        # Assert that player and dealer both have 2 cards
        self.assertEqual(len(player_hand), 2)
        self.assertEqual(len(dealer_hand), 2)

        # Print the cards that player and dealer have 2 cards
        print("")
        print(f"Aantal kaarten speler:{len(player_hand)}, Aantal kaarten
deler:{len(dealer_hand)}")
        print(player_hand)
        print(dealer_hand)

if __name__ == '__main__':
    unittest.main()

```

3.2.5 Code onder Test TC3

Het stukje code onder test is BlackJack_hit.py en bevat de functie deal_card() en is voldoende om na te gaan dat er een kaart wordt uitgedeeld:

```

#####
#Lodewijk Aris version 1, date 1 feb 2025                                     #
#This sourcecode is part of the code BlackJack and covers Test Case Hit      #
#####

import random

# The function to be tested
def deal_card(deck, hand):
    card = random.choice(deck)
    deck.remove(card)
    hand.append(card)

```

3.2.6 De Testcode voor TC3

De code om de functionaliteit te testen heet Test_BlackJack_hit.py en bevat scenario's waarin wordt gekeken dat:

- 1) Speler een of meerdere kaarten krijgt
- 2) Het dek kaarten kleiner wordt (met het aantal kaarten verminderd van speler)

```

#####
#Lodewijk Aris version 1, date 1 feb 2025                                     #
#This Test code belongs to Test Case Hit                                      #
#####

import unittest
from BlackJack_hit import deal_card

class TestBlackJackTC3(unittest.TestCase):
    def test_deal_card(self):
        # Zet een initieel dek en een hand op
        deck = [
            ('4', 'Clubs'), ('5', 'Spades'),
            ('6', 'Hearts'), ('7', 'Diamonds'), ('8', 'Clubs'), ('9', 'Spades'),
            ('10', 'Hearts'), ('Jack', 'Diamonds'), ('Queen', 'Clubs'), ('King', 'Spades'),
            ('Ace', 'Hearts'), ('2', 'Diamonds'), ('3', 'Clubs'), ('4', 'Spades'),
            ('5', 'Hearts'), ('6', 'Diamonds'), ('7', 'Clubs'), ('8', 'Spades'),
            ('9', 'Hearts'), ('10', 'Diamonds'), ('Jack', 'Clubs'), ('Queen', 'Spades'),
            ('King', 'Hearts'), ('Ace', 'Diamonds'), ('2', 'Clubs'), ('3', 'Spades'),
            ('4', 'Hearts'), ('5', 'Diamonds'), ('6', 'Clubs'), ('7', 'Spades'),

```



```

('8', 'Hearts'), ('9', 'Diamonds'), ('10', 'Clubs'), ('Jack', 'Spades'),
('Queen', 'Hearts'), ('King', 'Diamonds'), ('Ace', 'Clubs'), ('2', 'Spades'),
('3', 'Hearts'), ('4', 'Diamonds'), ('5', 'Clubs'), ('6', 'Spades'),
('7', 'Hearts'), ('8', 'Diamonds'),
('Jack', 'Hearts'), ('Queen', 'Diamonds'), ('King', 'Clubs'), ('Ace', 'Spades')
]

# Spel start met 2 kaarten voor speler en 2 voor deler, 48 kaarten over in dek
hand = [('2', 'Hearts'), ('3', 'Diamonds')]
self.assertEqual(len(hand), 2)
dealer_hand = [('9', 'Clubs'), ('10', 'Spades')]
self.assertEqual(len(dealer_hand), 2)

# Roep de functie aan om de te testen dat de speler 1 kaart neemt
deal_card(deck, hand)

# Doe een bewering dat de hand nu 3 kaarten heeft (2+1)
self.assertEqual(len(hand), 3)
# Beweer dat het dek 1 kaart minder heeft 48-1
self.assertEqual(len(deck), 47)
# Beweer dat de kaart niet langer in het dek is
self.assertNotIn(hand[0], deck)

# Roep de functie aan om de te testen dat de speler nog 2 kaarten neemt
deal_card(deck, hand)
deal_card(deck, hand)
# Doe een bewering dat de hand nu 2+3 kaarten heeft
self.assertEqual(len(hand), 5)
# Beweer dat het dek 1 kaart minder heeft 47-2
self.assertEqual(len(deck), 45)

# Beweer dat de kaart niet langer in het dek is
self.assertNotIn((hand[0]) and (hand[1]) and (hand[2]), deck)
print(f"")
print(f"Speler heeft nu de volgende vijf kaarten:{hand}")
print(f"Deler heeft nu de volgende twee kaarten:{dealer_hand}")
print(f"Aantal kaarten in dek over:{(len(deck))}")
if __name__ == '__main__':
    unittest.main()

```

3.2.7 Code onder Test TC4

Het stukje code onder test is BlackJack_stand.py en bevat de functies create_deck(), deal_card() en play_blackjack() en is voldoende om het spel te starten en de speler de gelegenheid te geven om specifiek te passen:

```

#####
#Lodewijk Aris version 2, date 14 feb 2025, print verwijderd, verplaatst naar test code #
#                               version 1, date 2 feb 2025                               #
#This sourcecode is part of the code BlackJack and covers Test Case Stand             #
#####

import random
print("Tester geeft meteen s voor stand op (,please insert 1 stand to let the test run smoothly")
# Functie om dek kaarten te creeren
def create_deck():
    deck = []
    suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']
    for suit in suits:
        for rank in ranks:
            deck.append((rank, suit))
    return deck

# Functie om kaart te delen
def deal_card(deck, hand):
    card = random.choice(deck)
    deck.remove(card)
    hand.append(card)

# Hoofdfunctie om blackjack te spelen

```

```
def play_blackjack():
    deck = create_deck()
    random.shuffle(deck)

    player_hand = []
    dealer_hand = []

    for _ in range(2):
        deal_card(deck, player_hand)
        deal_card(deck, dealer_hand)

    # Spelers beurt
    while True:
        choice = input("Do you want to hit or stand? (h/s): ").lower()
        if choice == 'h':
            deal_card(deck, player_hand)
        elif choice == 's':
            break
        else:
            print("Invalid choice. Please enter 'h' to hit or 's' to stand.")
    print("")
    # print(player_hand)
    # print(dealer_hand)

    return player_hand, dealer_hand, deck

# Run the game
play_blackjack()
```

3.2.8 De Testcode voor TC4

Er is voor gekozen om met de patch functie een bepaalt gedrag te simuleren van de speler. De code om de functionaliteiten te testen heet

Test_BlackJack_hit_hit_stand.py en bevat scenario's waarin wordt gekeken dat:

- 1) Speler na 2 hits past
- 2) Speler precies 4 kaarten heeft en deler (nog) 2.

```
#####
#Lodewijk Aris version 2, date 14 feb 2025: print kaarten deler en speler toegevoegd #
# version 1, date 5 feb 2025 #
#This Test code belongs to Test Case Stand #
# #
# belangrijke opmerking: Om de test uit te voeren moet de gebruiker de -s gebruiken #
# in de run configuration #
# In deze test moet de tester 1 keer s (stand) ingeven #
# Door de patch functie gaan er toch 2 hits en 1 stand worden ingevoerd; #
# dat resulteert in 4 kaarten speler en 2 kaarten deler #
#####

import unittest
from unittest.mock import patch
from BlackJack_stand import play_blackjack

class TestBlackJackTC4(unittest.TestCase):
    @patch('builtins.input', side_effect=['h', 'h', 's'])
    def test_hit_hit_stand(self, mock_input):
        # Roep de functie op om te testen
        player_hand, dealer_hand, deck = play_blackjack()

        # Beweer dat de spelers hand precies 4 kaarten is na input
        self.assertEqual(len(player_hand), 4)
        print(f"lengte van spelers hand is: {len(player_hand)}")
        print(player_hand)

        # Beweer dat de deler nog precies 2 kaarten heeft na het spel van de speler
        self.assertEqual(len(dealer_hand), 2)
        print(f"lengte van delers hand is: {len(dealer_hand)}")
        print(dealer_hand)
```

```
if __name__ == '__main__':  
    unittest.main()
```

3.2.9 Code onder Test TC5

Het stukje code onder test is BlackJack_count_hands.py en bevat de functie `calculate_hand_value()` en is voldoende om het aantal punten op de hand uit te rekenen van speler en deler:

```
#####  
#Lodewijk Aris version 1, date 1 feb 2025  
#This sourcecode is part of the code BlackJack and covers Test Case count hands  
#####  
  
import random  
  
# Function to calculate the value of a hand  
def calculate_hand_value(hand):  
    value = 0  
    ace_count = 0  
    for card in hand:  
        rank, suit = card  
        if rank in ['Jack', 'Queen', 'King']:  
            value += 10  
        elif rank == 'Ace':  
            ace_count += 1  
            value += 11  
        else:  
            value += int(rank)  
  
    # Adjust for aces  
    while value > 21 and ace_count:  
        value -= 10  
        ace_count -= 1  
  
    return value
```

3.2.10 De Testcode voor TC5

De code om de functionaliteiten te testen heet `Test_BlackJack_count_hands.py` en bevat scenario's waarin wordt gekeken dat de waarde van spelers hand correct zijn uitgerekend (in verschillende samenstellingen van de hand met azen en plaatjes).

```
#####  
#Lodewijk Aris version 1, date 1 feb 2025  
#This Test code belongs to Test Case count hands  
#####  
  
#Test Case geteld, de opgetelde waarde is correct voor speler en deler  
  
import unittest  
import time  
from BlackJack_count_hands import calculate_hand_value  
class TestBlackJackTC5(unittest.TestCase):  
    def test_hand_value_no_aces_no_tens(self):  
        hand = [('2', 'Hearts'), ('3', 'Diamonds'), ('4', 'Clubs')]  
        self.assertEqual(calculate_hand_value(hand), 9)  
        print(f"Speler heeft nu :{calculate_hand_value(hand)} punten")  
        time.sleep(0.5)  
    def test_hand_value_with_face_cards(self):  
        hand = [('Jack', 'Hearts'), ('3', 'Diamonds'), ('King', 'Clubs')]  
        self.assertEqual(calculate_hand_value(hand), 23)  
        print(f"Speler heeft nu :{calculate_hand_value(hand)} punten")  
        time.sleep(0.5)  
    def test_hand_value_with_aces(self):  
        hand = [('Ace', 'Hearts'), ('3', 'Diamonds'), ('Ace', 'Clubs')]  
        self.assertEqual(calculate_hand_value(hand), 15)
```

```
print(f"Speler heeft nu :{calculate_hand_value(hand)} punten")
time.sleep(0.5)
def test_hand_value_with_ace_adjustment(self):
    hand = [('Ace', 'Hearts'), ('Ace', 'Clubs'), ('10', 'Diamonds')]
    self.assertEqual(calculate_hand_value(hand), 12)
    print(f"Speler heeft nu :{calculate_hand_value(hand)} punten")
    time.sleep(0.5)
def test_hand_value_busting_with_aces(self):
    hand = [('Ace', 'Hearts'), ('Ace', 'Diamonds'), ('9', 'Clubs'), ('Ace', 'Spades'),
('10', 'Spades')]
    self.assertEqual(calculate_hand_value(hand), 22)
    print(f"Speler heeft nu :{calculate_hand_value(hand)} punten")
if __name__ == '__main__':
    unittest.main()
```

3.2.11 Code onder Test TC6

Het stukje code onder test is BlackJack_BJ.py en bevat alle functies, zie Annex A BlackJack Code

- a) create_deck
- b) calculate_hand_value
- c) display_hand
- d) deal_card
- e) play_black_jack

Er is echter geen specifieke code aangetroffen om blackjack te herkennen.

3.2.12 De Testcode voor TC6

Ondanks dat er geen functionaliteit is voor blackjack is er toch al testcode om de functionaliteit blackjack te testen en heet Test_BlackJack_BJ.py en bevat scenario's waarin wordt gekeken dat BlackJack wordt herkend:

```
#####
#Lodewijk Aris version 1, date 3 feb 2025
#This Test code belongs to Test Case BlackJack
#####

import unittest
#from BlackJack_Check_is_BJ import is_blackjack
#from tests.unit.TC6_BlackJack.BlackJack_Check import is_blackjack
from BlackJack_BJ import play_blackjack, is_blackjack

class TestIsBlackjack(unittest.TestCase):
    def test_blackjack(self):
        hand = [('Ace', 'Hearts'), ('10', 'Diamonds')]
        self.assertTrue(is_blackjack(hand))

        hand = [('King', 'Clubs'), ('Ace', 'Spades')]
        self.assertTrue(is_blackjack(hand))

        hand = [('Ace', 'Hearts'), ('Jack', 'Diamonds')]
        self.assertTrue(is_blackjack(hand))

        hand = [('Queen', 'Hearts'), ('Ace', 'Clubs')]
        self.assertTrue(is_blackjack(hand))

    def test_not_blackjack(self):
        hand = [('2', 'Hearts'), ('10', 'Diamonds')]
        self.assertFalse(is_blackjack(hand))

        hand = [('Ace', 'Hearts'), ('9', 'Diamonds')]
        self.assertFalse(is_blackjack(hand))

        hand = [('Jack', 'Hearts'), ('9', 'Diamonds')]
        self.assertFalse(is_blackjack(hand))
```

```

hand = [('2', 'Hearts'), ('3', 'Diamonds')]
self.assertFalse(is_blackjack(hand))

hand = [('2', 'Hearts'), ('10', 'Diamonds'), ('9', 'Diamonds')]
self.assertFalse(is_blackjack(hand))

hand = [('Ace', 'Hearts'), ('9', 'Diamonds'), ('Ace', 'Spades')]
self.assertFalse(is_blackjack(hand))

if __name__ == '__main__':
    unittest.main()

```

De functie “is_blackjack” moet worden toegevoegd aan de code om het te kunnen testen.

3.2.13 Code onder Test TC7

Het stukje code onder test is BlackJack_kapot.py en bevat de functie calculate_hand_value(hand) en is voldoende om het aantal punten op de hand uit te rekenen van speler en deler:

```

#####
#Lodewijk Aris version 1, date 3 feb 2025 #
##This sourcecode is part of the code BlackJack and covers Test Case Kapot #
#####

import random

# Function to calculate the value of a hand
def calculate_hand_value(hand):
    value = 0
    ace_count = 0
    for card in hand:
        rank, suit = card
        if rank in ['Jack', 'Queen', 'King']:
            value += 10
        elif rank == 'Ace':
            ace_count += 1
            value += 11
        else:
            value += int(rank)

    # Adjust for aces
    while value > 21 and ace_count:
        value -= 10
        ace_count -= 1

    return value

```

3.2.14 De Testcode voor TC7

De code om de functionaliteiten te testen heet Test_BlackJack_kapot.py en bevat scenario's waarin wordt gekeken dat het spel herkent dat de waarde van de hand voor deler en speler groter is dan 21 (en dus kapot):

```

#####
#Lodewijk Aris version 1, date 3 feb 2025 #
#This Test code belongs to Test Case Kapot #
#####

#Test Case Kapot; is waar als waarde hand groter is dan 21 voor speler en deler

import unittest

from BlackJack_kapot import calculate_hand_value

```

```

class TestBlackjackBusts(unittest.TestCase):
    def test_player_busts_with_3_cards(self):
        hand = [('10', 'Hearts'), ('10', 'Diamonds'), ('2', 'Clubs')]
        self.assertGreater(calculate_hand_value(hand), 21)
        print(f"Waarde Hand:{calculate_hand_value(hand)} punten")
    def test_player_busts_with_4_cards(self):
        hand = [('5', 'Hearts'), ('6', 'Diamonds'), ('7', 'Clubs'), ('4', 'Spades')]
        self.assertGreater(calculate_hand_value(hand), 21)
        print(f"Waarde Hand:{calculate_hand_value(hand)} punten")
    def test_dealer_busts_with_3_cards(self):
        hand = [('9', 'Hearts'), ('9', 'Diamonds'), ('4', 'Clubs')]
        self.assertGreater(calculate_hand_value(hand), 21)
        print(f"Waarde Hand:{calculate_hand_value(hand)} punten")
    def test_dealer_busts_with_5_cards(self):
        hand = [('3', 'Hearts'), ('4', 'Diamonds'), ('5', 'Clubs'), ('6', 'Spades'), ('5',
'Diamonds')]
        self.assertGreater(calculate_hand_value(hand), 21)
        print(f"Waarde Hand:{calculate_hand_value(hand)} punten")
    def test_dealer_busts_with_4_cards_and_face_card(self):
        hand = [('3', 'Hearts'), ('4', 'Diamonds'), ('5', 'Clubs'), ('6', 'Spades'), ('Queen',
'Diamonds')]
        self.assertGreater(calculate_hand_value(hand), 21)
        print(f"Waarde Hand:{calculate_hand_value(hand)} punten")
if name == ' main ':
    unittest.main()

```

3.2.15 Code onder Test TC8

Het stukje code onder test is BlackJack_dealer_turn.py en bevat de functies create_deck(), calculate_hand_value(hand), deal_card() en is voldoende om het de regels van het spel uit te voeren:

```

#####
#Lodewijk Aris version 2, date 11 feb 2025
# Initial version 1, date 4 feb 2025
#This sourcecode is part of the code BlackJack and covers Test Case Dealer Turn
#####

import random

# Function to create a deck of cards
def create_deck():
    deck = []
    suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']
    for suit in suits:
        for rank in ranks:
            deck.append((rank, suit))
    return deck

# Function to calculate the value of a hand
def calculate_hand_value(hand):
    value = 0
    ace_count = 0
    for card in hand:
        rank, suit = card
        if rank in ['Jack', 'Queen', 'King']:
            value += 10
        elif rank == 'Ace':
            ace_count += 1
            value += 11
        else:
            value += int(rank)

    # Adjust for aces
    while value > 21 and ace_count:
        value -= 10
        ace_count -= 1

    return value

```

```
# Function to deal a card
def deal_card(deck, hand):
    card = random.choice(deck)
    deck.remove(card)
    hand.append(card)
```

3.2.16 De Testcode voor TC8

De code om de functionaliteiten te testen heet Test_BlackJack_dealer_turn.py en bevat scenario's waarin wordt gekeken dat dealer

- een kaart neemt als <17 bij verschillende samenstellingen van de initiële kaartsamenstelling (met azen en plaatjes) en kapot kan gaan
- past tussen 17 en 21

```
#####
#Lodewijk Aris version 2, date 11 feb 2025                                     #
#      Initial version 1, date 4 feb 2025                                     #
#This Test code belongs to Test Case Dealer Turn                             #
#####
import unittest
from BlackJack_dealer_turn import create_deck, calculate_hand_value, deal_card

class TestBlackjack(unittest.TestCase):

    def setUp(self):
        self.deck = create_deck()
        self.dealer_hand = []

    def test_dealer_stands_17(self):
        # Dealer's hand starts with a value of 17 and stops
        self.dealer_hand = [('10', 'Hearts'), ('7', 'Spades')]
        print(f"")
        print(f"start waarde: {calculate_hand_value(self.dealer_hand)}")
        while calculate_hand_value(self.dealer_hand) < 17:
            deal_card(self.deck, self.dealer_hand)
            value = calculate_hand_value(self.dealer_hand)
            self.assertGreaterEqual(value, 17, "Dealer should stand with hand value of at least
17.")
        print(f"eind waarde: {calculate_hand_value(self.dealer_hand)}")

        # reset setup, deck and dealer hand
    def setUp(self):
        self.deck = create_deck()
        self.dealer_hand = []

    def test_dealer_stands_19(self):
        # Dealer's hand starts with a value of 17 and stops
        self.dealer_hand = [('10', 'Hearts'), ('9', 'Clubs')]
        print(f"")
        print(f"start waarde: {calculate_hand_value(self.dealer_hand)}")
        while calculate_hand_value(self.dealer_hand) < 17:
            deal_card(self.deck, self.dealer_hand)
            value = calculate_hand_value(self.dealer_hand)
            self.assertGreaterEqual(value, 17, "Dealer should stand with hand value of at least
17.")
        print(f"eind waarde: {calculate_hand_value(self.dealer_hand)}")

        # reset setup, deck and dealer hand
    def setUp(self):
        self.deck = create_deck()
        self.dealer_hand = []

    def test_dealer_hits(self):
        # Dealer's hand starts with a value less than 17
        self.dealer_hand = [('5', 'Hearts'), ('9', 'Spades')]
        print(f"")
        print(f"start waarde: {calculate_hand_value(self.dealer_hand)}")
        while calculate_hand_value(self.dealer_hand) < 17:
            deal_card(self.deck, self.dealer_hand)
            value = calculate_hand_value(self.dealer_hand)
```

```
17.")
    self.assertGreaterEqual(value, 17, "Dealer should hit until hand value is at least
17.")
    print(f"eind waarde:{calculate_hand_value(self.dealer_hand)}")

    # reset setup, deck and dealer hand
def setUp(self):
    self.deck = create_deck()
    self.dealer_hand = []

def test_dealer_busts_23(self):
    # Dealer's hand will bust after hitting
    self.dealer_hand = [('10', 'Hearts'), ('6', 'Spades')]
    print(f"")
    print(f"start waarde:{calculate_hand_value(self.dealer_hand)}")
    # Deal one more specific card to trigger bust
    while calculate_hand_value(self.dealer_hand) <= 21:
        self.dealer_hand.insert(0, ('7', 'Hearts'))
    # deal_card(self.deck, self.dealer_hand)
    value = calculate_hand_value(self.dealer_hand)
    self.assertGreater(value, 21, "Dealer should bust with hand value greater than 21.")
    print(f"eind waarde:{calculate_hand_value(self.dealer_hand)}")

def setUp(self):
    self.deck = create_deck()
    self.dealer_hand = []

def test_dealer_busts_25(self):
    # Dealer's hand will bust after hitting
    self.dealer_hand = [('10', 'Hearts'), ('5', 'Spades')]
    print(f"")
    print(f"start waarde:{calculate_hand_value(self.dealer_hand)}")
    # Deal one more specific card to trigger bust
    while calculate_hand_value(self.dealer_hand) <= 21:
        self.dealer_hand.insert(0, ('10', 'Diamonds'))
    value = calculate_hand_value(self.dealer_hand)
    self.assertGreater(value, 21, "Dealer should bust with hand value greater than 21.")
    print(f"eind waarde:{calculate_hand_value(self.dealer_hand)}")

    # reset setup, deck and dealer hand
def setUp(self):
    self.deck = create_deck()
    self.dealer_hand = []

def test_dealer_ace_adjustment_1(self):
    # Dealer's hand includes an Ace
    self.dealer_hand = [('Ace', 'Hearts'), ('6', 'Spades')]
    print(f"")
    print(f"start waarde:{calculate_hand_value(self.dealer_hand)}")
    # deal_card(self.deck, self.dealer_hand)
    while calculate_hand_value(self.dealer_hand) < 17:
        # deal_card(self.deck, self.dealer_hand)
        self.dealer_hand.insert(0, ('10', 'Hearts'))
    value = calculate_hand_value(self.dealer_hand)
    self.assertTrue(17 <= value <= 21, "Dealer hand value should be adjusted correctly
with an Ace.")
    print(f"eind waarde:{calculate_hand_value(self.dealer_hand)}")

    # reset setup, deck and dealer hand
def setUp(self):
    self.deck = create_deck()
    self.dealer_hand = []

def test_dealer_ace_adjustment_2(self):
    # Dealer's hand includes an Ace
    self.dealer_hand = [('Ace', 'Hearts'), ('5', 'Spades')]
    print(f"")
    print(f"start waarde:{calculate_hand_value(self.dealer_hand)}")
    # deal_card(self.deck, self.dealer_hand)
    while calculate_hand_value(self.dealer_hand) < 17:
        # deal_card(self.deck, self.dealer_hand)
        self.dealer_hand.insert(0, ('Ace', 'Clubs'))
    value = calculate_hand_value(self.dealer_hand)
    self.assertTrue(17 <= value <= 21, "Dealer hand value should be adjusted correctly
with two Aces.")
    print(f"eind waarde:{calculate_hand_value(self.dealer_hand)}")
```



```
if __name__ == '__main__':  
    unittest.main()
```

3.2.17 Code onder Test TC9

Er bestaat geen functionaliteit in de code om kaarten te splitsen zie Annex A BlackJack Code. Er is dus geen code onder test.

3.2.18 De Testcode voor TC9

De code om de functionaliteiten te testen om te splitsen is niet geschreven want de functionaliteit splitsen ontbreekt.

3.2.19 Code onder Test TC10

Dit is een systeem test en de code onder test is de hele file BlackJack.py, zie Annex A BlackJack Code. In de test zal de gebruiker door het gehele spel te spelen vanzelf scenario's creëren, waarin de speler ten minste 1 keer wint, 1 keer verliest en 1 keer gelijk speelt.

3.2.20 De Testcode voor TC10

Er is geen specifiek testcode, behalve de BlackJack.py file zelf, die moet gerund worden en de uitkomsten moeten worden vergeleken met de verwachtingen (door te inspecteren).

3.2.21 Code onder Test TC11

Dit is een systeem test en de code onder test is de hele file BlackJack.py, zie Annex A BlackJack Code. In de test zal de gebruiker het hele spel spelen en verkeerde inputs geven als speler.

3.2.22 De Testcode voor TC11

Er is geen specifiek testcode, behalve de BlackJack.py file zelf, die moet gerund worden en de uitkomsten moeten worden gelogd (dus door inspectie).

4 TESTRAPPORT

Project: BlackJack

Versie: v1.0

Tester: Lodewijk Aris

Voor test omgeving, zie Testomgeving.

Doel van de test:

Het valideren van de basisfunctionaliteiten van het blackjack spel met de focus dat de spelfunctionaliteiten naar behoren werken, op zowel unit als systeem niveau. De “pass/fail” criteria (verwachte resultaten) staan in Tabel 1.

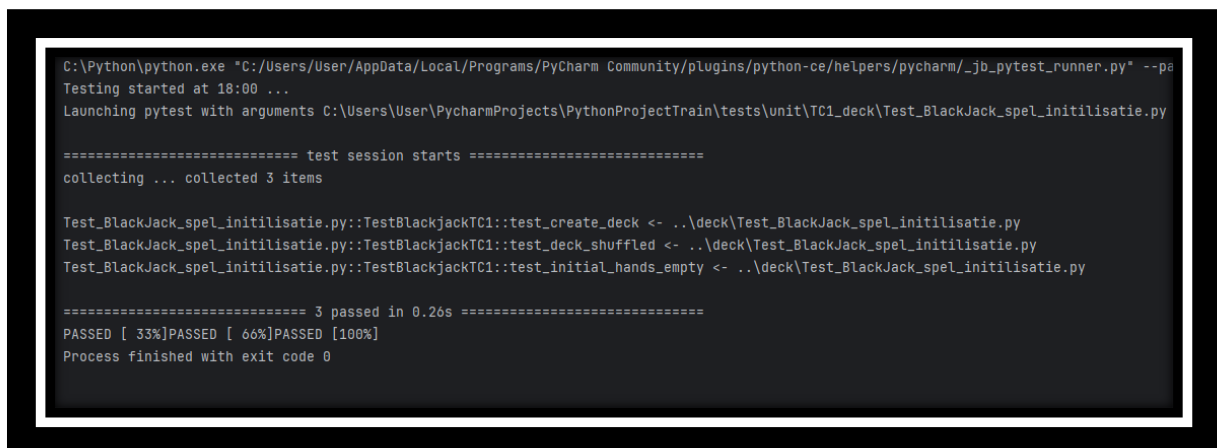
4.1 TC1 Spel initialisatie Test resultaten

4.1.1 Testprocedure TC1 Spel initialisatie:

Om tot een test resultaat te komen voor TC1: Run in PyCharm of Wing de Test_BlackJack_spel_initilisatie.py in de directory \tests\unit\TC1_deck\ omgeving. Hierin staat de functiecode create_deck van het BlackJack.py spel (genaamd BlackJack_spel_initilisatie.py).

Test datum: 9 februari 2025 en 13 februari.

Functionaliteit	Resultaat	Opmerkingen
create_deck	GESLAAGD:	het dek heeft 52 kaarten. Het dek is geschud. Spelers en Delers hand is leeg bij aanvang. De drie sub functies zijn succesvol getest, zie Figuur 6 en Figuur 7, [100%] en OK.



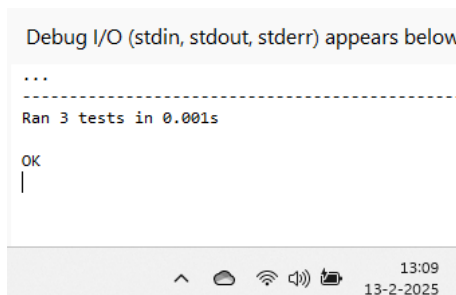
```
C:\Python\python.exe "C:/Users/User/AppData/Local/Programs/PyCharm Community/plugins/python-ce/helpers/pycharm/_jb_pytest_runner.py" --pa
Testing started at 18:00 ...
Launching pytest with arguments C:\Users\User\PycharmProjects\PythonProjectTrain\tests\unit\TC1_deck\Test_BlackJack_spel_initilisatie.py

===== test session starts =====
collecting ... collected 3 items

Test_BlackJack_spel_initilisatie.py::TestBlackjackTC1::test_create_deck <- ..\deck\Test_BlackJack_spel_initilisatie.py
Test_BlackJack_spel_initilisatie.py::TestBlackjackTC1::test_deck_shuffled <- ..\deck\Test_BlackJack_spel_initilisatie.py
Test_BlackJack_spel_initilisatie.py::TestBlackjackTC1::test_initial_hands_empty <- ..\deck\Test_BlackJack_spel_initilisatie.py

===== 3 passed in 0.26s =====
PASSED [ 33%]PASSED [ 66%]PASSED [100%]
Process finished with exit code 0
```

Figuur 6 TC1 test resultaat in PyCharm



```
Debug I/O (stdin, stdout, stderr) appears below

...
-----
Ran 3 tests in 0.001s

OK
|
```

Figuur 7 TC1 test resultaat in Wing

4.2 TC2 Spel Start Test resultaten

4.2.1 Testprocedure TC2 Spel Start:

Om tot een test resultaat te komen voor TC2: Run in PyCharm of Wing de Test_BlackJack_Test_Spel_Start_only_two_cards.py in de directory \tests\unit\TC2_start_two_cards\. Hierin staat de functiecode create_deck van het BlackJack.py spel (genaamd BlackJack_Spel_Start.py)

Test datum: 9 februari 2025 en 13 februari:

Functionaliteiten	Resultaat	Opmerkingen
create_deck deal_card play_blackjack	GESLAAGD	Speler en deler hebben 2 kaarten: De functie om te checken dat de speler en deler 2 kaarten hebben, is succesvol getest, zie Figuur 8 en Figuur 9 (passed en OK), maar de tweede kaart van de deler is zichtbaar dus daarom gefaald (Tweede kaart van de deler is niet verborgen).
	GEFAALD	



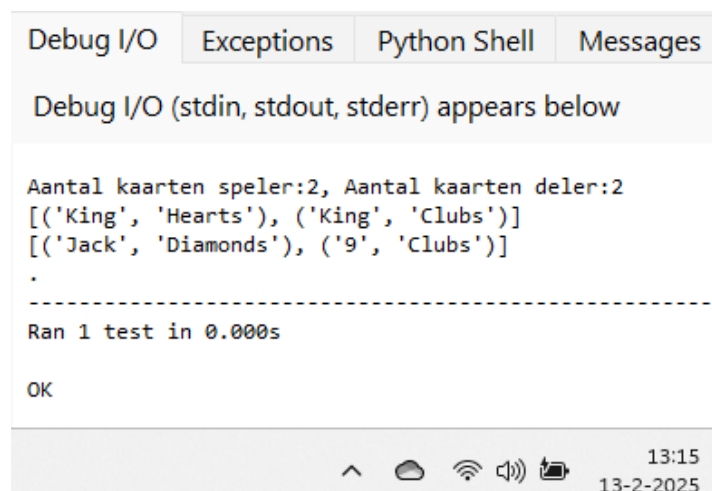
```
✓ Tests passed: 1 of 1 test - 0 ms
C:\Python\python.exe "C:/Users/User/AppData/Local/Programs/PyCharm Community/plugins/python-ce/helpers/pycharm/_jb_pytest_runner.py" --pa
Testing started at 18:52 ...
Launching pytest with arguments C:\Users\User\PycharmProjects\PythonProjectTrain\tests\unit\TC2_start_two_cards\Test_BlackJack_Test_Spel_

===== test session starts =====
collecting ... collected 1 item

Test_BlackJack_Test_Spel_Start_only_two_cards.py::TestBlackJackTC2::test_two_cards_dealt <- ..\start_two_cards\Test_BlackJack_Test_Spel_s
Aantal kaarten speler:2, Aantal kaarten deler:2
[('2', 'Diamonds'), ('3', 'Spades')]
[('5', 'Hearts'), ('8', 'Hearts')]

===== 1 passed in 0.02s =====
```

Figuur 8 TC2 test resultaat PyCharm



```
Debug I/O Exceptions Python Shell Messages

Debug I/O (stdin, stdout, stderr) appears below

Aantal kaarten speler:2, Aantal kaarten deler:2
[('King', 'Hearts'), ('King', 'Clubs')]
[('Jack', 'Diamonds'), ('9', 'Clubs')]
.
-----
Ran 1 test in 0.000s

OK

13:15
13-2-2025
```

Figuur 9 TC2 test resultaat

4.3 TC3 Hit Test resultaten

4.3.1 Testprocedure TC3 Hit:

Om tot een test resultaat te komen voor TC3: Run in PyCharm of Wing de Test_BlackJack_hit.py in de directory \tests\unit\TC3_hit\. Hierin staat de functiecode deal_card van het BlackJack.py spel (genaamd BlackJack_hit.py)

Test datum: 9 februari 2025 en 13 februari:

Functionaliteit	Resultaat	Opmerkingen
deal_card	GESLAAGD	De functie deel kaart is succesvol uitgevoerd en gecheckt voor de drie gedefinieerde scenario's, zie Figuur 10 en Figuur 11.

Note: de drie scenario's zijn:

1. Speler en deler hebben 2 kaarten, dek 48
2. Speler heeft 3 kaarten, dek 47.
3. Speler heeft 5, deler heeft 2 en dek heeft 45 kaarten



```

✓ Tests passed: 1 of 1 test - 1 ms

C:\Python\python.exe "C:\Users\User\AppData\Local\Programs\PyCharm Community\plugins\python-ce\helpers\pycharm\_jb_pytest_runner.py" --
Testing started at 21:33 ...
Launching pytest with arguments C:\Users\User\PycharmProjects\PythonProjectTrain\tests\unit\TC3_hit\Test_BlackJack_hit.py --no-header -

===== test session starts =====
collecting ... collected 1 item

Test_BlackJack_hit.py::TestBlackJackTC3::test_deal_card

===== 1 passed in 0.03s =====
PASSED [100%]
Speler heeft nu de volgende vijf kaarten:[('2', 'Hearts'), ('3', 'Diamonds'), ('8', 'Spades'), ('4', 'Spades'), ('King', 'Spades')]
Deler heeft nu de volgende twee kaarten:[('9', 'Clubs'), ('10', 'Spades')]
aantal kaarten in dek over:45

Process finished with exit code 0
  
```

Figuur 10 TC3 test resultaat PyCharm



```

Debug I/O | Exceptions | Python Shell | Messages | OS Commands |
Debug I/O (stdin, stdout, stderr) appears below

Speler heeft nu de volgende vijf kaarten:[('2', 'Hearts'), ('3', 'Diamonds'), ('Queen', 'Diamonds'), ('King', 'Clubs'), ('9', 'Hearts')]
Deler heeft nu de volgende twee kaarten:[('9', 'Clubs'), ('10', 'Spades')]
Aantal kaarten in dek over:45
.
-----
Ran 1 test in 0.000s

OK
  
```

Figuur 11 TC3 test resultaat Wing

4.4 TC4 Stand Test resultaten

4.4.1 Testprocedure TC4 Stand:

Om tot een test resultaat te komen voor TC4: Run in PyCharm of Wing de Test_BlackJack_hit_hit_stand.py in de directory \tests\unit\TC4_stand\ omgeving. Hierin staat de functiecode play_blackjack van het BlackJack.py spel (genaamd BlackJack_stay.py). Het is belangrijk dat de test wordt uitgevoerd met de -s optie in de “run configuration” van PyCharm (zie Testopzet), omdat de tester input moet geven: [s]. Doormiddel van de patch functie, wordt dan in de testrun [h,h,s] gesimuleerd en ingevoerd, zie code in sectie 3.2.8 De Testcode voor TC4.

Test datum: 14 februari 2025:

Functionaliteit	Resultaat	Opmerkingen
play_blackjack	GESLAAGD	Het programma stopt na 2 keer hit en 1 keer stand en er zijn 4 kaarten zichtbaar, zie Figuur 12 en Figuur 13.

Note: Het spel zou in werkelijkheid al na 1 hit zijn gestopt, maar is voor de unittest niet van belang. Het gaat erom dat de “stand” functie werkt (op het gegeven moment).



```

✓ Tests passed: 1 of 1 test - 1ms
C:\Python\python.exe "C:/Users/User/AppData/Local/Programs/PyCharm Community/plugins/python-ce/helpers/py
Testing started at 10:49 ...
Launching pytest with arguments -s C:\Users\User\PycharmProjects\PythonProjectTrain\tests\unit\TC4_stand\

===== test session starts =====
collecting ... Tester geeft meteen s voor stand op (,please insert 1 stand to let the test run smoothly
Do you want to hit or stand? (h/s):
collected 1 item

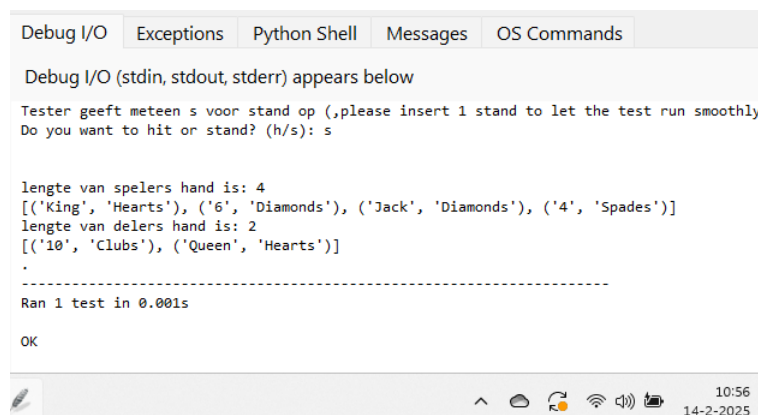
Test_BlackJack_hit_hit_stand.py::TestBlackJackTC4::test_hit_hit_stand

===== 1 passed in 3.18s =====

lengte van spelers hand is: 4
[('4', 'Clubs'), ('Ace', 'Diamonds'), ('7', 'Clubs'), ('4', 'Hearts')]
lengte van delers hand is: 2
[('King', 'Hearts'), ('2', 'Diamonds')]
PASSED
Process finished with exit code 0

```

Figuur 12 TC4 test resultaat PyCharm



```

Debug I/O Exceptions Python Shell Messages OS Commands
Debug I/O (stdin, stdout, stderr) appears below
Tester geeft meteen s voor stand op (,please insert 1 stand to let the test run smoothly
Do you want to hit or stand? (h/s): s

lengte van spelers hand is: 4
[('King', 'Hearts'), ('6', 'Diamonds'), ('Jack', 'Diamonds'), ('4', 'Spades')]
lengte van delers hand is: 2
[('10', 'Clubs'), ('Queen', 'Hearts')]
.
-----
Ran 1 test in 0.001s

OK

```

Figuur 13 TC4 test resultaat Wing

4.5 TC5 count hands Test resultaten

4.5.1 Testprocedure TC5 count hands:

Om tot een test resultaat te komen voor TC5: Run in PyCharm of Wing de Test_BlackJack_count_hands.py in de directory \tests\unit\TC5_count_hands\ omgeving. Hierin staat de functiecode calculate_hand_value van het BlackJack.py spel (hier genoemd BlackJack_count_hands.py).

Test datum: 10 februari 2025 en 13 februari:

Functionaliteit	Resultaat	Opmerkingen
calculate_hand_value	GESLAAGD	De 5 testcases werken zoals verwacht, zie Figuur 14 en Figuur 15.

```

✓ Tests passed: 5 of 5 tests – 2 sec 12 ms
C:\Python\python.exe "C:/Users/User/AppData/Local/Programs/PyCharm Community/plugins/python-ce/helpers/pycharm/_jb_pytest_runner.py" --pat
Testing started at 12:57 ...
Launching pytest with arguments C:\Users\User\PycharmProjects\PythonProjectTrain\tests\unit\TC5_count_hands\Test_BlackJack_count_hands.py

===== test session starts =====
collecting ... collected 5 items

Test_BlackJack_count_hands.py::TestBlackJackTC5::test_hand_value_busting_with_aces
Test_BlackJack_count_hands.py::TestBlackJackTC5::test_hand_value_no_aces_no_tens
Test_BlackJack_count_hands.py::TestBlackJackTC5::test_hand_value_with_ace_adjustment
Test_BlackJack_count_hands.py::TestBlackJackTC5::test_hand_value_with_aces
Test_BlackJack_count_hands.py::TestBlackJackTC5::test_hand_value_with_face_cards

===== 5 passed in 2.10s =====
PASSED [ 20%]Speler heeft nu :22 punten
PASSED [ 40%]Speler heeft nu :9 punten
PASSED [ 60%]Speler heeft nu :12 punten
PASSED [ 80%]Speler heeft nu :15 punten
PASSED [100%]Speler heeft nu :23 punten

Process finished with exit code 0
  
```

Figuur 14 TC5 test resultaat PyCharm

```

Debug I/O  Exceptions  Python Shell  Me

Debug I/O (stdin, stdout, stderr) appears below

Speler heeft nu :22 punten
.Speler heeft nu :9 punten
.Speler heeft nu :12 punten
.Speler heeft nu :15 punten
.Speler heeft nu :23 punten
.
-----
Ran 5 tests in 2.021s

OK
  
```

Figuur 15 TC5 test resultaat Wing

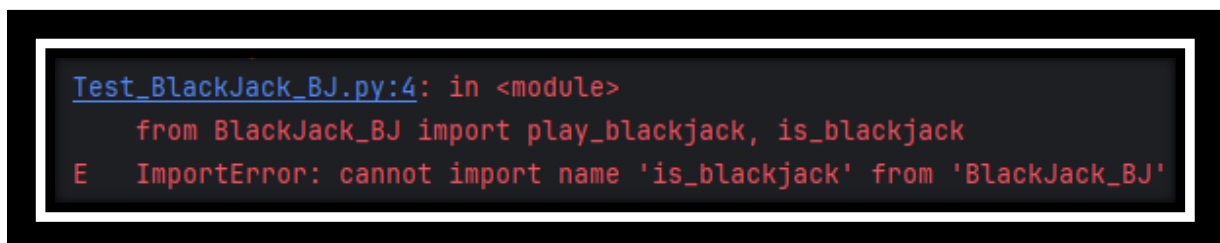
4.6 TC6 BlackJack Test resultaten

4.6.1 Testprocedure TC6 BlackJack:

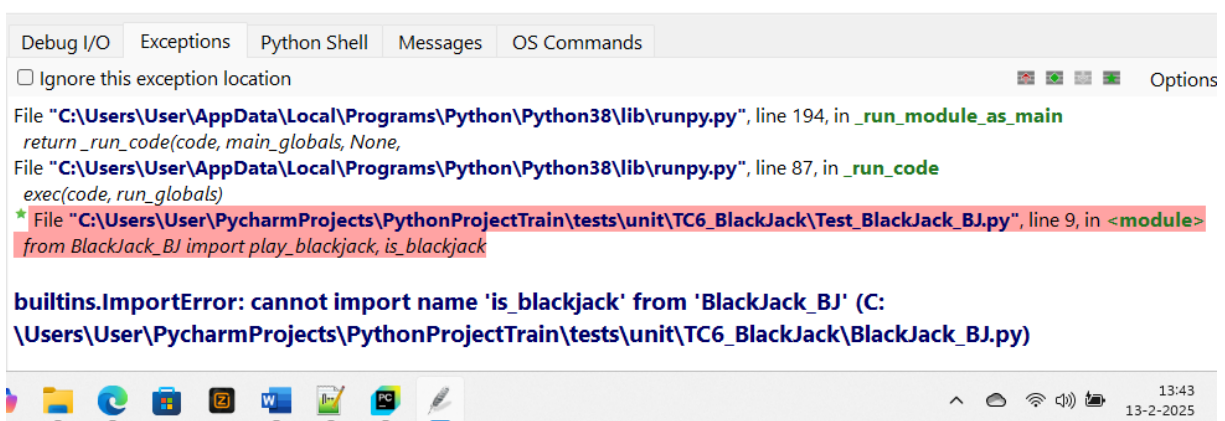
Om tot een test resultaat te komen voor TC6: Run in PyCharm of Wing de Test_BlackJack_BJ.py in de directory \tests\unit\TC6_count_hands\ omgeving. Hierin zou de functiecode is_blackjack van het BlackJack.py spel (hier genoemd BlackJack_BJ.py) moeten staan (maar ontbreekt dus). De test kan ondanks het ontbreken van de code wel worden uitgevoerd. Het resultaat staat hieronder.

Test datum: 10 februari 2025:

Functionaliteit	Resultaat	Opmerkingen
is_blackjack	GEFAALD	De functie om blackjack te detecteren ontbreekt in de code en in de test, zie Figuur 16 en Figuur 17.



Figuur 16 TC6 test resultaat PyCharm



Figuur 17 TC6 test resultaat Wing

4.7 TC7 kapot Test resultaten

4.7.1 Testprocedure TC7 kapot:

Om tot een test resultaat te komen voor TC7: Run in PyCharm of Wing de Test_BlackJack_kapot.py in de directory \tests\unit\TC5_count_hands\ omgeving. Hierin staat de functiecode calculate_hand_value van het BlackJack.py spel (hier genoemd BlackJack_kapot.py).

Test datum: 10 februari 2025:

Functionaliteit	Resultaat	Opmerkingen
calculate_hand_value	GESLAAGD	De 5 testcases werken zoals verwacht, zie Figuur 18 en Figuur 19.

```

✓ Tests passed: 5 of 5 tests – 2 ms
C:\Python\python.exe "C:/Users/User/AppData/Local/Programs/PyCharm Community/plugins/python-ce/helpers/pycharm/_jb_pytest_runner.py"
Testing started at 14:56 ...
Launching pytest with arguments C:\Users\User\PycharmProjects\PythonProjectTrain\tests\unit\TC7_kapot\Test_BlackJack_kapot.py -
===== test session starts =====
collecting ... collected 5 items

Test_BlackJack_kapot.py::TestBlackjackBusts::test_dealer_busts_with_3_cards
Test_BlackJack_kapot.py::TestBlackjackBusts::test_dealer_busts_with_4_cards_and_face_card
Test_BlackJack_kapot.py::TestBlackjackBusts::test_dealer_busts_with_5_cards
Test_BlackJack_kapot.py::TestBlackjackBusts::test_player_busts_with_3_cards
Test_BlackJack_kapot.py::TestBlackjackBusts::test_player_busts_with_4_cards

===== 5 passed in 0.04s =====
PASSED [ 20%]Waarde Hand:22 punten
PASSED [ 40%]Waarde Hand:28 punten
PASSED [ 60%]Waarde Hand:23 punten
PASSED [ 80%]Waarde Hand:22 punten
PASSED [100%]Waarde Hand:22 punten

Process finished with exit code 0

```

Figuur 18 TC7 test resultaat PyCharm

```

Debug I/O   Exceptions   Python Shell   Messages

Debug I/O (stdin, stdout, stderr) appears below

Waarde Hand:22 punten
.Waarde Hand:28 punten
.Waarde Hand:23 punten
.Waarde Hand:22 punten
.Waarde Hand:22 punten
.
-----
Ran 5 tests in 0.001s

OK

```

Figuur 19 TC7 test resultaat Wing

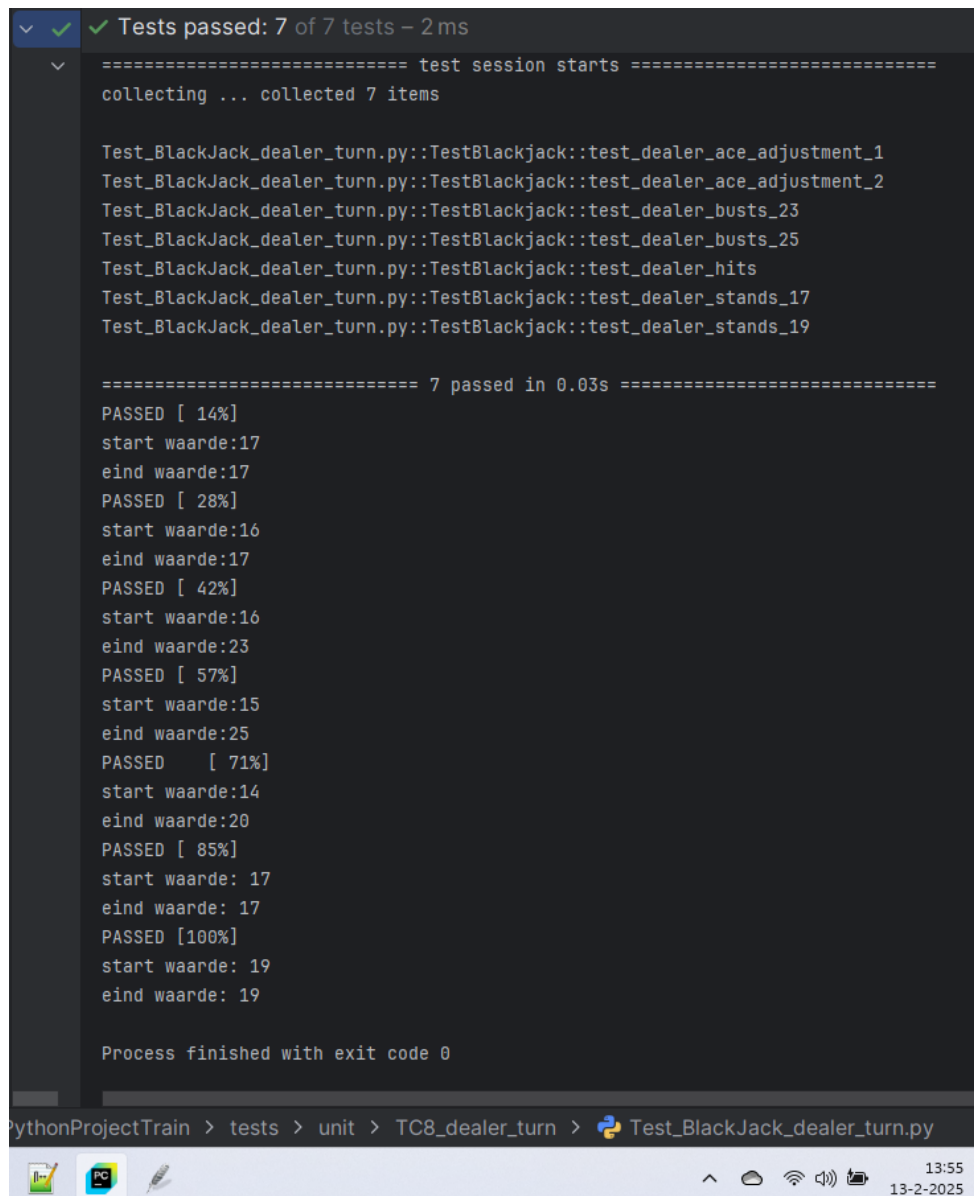
4.8 TC8 Dealer Turn Test resultaten

4.8.1 Testprocedure TC8 Dealer Turn:

Om tot een test resultaat te komen voor TC8: Run in PyCharm of Wing de Test_BlackJack_dealer_turn.py in de directory \tests\unit\TC8_dealer_turn\ omgeving. Hierin staat de functiecodes create_deck, calculate_hand_value, deal_card van het BlackJack.py spel (hier genoemd BlackJack_dealer_turn.py).

Test datum: 13 februari 2025:

Functionaliteit	Resultaat	Opmerkingen
create_deck, calculate_hand_value, deal_card	GESLAAGD	De 7 testcases werken zoals verwacht, zie Figuur 18 en Figuur 19.



```
✓ Tests passed: 7 of 7 tests - 2 ms
===== test session starts =====
collecting ... collected 7 items

Test_BlackJack_dealer_turn.py::TestBlackjack::test_dealer_ace_adjustment_1
Test_BlackJack_dealer_turn.py::TestBlackjack::test_dealer_ace_adjustment_2
Test_BlackJack_dealer_turn.py::TestBlackjack::test_dealer_busts_23
Test_BlackJack_dealer_turn.py::TestBlackjack::test_dealer_busts_25
Test_BlackJack_dealer_turn.py::TestBlackjack::test_dealer_hits
Test_BlackJack_dealer_turn.py::TestBlackjack::test_dealer_stands_17
Test_BlackJack_dealer_turn.py::TestBlackjack::test_dealer_stands_19

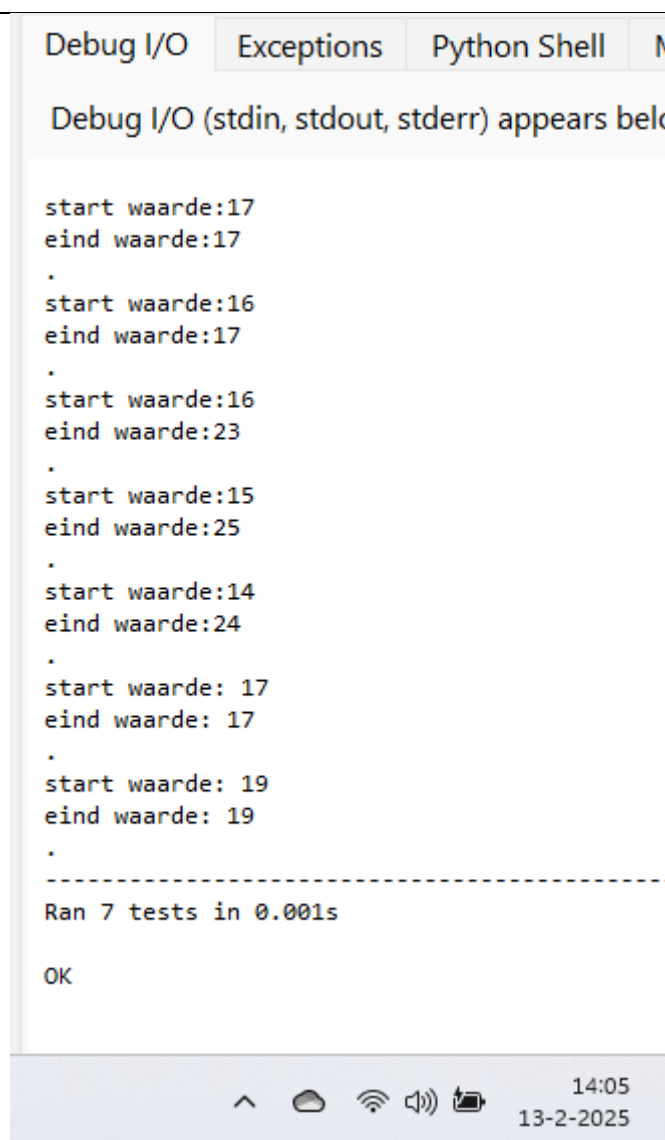
===== 7 passed in 0.03s =====

PASSED [ 14%]
start waarde:17
eind waarde:17
PASSED [ 28%]
start waarde:16
eind waarde:17
PASSED [ 42%]
start waarde:16
eind waarde:23
PASSED [ 57%]
start waarde:15
eind waarde:25
PASSED [ 71%]
start waarde:14
eind waarde:20
PASSED [ 85%]
start waarde: 17
eind waarde: 17
PASSED [100%]
start waarde: 19
eind waarde: 19

Process finished with exit code 0

PythonProjectTrain > tests > unit > TC8_dealer_turn > Test_BlackJack_dealer_turn.py
```

Figuur 20 TC8 test resultaat PyCharm



```
Debug I/O (stdin, stdout, stderr) appears below

start waarde:17
eind waarde:17
.
start waarde:16
eind waarde:17
.
start waarde:16
eind waarde:23
.
start waarde:15
eind waarde:25
.
start waarde:14
eind waarde:24
.
start waarde: 17
eind waarde: 17
.
start waarde: 19
eind waarde: 19
.
-----
Ran 7 tests in 0.001s

OK
```

Figuur 21 TC8 test resultaat Wing

4.9 TC9 Split Test resultaten

Er zijn geen testresultaten voor TC9. De test is daarom gefaald.

Functionaliteit	Resultaat	Opmerkingen
Split	GEFAALD	De functie split ontbreekt in het spel (en dus ook in de code)

4.9.1 Testprocedure TC9 Split:

Er zijn geen testprocedures voor TC9.

4.10 TC10 Game Outcome Test resultaten

4.10.1 Testprocedure TC10 Game Outcome:

Om tot een test resultaat te komen voor TC10: Run in PyCharm of Wing de BlackJack.py in de directory \tests\unit\TC10_game_outcome\ omgeving. Herhaal dit totdat alle drie verschillende uitkomsten zijn bereikt (en maak per uitkomst een snapshot).

Test datum: 11 februari 2025 en 13 februari:

Functionaliteit	Resultaat	Opmerkingen
Alle functionaliteiten	GESLAAGD	Speluitkomsten zoals verwacht, zie snapshots.

Speler wint in PyCharm:

```
C:\Python\python.exe C:\Users\User\PycharmProjects\PythonProjectTrain\tests\system\TC10_game_outcome\BlackJack.py
Player's hand:
7 of Diamonds
3 of Diamonds
Value: 10

Do you want to hit or stand? (h/s): h
Player's hand:
7 of Diamonds
3 of Diamonds
Ace of Hearts
Value: 21

Do you want to hit or stand? (h/s): s
Dealer's hand:
Ace of Spades
9 of Diamonds
Value: 20

Player wins.

Process finished with exit code 0
```

Deler wint in PyCharm:

```
C:\Python\python.exe C:\Users\User\PycharmProjects\PythonProjectTrain\tests\system\TC10_game_outcome\BlackJack.py
Player's hand:
3 of Hearts
Ace of Diamonds
Value: 14

Do you want to hit or stand? (h/s): h
Player's hand:
3 of Hearts
Ace of Diamonds
Ace of Spades
Value: 15

Do you want to hit or stand? (h/s): s
Dealer's hand:
7 of Spades
Jack of Spades
Value: 17

Dealer wins.

Process finished with exit code 0
```

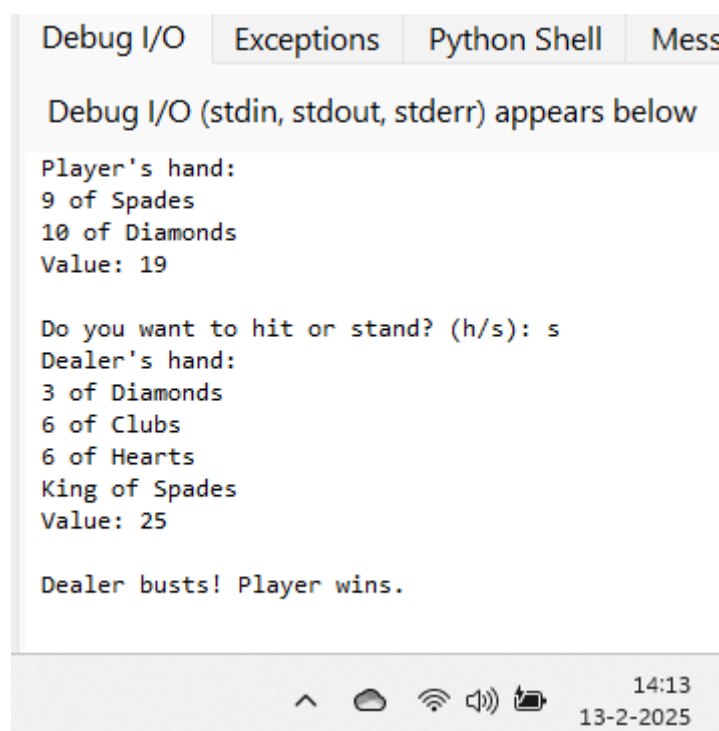
Gelijk spel in PyCharm:

```
C:\Python\python.exe C:\Users\User\PycharmProjects\PythonProjectTrain\tests\system\TC10_game_outcome\BlackJack.py
Player's hand:
10 of Hearts
Queen of Hearts
Value: 20

Do you want to hit or stand? (h/s): s
Dealer's hand:
King of Clubs
Jack of Clubs
Value: 20

It's a tie!

Process finished with exit code 0
```

Speler wint in Wing:

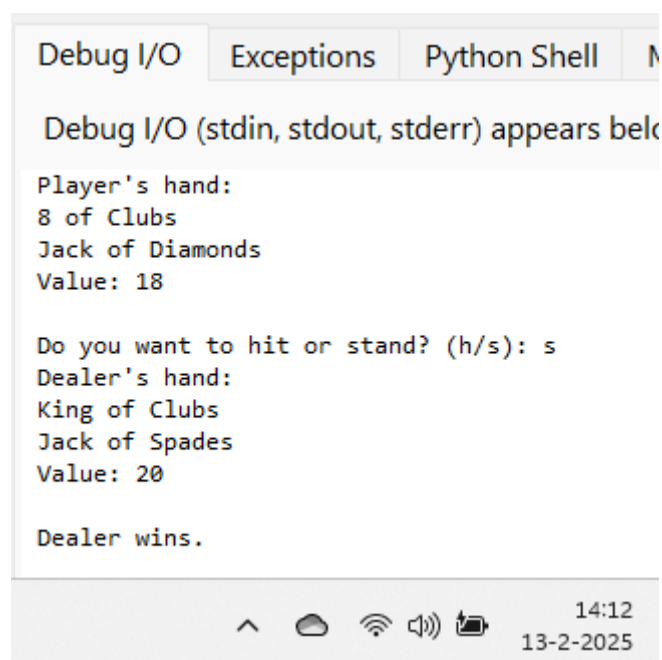
Debug I/O Exceptions Python Shell Mess

Debug I/O (stdin, stdout, stderr) appears below

```
Player's hand:  
9 of Spades  
10 of Diamonds  
Value: 19  
  
Do you want to hit or stand? (h/s): s  
Dealer's hand:  
3 of Diamonds  
6 of Clubs  
6 of Hearts  
King of Spades  
Value: 25  
  
Dealer busts! Player wins.
```

14:13
13-2-2025

Deler wint in Wing:

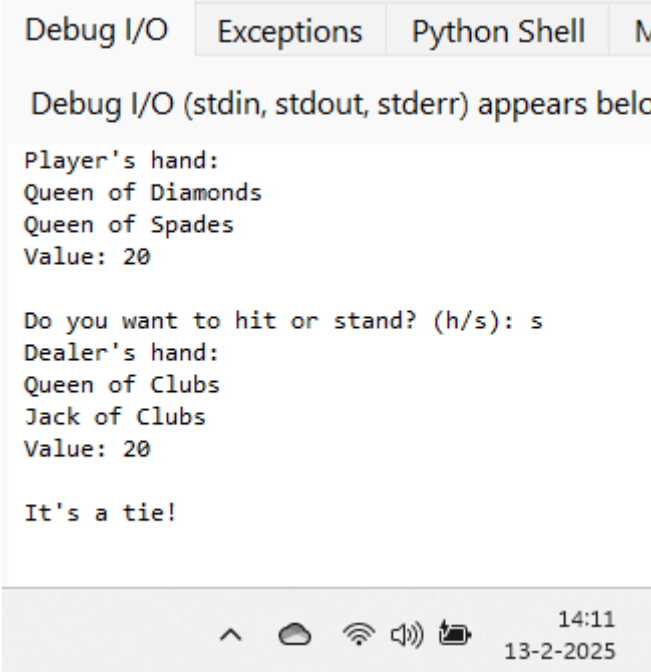


Debug I/O Exceptions Python Shell M

Debug I/O (stdin, stdout, stderr) appears below

```
Player's hand:  
8 of Clubs  
Jack of Diamonds  
Value: 18  
  
Do you want to hit or stand? (h/s): s  
Dealer's hand:  
King of Clubs  
Jack of Spades  
Value: 20  
  
Dealer wins.
```

14:12
13-2-2025

Gelijkspel in Wing:

The screenshot shows the 'Debug I/O' tab in Wing IDE. The text in the window is as follows:

```
Debug I/O (stdin, stdout, stderr) appears below
```

```
Player's hand:  
Queen of Diamonds  
Queen of Spades  
Value: 20  
  
Do you want to hit or stand? (h/s): s  
Dealer's hand:  
Queen of Clubs  
Jack of Clubs  
Value: 20  
  
It's a tie!
```

At the bottom of the window, there is a system tray area showing icons for network, volume, and battery, along with the time 14:11 and date 13-2-2025.

4.11 TC11 Edge Case Test resultaten

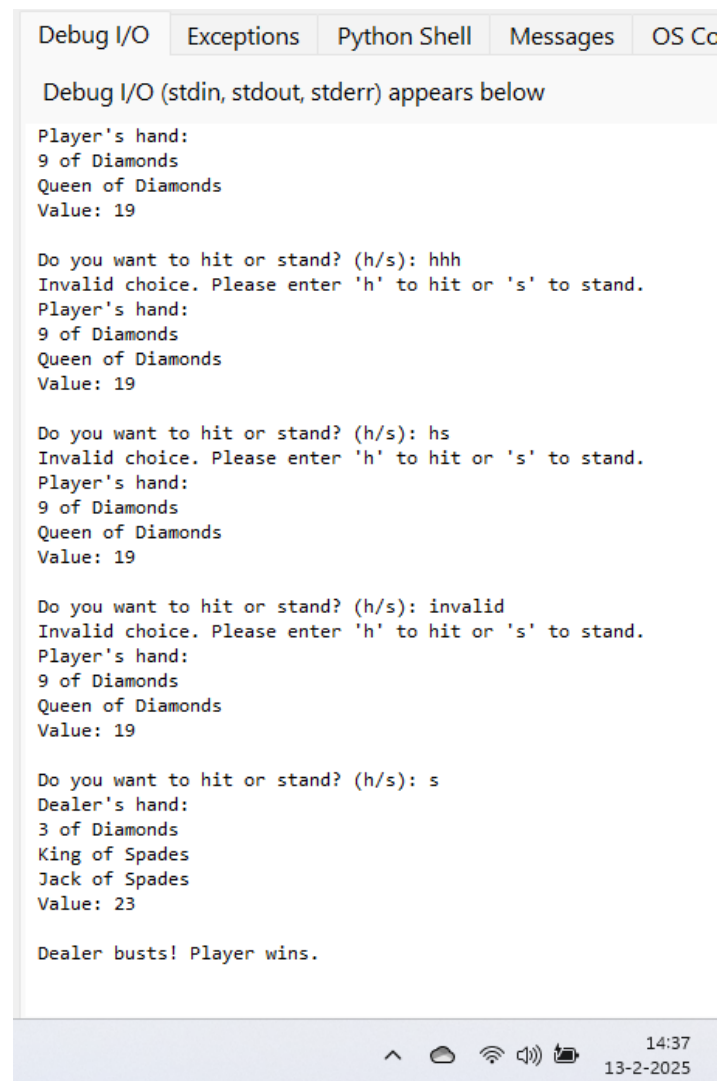
4.11.1 Testprocedure TC11 Edge Case:

Om tot een test resultaat te komen voor TC11: Run in PyCharm of Wing de BlackJack.py in de directory \tests\unit\TC11_edge_case\ omgeving. De speler voert een aantal invalide inputs in en check dat het programma hiermee goed omgaat (en maak snapshots).

Test datum: 13 februari 2025:

Functionaliteit	Resultaat	Opmerkingen
choice_input	GESLAAGD	Het programma handelt verkeerde inputs goed af, zie Figuur 22 en Figuur 23.

Wing:



```
Debug I/O (stdin, stdout, stderr) appears below

Player's hand:
9 of Diamonds
Queen of Diamonds
Value: 19

Do you want to hit or stand? (h/s): hhh
Invalid choice. Please enter 'h' to hit or 's' to stand.
Player's hand:
9 of Diamonds
Queen of Diamonds
Value: 19

Do you want to hit or stand? (h/s): hs
Invalid choice. Please enter 'h' to hit or 's' to stand.
Player's hand:
9 of Diamonds
Queen of Diamonds
Value: 19

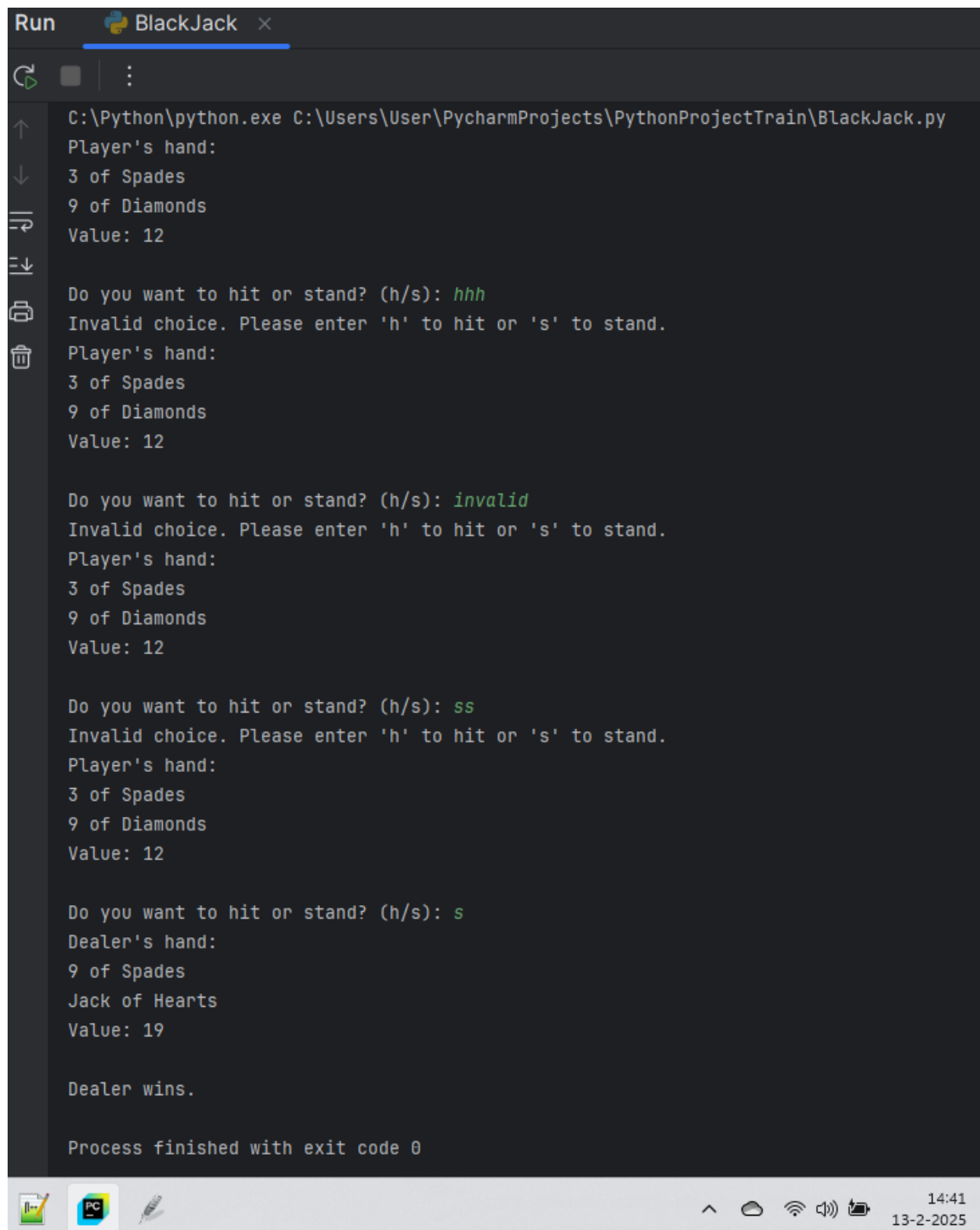
Do you want to hit or stand? (h/s): invalid
Invalid choice. Please enter 'h' to hit or 's' to stand.
Player's hand:
9 of Diamonds
Queen of Diamonds
Value: 19

Do you want to hit or stand? (h/s): s
Dealer's hand:
3 of Diamonds
King of Spades
Jack of Spades
Value: 23

Dealer busts! Player wins.
```

Figuur 22 TC11 test resultaat Wing

PyCharm:



```
Run BlackJack x
C:\Python\python.exe C:\Users\User\PycharmProjects\PythonProjectTrain\BlackJack.py
Player's hand:
3 of Spades
9 of Diamonds
Value: 12

Do you want to hit or stand? (h/s): hhh
Invalid choice. Please enter 'h' to hit or 's' to stand.
Player's hand:
3 of Spades
9 of Diamonds
Value: 12

Do you want to hit or stand? (h/s): invalid
Invalid choice. Please enter 'h' to hit or 's' to stand.
Player's hand:
3 of Spades
9 of Diamonds
Value: 12

Do you want to hit or stand? (h/s): ss
Invalid choice. Please enter 'h' to hit or 's' to stand.
Player's hand:
3 of Spades
9 of Diamonds
Value: 12

Do you want to hit or stand? (h/s): s
Dealer's hand:
9 of Spades
Jack of Hearts
Value: 19

Dealer wins.

Process finished with exit code 0
```

Figuur 23 TC11 test resultaat PyCharm

4.12 Overzicht testresultaten en conclusies van de uitgevoerde tests

Tabel 2 geeft een kort overzicht van de testresultaten.

Tabel 2 Overzicht van de testresultaten

Test Case	Resultaat	Opmerking
1 Spel initialisatie	GESLAAGD	
2 Spel Start	GEFAALD	Het zichtbaar maken van kaarten van de deler faalt.
3 Hit	GESLAAGD	
4 Stand	GESLAAGD	
5 Count Hand(s)	GESLAAGD	
6 BlackJack	GEFAALD	BlackJack wordt niet herkend in het spel.
7 Kapot	GESLAAGD	
8 Dealer Turn	GESLAAGD	
9 Split	GEFAALD	De functionaliteit splitsen ontbreekt in spel.
10 Game Outcome	GESLAAGD	
11 Edge Case	GESLAAGD	

Voor meer details zie de testresultaten in de voorgaande paragrafen.

Op basis van de resultaten van de uitgevoerde tests kunnen we de volgende conclusies trekken:

1. **Spel initialisatie:** Het initiëren van het spel is zonder problemen verlopen.
2. **Spel Start:** Er is een probleem met het zichtbaar maken van één kaart voor de deler tijdens het starten van het spel.
3. **Hit:** Deze functionaliteit werkt zoals verwacht.
4. **Stand:** Ook deze functie presteert correct.
5. **Count Hand(s):** Het tellen van handen werkt goed.
6. **BlackJack:** Het spel herkent BlackJack situaties niet, wat een groot probleem kan vormen voor de spelervaring.
7. **Kapot:** Deze functie werkt naar behoren.
8. **Dealer Turn:** De beurt van de dealer verloopt zoals verwacht.
9. **Split:** De functionaliteit om te splitsen ontbreekt in het spel, wat een belangrijk gemis is.

10. **Game Outcome:** De uitkomst van het spel wordt correct berekend en weergegeven.

11. **Edge Case:** Verkeerde inputs worden goed afgehandeld.

De testen zijn uitgevoerd op twee verschillende versies van python zonder problemen. Zowel PyCharm als Wing platforms geven een goede output van de testresultaten. PyCharm geeft iets meer overzicht van de behaalde resultaten, terwijl Wing iets meer terugkoppeling geeft bij specifieke problemen.

Samenvatting van bevindingen:

Kritieke issues:

1. BlackJack wordt niet herkend en is een groot gemis voor de spelervaring.
2. Splitsen van kaarten is niet ingebouwd en is een tekortkoming van het programma.

issue:

Het niet zichtbaar maken van de kaarten van de deler is een gemis voor de spelervaring, maar niet kritisch.

Minder kritieke issues:

Het spel is eigenlijk een vereenvoudigde versie van het echte blackjack spel, omdat

- a) er geen multiple spelers mogelijk zijn.
- b) er niet met meerdere dekken kaarten wordt gespeeld.

Dit is voor de spelervaring niet kritisch, maar wel een beperking.

Conclusie:

Er zijn dus enkele kritieke punten die aangepakt moeten worden, zoals het niet herkennen van BlackJack en het ontbreken van de splits functionaliteit. Over het algemeen presteren het spel en de functionaliteiten goed in de geteste gevallen.

Opmerkingen van de tester:

Het is belangrijk om de kritieke issues met prioriteit aan te pakken, omdat deze een negatieve gebruikerservaring veroorzaken.

Voor testcase TC 4: Het geven van input voor stand (s) in deze test om hit, hit, stand te simuleren is onlogisch en komt ook niet overeen met de output van de testcase TC4. Dit is verwarrend.

5 VERBETERVOORSTELLEN

In deze paragraaf worden de verbetervoorstellen gedaan voor het blackjack spel. Er zijn twee soorten verbetervoorstellen:

- Verbeteringen door gebruikerservaring.
- Verbeteringen door testervaring (en door nieuw inzicht aan de hand van e-learning tools).

Dit document is met de code intern beoordeeld door twee collega's van BEE-Ideas, zie Annex C Bevindingen. Deze verbetervoorstellen zijn hierin ook meegenomen.

5.1 De verbeteringsvoorstellen a.d.h.v. gebruikerservaringen

Zoals al aangegeven in sectie 3.1.1 Aannames en beperkingen is het spel ontworpen voor een enkele speler. De speler heeft geen beginsaldo en kan geen inzet bepalen. Bovendien worden niet alle blackjackregels gevolgd. Het spel herkent geen blackjack, splitsen is niet mogelijk en inzetverdubbeling bij twee kaarten is niet toegestaan. De verbeteringsvoorstellen zijn samengevat in Tabel 3. Sommige verbeteringen zijn belangrijker dan andere. Dit is aangegeven in de kolom prioriteit. Bij prioriteit met waarde 'Hoog', is de verbetering noodzakelijk, bij waarde 'Medium' is de verbetering significant, maar niet onoverkomelijk. Een waarde 'Laag' duidt op een "nice to have" verbetering.

Tabel 3 bevindingen en verbetervoorstellen a.d.h.v. gebruikerservaring

Bevinding	Verbetering	Prioriteit	Opmerkingen
Spel is gemaakt voor 1 speler	Breid het spel uit voor meerdere spelers (voer functionaliteit toe voor meerdere spelers), zie ook Annex C Bevindingen.	Medium	Code is hierop redelijk goed op aan te passen. Kans dat 1 dek kaarten dan te weinig is. Dit zou dan moeten worden uitgebreid en is niet veel extra werk.
De kaarten van de deler zijn onzichtbaar bij start spel	Voeg functionaliteit toe voor het zichtbaar maken van 1 kaart deler bij start,), zie ook Annex C Bevindingen.	Hoog	Voor het toevoegen van zichtbaar maken van 1 kaart deler is niet veel code nodig en vergt niet veel testwerk. Voorstel Code is gegeven in Annex B Code Verbeteringen.
Speler kan geen inzet doen	Voeg functionaliteit voor inzet toe, zie ook Annex C Bevindingen.	Medium	Voor het toevoegen van inzet doen is niet veel code nodig en vergt niet veel testwerk.
Spel herkent geen blackjack	Voeg functionaliteit voor herkennen splitsen toe,), zie ook	Hoog	Voor het toevoegen van splitsen is niet veel code nodig en vergt niet veel testwerk. Er is al test functionaliteit geschreven

Bevinding	Verbetering	Prioriteit	Opmerkingen
	Annex C Bevindingen.		, zie 3.2.12 De Testcode voor TC6.
Spel herkent geen splitsen	Voeg functionaliteit voor splitsen toe, zie ook Annex C Bevindingen.	Hoog	Voor het toevoegen van splitsen is wel veel code nodig en vergt veel testwerk.
Spel laat verzekeren voor de blackjack (deler) niet toe	Voeg functionaliteit voor verzekeren toe.	Laag	Code is hierop redelijk goed op aan te passen, maar zal wel wat testwerk kosten. Voorwaarde is dan dat de eerste kaart van de deler zichtbaar is (zie boven).
Spel laat geen verdubbeling inzet bij 9,10, 11 punten na 2 kaarten toe	Voeg functionaliteit voor verzekeren toe.	Laag	Code is hierop redelijk goed op aan te passen
Spel vraagt speler bij 21 punten om een kaart te nemen of te passen (h/s). Het is onlogisch om bij de maximale score van 21 punten de speler te vragen om een kaart.	Voeg functionaliteit toe dat speler automatisch past bij (de maximale score van) 21 punten	Laag	Code is hierop redelijk goed op aan te passen

5.2 De verbeteringsvoorstellen a.d.h.v. testervaringen

Bij het uitvoeren van de testen zijn er een aantal verbeteringsinzichten ontstaan. Sommige inzichten kwamen naar voren tijdens het uitvoeren van de testen en het inspecteren van de code, terwijl andere inzichten werden opgedaan na het bekijken van video's via de LinkedIn e-learning tool. De verbetervoorstellen worden hier verder uiteengezet:

De huidige structuur van de te testen code biedt een goed overzicht voor eenvoudige uitvoering van tests, maar maakt automatisch testen lastiger;

Over het algemeen biedt de structuur van de te testen code in de bijbehorende mappen (zie Testopzet en testprocedure) een goed overzicht van de tests, waardoor deze eenvoudig kunnen worden uitgevoerd. Echter, deze structuur maakt automatisch testen minder eenvoudig. Momenteel is het aantal tests nog redelijk beperkt, maar bij uitbreiding is het verstandig de structuur zodanig op te zetten dat automatisch testen mogelijk wordt. Dit wordt beschouwd als een verbeter voorstel.

Voor TC4 Test Code:

De testcode is nog niet helemaal betrouwbaar en bevat slechts één scenario, maar is uit te breiden;

De testcode in TC4 is niet volledig betrouwbaar (zie ook Overzicht testresultaten en conclusies van de uitgevoerde tests), omdat ik met behulp van de printfunctie van

Python ontdekte dat er al 4 kaarten werden uitgedeeld (twee aan speler en twee aan deler), die later verdwenen zijn. Dit zal moeten worden verbeterd. Daarnaast simuleert TC4 slechts één scenario, namelijk het “hit, hit, stand”-scenario. Een logische optie is om ook het scenario dat de speler meteen past te testen. Voor meer robuustheid zouden deze testcase dus moeten worden uitgebreid naar meerdere test scenario's en het verdwijnen van kaarten moet worden voorkomen.

Uit de testcodes van TC4,5,7 en 11 blijkt dat de code meer “test-driven” zou kunnen zijn ontwikkeld;

Als de code meer “test-driven” was ontwikkeld, zou het niet nodig zijn geweest om sommige functionaliteiten dubbel te testen. Zie bijvoorbeeld testcases TC5 (Count Hand(s)) en TC7 (Kapot). Beide testcases maken gebruik van dezelfde functionaliteit `calculate_hand_value`. Dit geldt ook voor testcases TC4 (stand) en TC11 (edge case), die beide de functionaliteit `play_blackjack` gebruiken. Met een meer “test-driven” benadering van het opstellen van de code zou het opnieuw testen van dezelfde functionaliteiten kunnen worden voorkomen. Het principe van een test-driven development (TDD), waarbij het schrijven van testen voorafgaat aan de implementatie, zorgt ervoor dat de code goed getest en robuust is. Dit draagt bij aan een stabielere en beter onderhoudbare code. Dit is een verbetervoorstel voor het proces, zie ook Annex C Bevindingen. Het aantal testcases aan de hand van de gedefinieerde functionaliteiten had dan kunnen worden ingekort.

Voor TC5 Test Code:

De testcode zou kunnen worden uitgebreid;

Het uitbreiden van de test voor de `calculate_hand_value()`-functie om verschillende scenario's met meerdere azen te verifiëren, kan een verbetering zijn, en zou testcase TC5 (Count Hand(s)) nauwkeuriger maken. Als de testen dan allemaal slagen kan het spel blackjack als meer robuust worden verklaard.

Voor TC8 Test Code:

Uit de test blijkt dat het een system test is en dat het mogelijk om de testcode in te korten;

Bij het schrijven van testcases voor TC8 (Dealer Turn) was het noodzakelijk om meerdere functionaliteiten te importeren uit de code om de testcase te kunnen schrijven en testen. Hierdoor lijkt het erop dat test case TC8 meer een systeemtest is dan een unit test. Een verbetering zou kunnen zijn als deze test als systeem test was aangemerkt en in de opzet van het testprogramma wordt aangepast (zie Testopzet en testprocedure). Ook geldt dat de testcode van testcase TC8 (Dealer Turn) met behulp van “pytest fixtures” verkort had kunnen worden, waardoor codeduplicatie voor de (herhaalde) test setup zou worden voorkomen. Voorgestelde verbeterde testcode is te vinden in Annex B Code Verbeteringen.

Voor TC10 Test Code:

Met een simpele aanpassing in de code had de test gereduceerd kunnen worden tot een unittest;

De systeem testcase TC10 Game Outcome gebruikt nu alle functies om te kijken of het spel de uitkomst van het spel goed concludeert. Het is echter redelijk eenvoudig de speluitkomst te controleren als er een eigen definitie wordt gemaakt die de kaarten van de speler met de kaarten van de deler vergelijkt. De definitie zou kunnen heten `bepaal_uitkomst` (`determine_outcome`) en kan een aparte functie in de code zijn die

kan worden opgeroepen. Zo kan TC10 Game Outcome uiteindelijk worden vereenvoudigd tot een unittest. De voorgestelde (verbeterde) code is te vinden in Annex B Code Verbeteringen en is een verbetering voor het proces.

Voor alle Test Cases geldt dat er meer “negative testing” gedefinieerd had kunnen worden;

De testen bevatten vooral “happy paths” maar weinig “sad paths” of “negative case testing”. Een suggestie is om voor de functie `calculate_hand_value()` te testen met een ongeldige kaart (zie ook Annex C Bevindingen). De handwaarde moet dan worden berekend met een ongeldige kaart in de hand. Het verwachte gedrag is dat de functie een uitzondering moet genereren of de ongeldige kaart zou moeten negeren. Deze “sad path test” zou ervoor zorgen dat het blackjack spel dit onverwachte of ongeldige scenario correct afhandelt, waardoor het spel robuuster en gebruiksvriendelijker wordt. Dit is een verbetervoorstel voor het proces.

Een nieuwe testconfiguratie met de twee Python versies op de gegeven platforms zou kunnen worden uitgevoerd;

Met de huidige tools kan een andere testconfiguratie worden getest om de robuustheid van het blackjackspel verder aan te tonen. Python 3.8.10 is momenteel gekoppeld aan het Wing Personal-platform (zie Testopzet en testprocedure), maar kan ook worden gekoppeld aan het PyCharm-platform. Bovendien kan Python 3.13.1 worden gekoppeld aan het Wing Personal-platform. Dit zou voor alle tests nieuwe resultaten opleveren en indien het werkt, kan de blackjackcode als robuuster worden beschouwd.

ANNEX A BLACKJACK CODE

De code van het blackjack spel BlackJack.py version v1.0 (25 januari 2025).

```
1  import random
2
3
4  # Function to create a deck of cards
5  def create_deck():
6      deck = []
7      suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
8      ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen',
9 'King', 'Ace']
10     for suit in suits:
11         for rank in ranks:
12             deck.append((rank, suit))
13     return deck
14
15 # Function to calculate the value of a hand
16 def calculate_hand_value(hand):
17     value = 0
18     ace_count = 0
19     for card in hand:
20         rank, suit = card
21         if rank in ['Jack', 'Queen', 'King']:
22             value += 10
23         elif rank == 'Ace':
24             ace_count += 1
25             value += 11
26         else:
27             value += int(rank)
28
29     # Adjust for aces
30     while value > 21 and ace_count:
31         value -= 10
32         ace_count -= 1
33
34     return value
35
36
37 # Function to display the hand
38 def display_hand(hand, name):
39     print(f"{name}'s hand:")
40     for card in hand:
41         rank, suit = card
42         print(f"{rank} of {suit}")
43     print(f"Value: {calculate_hand_value(hand)}")
44     print()
45
46
47 # Function to deal a card
48 def deal_card(deck, hand):
49     card = random.choice(deck)
50     deck.remove(card)
51     hand.append(card)
52
```

```
53
54 # Main function to play blackjack
55 def play_blackjack():
56     deck = create_deck()
57     random.shuffle(deck)
58
59     player_hand = []
60     dealer_hand = []
61
62     for _ in range(2):
63         deal_card(deck, player_hand)
64         deal_card(deck, dealer_hand)
65
66     # Player's turn
67     while True:
68         display_hand(player_hand, "Player")
69         if calculate_hand_value(player_hand) > 21:
70             print("Player busts! Dealer wins.")
71             return
72         choice = input("Do you want to hit or stand? (h/s): ").lower()
73         if choice == 'h':
74             deal_card(deck, player_hand)
75         elif choice == 's':
76             break
77         else:
78             print("Invalid choice. Please enter 'h' to hit or 's' to stand.")
79
80     # Dealer's turn
81     while calculate_hand_value(dealer_hand) < 17:
82         deal_card(deck, dealer_hand)
83
84     display_hand(dealer_hand, "Dealer")
85
86     player_value = calculate_hand_value(player_hand)
87     dealer_value = calculate_hand_value(dealer_hand)
88
89     if dealer_value > 21:
90         print("Dealer busts! Player wins.")
91     elif dealer_value > player_value:
92         print("Dealer wins.")
93     elif dealer_value < player_value:
94         print("Player wins.")
95     else:
96         print("It's a tie!")
97
98
99 # Run the game
100 play_blackjack()
```


ANNEX B CODE VERBETERINGEN

Deze Annex bevat voorstellen van zowel losse stukken voor het verbeteren van het blackjack spel als de uiteindelijke verbeterde blackjack code en het testen hiervan zoals aangegeven in 5 Verbetervoorstellen.

Functionaliteit voor het laten zien van 1 of 2 kaarten:

```
# Functionaliteit om kaart(en) te laten zien van de hand
def display_hand(hand, name, hide_second_card=False):
    print(f"{name}'s hand:")
    for i, card in enumerate(hand):
        rank, suit = card
        if hide_second_card and i == 1:
            print("Hidden")
        else:
            print(f"{rank} of {suit}")
    if not hide_second_card:
        print(f"Value: {calculate_hand_value(hand)}")
    print()
...

# Player's turn
while True:
    display_hand(dealer_hand, "Dealer", hide_second_card=True)
    display_hand(player_hand, "Player")
...
```

Functionaliteit voor het checken van blackjack:

```
# Functionaliteit om blackjack te herkennen
def is_blackjack(hand):
    values = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9,
              '10': 10, 'Jack': 10, 'Queen': 10, 'King': 10, 'Ace': 11}

    # Check dat het aantal in de hand precies 2 kaarten is
    if len(hand) != 2:
        return False

    # Reken de total waarde uit van de hand
    total_value = sum(values[card[0]] for card in hand)

    # Check voor blackjack (een aas en een 10-punten kaart)
    if total_value == 21 and any(card[0] == 'Ace' for card in hand):
        return True

    return False
...

# Check voor initieel blackjack
if is_blackjack(player_hand):
    display_hand(player_hand, "Player")
    display_hand(dealer_hand, "Dealer")
    print("Player has Blackjack! Player wins.")
    return
```

```
elif is_blackjack(dealer_hand):
    display_hand(player_hand, "Player")
    display_hand(dealer_hand, "Dealer")
    print("Dealer has Blackjack! Dealer wins.")
    return

# Player's turn
...
```

Voorstel code om de speluitkomst te bepalen:

```
# Functie om de speluitkomst te bepalen

def determine_outcome(dealer_value, player_value):
    if dealer_value > 21:
        return "Dealer busts! Player wins."
    elif dealer_value > player_value:
        return "Dealer wins."
    elif dealer_value < player_value:
        return "Player wins."
    else:
        return "It's a tie!"

...
outcome = determine_outcome(dealer_value, player_value)
print(outcome)
```

Voorstel tot toevoegen van pytest fixture functionaliteit voor het testen van Dealer Turn:

```
import pytest
from BlackJack import create_deck, calculate_hand_value, deal_card

@pytest.fixture
def setup_deck():
    return create_deck()

@pytest.fixture
def setup_dealer_hand():
    return []

def test_dealer_stands_17(setup_deck, setup_dealer_hand):
    deck = setup_deck
    dealer_hand = [('10', 'Hearts'), ('7', 'Spades')]
    print(f"\nstart waarde: {calculate_hand_value(dealer_hand)}")
    while calculate_hand_value(dealer_hand) < 17:
        deal_card(deck, dealer_hand)
    value = calculate_hand_value(dealer_hand)
    assert value >= 17, "Dealer should stand with hand value of at least 17."
    print(f"eind waarde: {calculate_hand_value(dealer_hand)}")

def test_dealer_stands_19(setup_deck, setup_dealer_hand):
    deck = setup_deck
    dealer_hand = [('10', 'Hearts'), ('9', 'Clubs')]
    print(f"\nstart waarde: {calculate_hand_value(dealer_hand)}")
    while calculate_hand_value(dealer_hand) < 17:
```

```
        deal_card(deck, dealer_hand)
    value = calculate_hand_value(dealer_hand)
    assert value >= 17, "Dealer should stand with hand value of at least 17."
    print(f"eind waarde: {calculate_hand_value(dealer_hand)}")

def test_dealer_hits(setup_deck, setup_dealer_hand):
    deck = setup_deck
    dealer_hand = [('5', 'Hearts'), ('9', 'Spades')]
    print(f"\nstart waarde: {calculate_hand_value(dealer_hand)}")
    while calculate_hand_value(dealer_hand) < 17:
        deal_card(deck, dealer_hand)
    value = calculate_hand_value(dealer_hand)
    assert value >= 17, "Dealer should hit until hand value is at least 17."
    print(f"eind waarde: {calculate_hand_value(dealer_hand)}")

def test_dealer_busts_23(setup_deck, setup_dealer_hand):
    deck = setup_deck
    dealer_hand = [('10', 'Hearts'), ('6', 'Spades')]
    print(f"\nstart waarde: {calculate_hand_value(dealer_hand)}")
    while calculate_hand_value(dealer_hand) <= 21:
        dealer_hand.insert(0, ('7', 'Hearts'))
    value = calculate_hand_value(dealer_hand)
    assert value > 21, "Dealer should bust with hand value greater than 21."
    print(f"eind waarde: {calculate_hand_value(dealer_hand)}")

def test_dealer_busts_25(setup_deck, setup_dealer_hand):
    deck = setup_deck
    dealer_hand = [('10', 'Hearts'), ('5', 'Spades')]
    print(f"\nstart waarde: {calculate_hand_value(dealer_hand)}")
    while calculate_hand_value(dealer_hand) <= 21:
        dealer_hand.insert(0, ('10', 'Diamonds'))
    value = calculate_hand_value(dealer_hand)
    assert value > 21, "Dealer should bust with hand value greater than 21."
    print(f"eind waarde: {calculate_hand_value(dealer_hand)}")

def test_dealer_ace_adjustment_1(setup_deck, setup_dealer_hand):
    deck = setup_deck
    dealer_hand = [('Ace', 'Hearts'), ('6', 'Spades')]
    print(f"\nstart waarde: {calculate_hand_value(dealer_hand)}")
    while calculate_hand_value(dealer_hand) < 17:
        dealer_hand.insert(0, ('10', 'Hearts'))
    value = calculate_hand_value(dealer_hand)
    assert 17 <= value <= 21, "Dealer hand value should be adjusted correctly with an Ace."
    print(f"eind waarde: {calculate_hand_value(dealer_hand)}")

def test_dealer_ace_adjustment_2(setup_deck, setup_dealer_hand):
    deck = setup_deck
    dealer_hand = [('Ace', 'Hearts'), ('5', 'Spades')]
    print(f"\nstart waarde: {calculate_hand_value(dealer_hand)}")
    while calculate_hand_value(dealer_hand) < 17:
        dealer_hand.insert(0, ('Ace', 'Clubs'))
    value = calculate_hand_value(dealer_hand)
    assert 17 <= value <= 21, "Dealer hand value should be adjusted correctly with two Aces."
    print(f"eind waarde: {calculate_hand_value(dealer_hand)}")

if __name__ == '__main__':
    pytest.main()
```

De uiteindelijke verbeterende blackjack code is hieronder gegeven:

```
#####
#Lodewijk Aris  version 2, date 16 februari  2025                                     #
#                                     version 1, date 30 januari  2025                                     #
#De verbeterde BlackJack code na testen en leren                                     #
#####

import random

# Function to create a deck of cards
def create_deck():
    deck = []
    suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King', 'Ace']
    for suit in suits:
        for rank in ranks:
            deck.append((rank, suit))
    return deck

# Function to calculate the value of a hand
def calculate_hand_value(hand):
    value = 0
    ace_count = 0
    for card in hand:
        rank, suit = card
        if rank in ['Jack', 'Queen', 'King']:
            value += 10
        elif rank == 'Ace':
            ace_count += 1
            value += 11
        else:
            value += int(rank)

    # Adjust for aces
    while value > 21 and ace_count:
        value -= 10
        ace_count -= 1

    return value

# Functionaliteit om kaart(en) te laten zien van de hand
def display_hand(hand, name, hide_second_card=False):
    print(f"{name}'s hand:")
    for i, card in enumerate(hand):
        rank, suit = card
        if hide_second_card and i == 1:
            print("Hidden")
        else:
            print(f"{rank} of {suit}")
    if not hide_second_card:
        print(f"Value: {calculate_hand_value(hand)}")
    print()

# Function to deal a card
```

```
def deal_card(deck, hand):
    card = random.choice(deck)
    deck.remove(card)
    hand.append(card)

# Functie om de speluitkomst te bepalen
def determine_outcome(dealer_value, player_value):
    if dealer_value > 21:
        return "Dealer busts! Player wins."
    elif dealer_value > player_value:
        return "Dealer wins."
    elif dealer_value < player_value:
        return "Player wins."
    else:
        return "It's a tie!"

# Main function to play blackjack
def play_blackjack():
    deck = create_deck()
    random.shuffle(deck)

    player_hand = []
    dealer_hand = []

    for _ in range(2):
        deal_card(deck, player_hand)
        deal_card(deck, dealer_hand)

    # Player's turn met verborgen kaart deler
    while True:
        display_hand(dealer_hand, "Dealer", hide_second_card=True)
        display_hand(player_hand, "Player")
        if calculate_hand_value(player_hand) > 21:
            print("Player busts! Dealer wins.")
            return
        choice = input("Do you want to hit or stand? (h/s): ").lower()
        if choice == 'h':
            deal_card(deck, player_hand)
        elif choice == 's':
            break
        else:
            print("Invalid choice. Please enter 'h' to hit or 's' to stand.")

    # Dealer's turn
    while calculate_hand_value(dealer_hand) < 17:
        deal_card(deck, dealer_hand)

    display_hand(dealer_hand, "Dealer")

    player_value = calculate_hand_value(player_hand)
    dealer_value = calculate_hand_value(dealer_hand)

    outcome = determine_outcome(dealer_value, player_value)
    print(outcome)

# Run the game
play_blackjack()
```

ANNEX C BEVINDINGEN

Er is een interne review gedaan binnen het ICT team van de BEE_ideas collega's. De revisors zijn Tina Slot (begeleidster BEE-Ideas) en Corné Brouwer.

Aanbevelingen Tina Slot: "Hierbij mijn bevindingen en knap staaltje werk qua codering en updating bestaand spel naar hoger herkenbaar niveau inclusief uitgevoerde tests!"

Voor het testen van het blackjackspel zijn verschillende testcases gedefinieerd, die de functionaliteiten van het spel valideren. Hier zijn enkele aanbevelingen op basis van de testresultaten:

Aanbevelingen

- **Herkenning van Blackjack:**

- **Probleem:** Het spel herkent geen blackjack-situaties.
- **Aanbeveling:** Voeg een functie toe die controleert of de speler blackjack heeft (een aas en een kaart ter waarde van 10) en zorg ervoor dat dit correct wordt afgehandeld in de spelregels

- **Functionaliteit voor Splitsen:**

- **Probleem:** De mogelijkheid om kaarten te splitsen ontbreekt.
- **Aanbeveling:** Implementeer de splits-functionaliteit zodat spelers hun hand kunnen splitsen als ze twee kaarten van dezelfde waarde hebben. Dit is een belangrijke functie in blackjack

- **Zichtbaarheid van de Deler's Kaarten:**

- **Probleem:** De tweede kaart van de deler is zichtbaar bij de start van het spel.
- **Aanbeveling:** Zorg ervoor dat de tweede kaart van de deler verborgen blijft totdat het de beurt van de deler is om te spelen

- **Meerdere Spelers:**

- **Probleem:** Het spel is ontworpen voor slechts één speler.
- **Aanbeveling:** Breid het spel uit om meerdere spelers toe te staan, wat de gebruikerservaring zou verbeteren

- **Inzetfunctionaliteit:**

- **Probleem:** Spelers kunnen geen inzet doen.
- **Aanbeveling:** Voeg een functionaliteit toe voor spelers om hun inzet te bepalen voordat het spel begint

- **Negatieve Testcases:**

- **Probleem:** Er zijn voornamelijk positieve testcases uitgevoerd.
- **Aanbeveling:** Implementeer negatieve testcases om te controleren hoe het spel omgaat met ongeldige invoer, zoals het invoeren van een niet-bestaande kaart

- **Testgedreven Ontwikkeling:**

- **Aanbeveling:** Overweeg om de ontwikkeling van de code meer test gedreven te maken, zodat functionaliteiten niet dubbel getest hoeven te worden en de code robuuster is

s

Conclusie

Het is essentieel om de kritieke punten, zoals de herkenning van blackjack en de splits-functionaliteit, met prioriteit aan te pakken. Dit zal niet alleen de spelervaring verbeteren, maar ook de algehele functionaliteit van het spel versterken. Daarnaast is als aanbeveling een robuuste test gedreven codering een win/win situatie. (korte tijd van testen met maximale uitkomst).

Feedback van Corné Brouwer:

Feedback op opdracht van Lodewijk Aris

Pagina/hoofdstuk	Advies	
Pagina 4, hoofdstuk 1	woordkeuze	Je hebt het over een 'klein programma'. Uitleg geven wat je hiermee precies bedoelt. Bestand met kort geschreven stuk code?
Pagina 4, hoofdstuk 1	grammaticaal	Je mist een voorzetsel. Verkreeg ik een code om het blackjackspel te spelen.
Pagina 6, hoofdstuk 2	Opbouw	Misschien het hoofdstuk nummer doorgeven van 'aannames en verbeteringen' of haal dit hoofdstuk naar voren dat het duidelijk is wat de restricties zijn.
Pagina 6, hoofdstuk 2	Opbouw	Bij punt 2; uitdelen van kaarten is verschillend in Blackjack1 en Blackjack2, wat je visueel ziet.
Pagina 6 hoofdstuk 2	Grammaticaal	Hieruit zijn de volgende test casus zijn (dubbel)
Pagina 6 hoofdstuk 2	Uitleg	Tabel, punt 1, leg uit wat een dek kaarten is, bestaan uit 52 kaarten.
		Tabel, punt 2, verschilt per Blackjack1 of Blackjack2 of je visueel ziet of kaarten ook aan de dealer worden gedeald.
Pagina 7 hoofdstuk 2	Uitleg	Tabel figuur 1, punt 3. Bij punt 2 heb je de

		speler al twee open kaarten gegeven.
		Punt 6, ook een kaart met 10 maakt blackjack samen met een A. nadat je blackjack hebt, krijg je nu de optie om een kaart te kiezen, dat klopt niet.
Pagina 8, hoofdstuk 2	Uitleg	Punt 8, voordat de dealer verder gaat als de dealer lager heeft dan 17, draait de dealer eerst de twee kaart open.
	Uitleg	Punt 9, elke losse kaart krijgt beide 1 open kaart. Extra betaaloctie bij het splitsen? Je geeft aan dat ze beide 1 kaart kunnen krijgen, maar de speler krijgt de mogelijkheid om meerdere kaarten te krijgen tot dat de speler past (s)
Pagina 9 hoofdstuk 2		10. als een speler bust, moet de dealer dan als nog zijn kaarten laten zien en onder de 17 een kaart trekken waardoor de bank ook kan busten?
Pagina 9 hoofdstuk 2		Wordt er uitgelegd wat je met H(it) en (S)and/Pass bedoelt?
		Uitleg wat functies zijn in Python/Pycharm
		Wat is een patchfunctie?
		Unit test van python, wat doe je daarmee?
Pagina 10 hoofdstuk 3, inleiding		Beide versies zijn geschikt voor de testvereisten, waar staan die en zijn die door jou vastgesteld?
Pagina 10, hoofdstuk 3.1		Je hebt het over platformen, 1 gebruikt een interpreter, wat is dit en die andere gebruikt dit niet?
Pagina 13, 3.1.3		Maw?
		Test dat je na de twee kaarten voor de speler al kan passen en geen kaarten kan vragen.
39, 4.11.1		Heb je bij de ener laatste zelf het wordt

		'invalid' ingevuld als input?
		Wanneer je als de dealer 23 hebt, waarom krijg je dan nog de optie om h of s in te voeren? In Wing omgeving, laatste onderdeel
42 samenvattingen en bevindingen		Issue, dat een kaart van de dealer niet open is, heeft wel degelijk invloed op deze van de speler hoeveel kaarten je vraagt
44 5.2		Wat bedoel je met 'test-driven'

Dit is de laatste pagina van het document.