Paradigma Onderzoek

APP22 2022-2023

Osama Halabi
Sn: 628160

HBO ICT Software Development 2022/2023.

Docent: Michel Koolwaaij

Arnhem *10-03-2023*

v. 6.5

Inhoudsopgave

SAMENVATTING

Dit onderzoek richt zich op het leren van de functionele programmeertaal Haskell en het implementeren van het spel 'Connect four' met behulp van Haskell. Het doel is om te onderzoeken hoe Haskell-concepten kunnen worden toegepast bij de ontwikkeling van het spel en zo zoveel mogelijk over Haskell te leren. De onderzoeksvragen omvatten wat Haskell is, wat Haskell concepten zijn, hoe deze concepten kunnen worden geïmplementeerd in het spel, en het verschil tussen functioneel programmeren en object georiënteerd programmeren.

Het onderzoek maakt gebruik van literatuuronderzoek en codeanalyse om de vragen te beantwoorden. De belangrijkste Haskell concepten die behandeld worden zijn Higher-Order Functions, Algebraic Data Types, Pattern Matching, Lambda Functions, Immutability, Recursion, Laziness, Type Inference, Layout Sensitivity en Garbage Collection. Deze concepten worden geïmplementeerd in het spel, waarbij onder andere pattern matching wordt gebruikt om te bepalen of een speler heeft gewonnen, en lambda functies worden gebruikt om patronen van cellen te controleren.

Het onderzoek concludeert dat Haskell een moderne en veelzijdige programmeertaal is die geschikt is voor het ontwikkelen van complexe en logische programma's. Het verschil tussen functioneel programmeren en object georiënteerd programmeren wordt benadrukt, waarbij FP zich richt op functies en datatransformaties, en OOP op het modelleren van de werkelijkheid door middel van objecten. Het onderzoek draagt bij aan het begrip van Haskell en functioneel programmeren door de praktische toepassing in het Connect four spel.

1 INLEIDING

In dit onderzoek wil ik me richten op het leren van de programmeertaal Haskell en het implementeren van een connect spel met behulp van Haskell. Vervolgens relateer ik het connect spel aan Haskell concepten. Het doel is om te bestuderen hoe Haskell-concepten kunnen worden toegepast bij de ontwikkeling van het connect spel.

Hoofdvraag:

Hoe leer ik zoveel mogelijk over Haskell door het Connect spel te maken?

Deelvragen:

1- Wat is Haskell?
2- Wat zijn Haskell concepten?
3- Hoe kunnen deze Haskell concepten worden geïmplementeerd in de connect spel?
4- Wat is het verschil tussen FP-taal en OOP?

2 METHODENKAART

In dit hoofdstuk beschrijf ik welke onderzoekmethoden ga ik gebruiken om de hoofvraag en deelvragen te kunnen beantwoorden.

Om deze vragen te beantwoorden, gaan ik gebruik maken van literatuuronderzoek en codeanalyse. Literatuuronderzoek zal me helpen om de relevante Haskell concepten te identificeren en te begrijpen hoe deze kunnen worden toegepast in de ontwikkeling van het connect spel.

Codeanalyse zal me helpen om de geïdentificeerde Haskell concepten te implementeren in dit spel. Hiermee wil ik nagaan hoe deze concepten in de praktijk toegepast kunnen worden en of ze bijdragen aan de ontwikkeling van het spel.

Literatuur waar ik mee ga beginnen:

1- Introductory Haskell course of the University of Pennsylvania (CIS194)
2- haskell-beginners-2022

Beide cursussen bieden me de gelegenheid om de basisprincipes van Haskell te leren en vertrouwd te raken met de syntax en de functies van deze taal. Deze cursussen bieden ook de mogelijkheid om deze kennis toe te passen door een aantal oefeningen en projecten te maken.

Bronnen:

- *Introductory Haskell course of the University of Pennsylvania*
- *haskell-beginners-2022*

3 RESULTATEN

In dit hoofdstuk beantwoord ik alle deelvragen, dat zou me kunnen helpen om mijn hoofdvraag te kunnen beantwoorden.

3.1 Wat is Haskell

Aan het einde van de jaren '80 brak er een nieuw tijdperk aan in de wereld van programmeertalen en paradigma's. Een groep onderzoekers werkte hard aan de ontwikkeling van Haskell, een functionele programmeertaal. Na veel hard werk werd Haskell uiteindelijk in 1990 gepresenteerd en vernoemd naar de beroemde Amerikaanse wiskundige Haskell Brooks Curry.

Haskell is een puur functionele programmeertaal die logica en extreem gecompliceerde berekeningen verwerkt door functies te definiëren en te combineren. De voordelen zijn onder meer dat het een ideale keuze is voor het maken van complexe programma's en logische processen.

Haskell is een moderne en veelgebruikte functionele programmeertaal die volledig losstaat van andere talen. Het is ontworpen om toepassingen te ondersteunen die variëren van cijfers tot symbolen. De expressieve syntaxis en het uitgebreide type systeem zorgen ervoor dat ontwikkelaars meer flexibiliteit en controle hebben bij het bouwen van software, waardoor het een ideale keuze is voor hen die op zoek zijn naar meer mogelijkheden.

Haskell is ook zeer geschikt voor parallel en gelijktijdig programmeren, wat betekent dat het gemakkelijk is om programma's te schrijven die tegelijkertijd op meerdere processors of systemen kunnen draaien. Dit maakt het een populaire keuze voor bijvoorbeeld grote datacenters en Cloud omgevingen.

Bronnen:

- *What is Haskell Programming Language?*
- *Haskell Language*
- *Why Haskell?*

3.2 Wat zijn Haskell concepten?

Haskell heeft veel concepten die belangrij zijn om met deze programmeertaal aan de slag te kunnen gaan. Deze concepten zijn allemaal belangrijk om Haskell te leren omdat ze fundamentele eigenschappen en technieken van de taal vertegenwoordigen. Het begrijpen van deze concepten helpt me om Haskell-code te optimaliseren, beter leesbaar te maken en efficiënter te werken met deze taal. Hieronder staan de belangrijkste concepten van Haskell:

3.2.1 Higher-Order Functions (HOF)

Higher order functions zijn functies die als argumenten andere functies kunnen aannemen en/of functies kunnen retourneren als resultaat. Haskell is een functionele programmeertaal die veel gebruik maakt van higher order functions.

3.2.2 Algebraic Data Types (ADT)

ADT's zijn datatypen die worden gedefinieerd door algebraïsche operaties zoals sommen (sum types) en producten (product types) Deze typen worden gedefinieerd door gegevensconstructeurs die gegevens vertegenwoordigen en/of nieuwe waarden creëren.

3.2.3 Pattern Matching

Dit is een manier om waarden te matchen met een patroon (pattern), en vervolgens een bepaalde actie uit te voeren op basis van de overeenkomst.

In Haskell wordt pattern matching vaak gebruikt met functiedefinities en ADT's.

3.2.4 Lambda functie

Een lambda-functie is een functie zonder naam die je kunt gebruiken om een andere functie te definiëren of als argument aan een hogere-orde functie te geven. Een lambda-functie wordt aangeduid met het symbool \ en heeft de vorm \x -> y, waarbij x de parameter(s) is/zijn en y het resultaat. Lambda-functie wordt vaak gebruikt om je code korter en leesbaar te maken.

3.2.5 Immutability

Dit betekent dat eenmaal aangemaakte gegevens niet meer kunnen worden gewijzigd. In plaats daarvan worden nieuwe gegevens gemaakt op basis van bestaande gegevens.

In Haskell worden alle waarden standaard niet beïnvloed. Met andere woorden, het kan niet worden gewijzigd nadat het is gemaakt. In plaats daarvan worden nieuwe waarden gecreëerd op basis van bestaande waarden.

3.2.6 Recursion

Het recursieve concept is van groot belang in Haskell en wordt gebruikt om problemen op te lossen door ze op te delen in kleinere en eenvoudigere subproblemen. Recursieve functies in Haskell worden dgedefinieerd door zichzelf aan te roepen totdat een basisgeval wordt bereikt.

Recursie is een belangrijk concept in Haskell, vooral als het gaat om het verdelen van problemen in kleinere en eenvoudigere deeltjes. Recursie is een belangrijk concept in Haskell dat ik moet begrijpen als ik complexe problemen wil oplossen.

3.2.7 Laziness

Dit betekent dat de evaluatie van de uitdrukking wordt uitgesteld tot het moment dat het resultaat nodig is. Het houdt in dat waarden pas worden geëvalueerd als ze nodig zijn, en de uitvoering van het programma kan worden geoptimaliseerd door alleen die waarden te evalueren die echt nodig zijn.

3.2.8 Type inference

Dit betekent dat de compiler automatisch de types van expressies en functies kan afleiden zonder dat de programmeur ze apart hoeft te definiëren.

In Haskell gebruikt de compiler een type-inferentiesysteem om de types van expressies te bepalen op basis van de context waarin ze worden gebruikt.

3.2.9 layout sensitive

Haskell is een lay-outgevoelige taal, wat betekent dat de lay-out van de code belangrijk is voor de interpretatie van het programma. In Haskell wordt codeplaatsing bepaald door witruimte, inclusief spaties en tabs, per positie.

In tegenstelling tot veel andere programmeertalen gebruikt Haskell geen accolades of sleutelwoorden om het begin en einde van een codeblok aan te geven. In plaats daarvan wordt de code niet door een vast patroon bepaald, maar door de mate waarin de verschillende delen van het programma inspringen.

De basisregel van de Haskell-lay-out is dat alle onderdelen van een structuur even ver uitgelijnd moeten staan. Dit impliceert dat alle regels die tot hetzelfde codeblok behoren in dezelfde kolom moeten worden gestart.

3.2.10 Garbage collector

Haskell gebruikt garbage collection voor het eenvoudiger maken van het geheugenbeheer en het verzekeren van de veiligheid en efficiëntie van programma's. Haskell heeft een eigen runtime systeem en dit systeem heeft een garbage collector. Deze collector controleert automatisch de geheugenruimte van het programma tijdens het garbage collection proces en verwijdert objecten die niet meer in gebruik zijn uit het geheugen.

Haskell maakt objecten door functieaanroepen en plaatst deze objecten in de "heap". Wanneer er geen referenties meer zijn die naar een object verwijzen, is het niet langer bruikbaar. De garbage collector zoekt naar objecten die niet meer gebruikt worden en maakt zo weer ruimte in het geheugen vrij.

De garbage collector van Haskell werkt volgens het "mark-and-sweep" algoritme. Dit algoritme werkt door eerst alle objecten in het geheugen te markeren die nog door het programma in gebruik zijn. Dan worden alle objecten die niet zijn gemarkeerd verwijderd en wordt het geheugen dat ze innemen vrijgemaakt.

Bronnen:

- *Learn You a Haskell for Great Good!*
- *Haskell Beginners 2022*
- *Haskell Tutorial: get started with functional programming*

3.3 Hoe kunnen deze Haskell concepten worden geïmplementeerd in de connect spel?

Ik heb veel van de belangrijkste Haskell-concepten geïmplementeerd in het spel. Sommige waren makkelijker voor mij dan andere om te implementeren. Hieronder staan een aantal van de concepten die mogelijk waren.

3.3.1 Laziness

Ik kan Laziness gebruiken om het spel efficiënter te maken. Door gebruik te maken van de take- en drop-functies kan ik lijnen checken die alleen nodig zijn voor het huidige deel van het spelbord, in plaats van alle rijen en kolommen te controleren. Hierdoor wordt het spel sneller en efficiënter uitgevoerd.

Ik heb dit concept toegepast in de checkWin-functie, die ik in de module board heb gemaakt. Deze functie heeft een bord en speler als parameters en geeft een boolean terug.

3.3.2 Algebraic Data Types (ADT)

Ik kan Abstract Data Types (ADT's) gebruiken om het spelbord en de spelers te definiëren. Met behulp van ADT's kan ik een bord datatype definiëren dat een matrix van cellen bevat. Vervolgens kan ik ook een datatype voor de spelers maken.

Ik heb dit concept toegepast in zowel de board- als de player-module. In de board-module heb ik een bord datatype gemaakt dat een matrix van cel-datatype bevat. Het cel-datatype kan leeg zijn of een waarde van een speler bevatten.

In de player-module heb ik dit concept nogmaals toegepast door een speler-datatype te maken die kan worden ingesteld als X of O. Met andere woorden, X is de eerste speler en O is de tweede speler.

3.3.3 Recursie:

Ik kan recursie gebruiken om het spel door te laten lopen totdat het voorbij is. Hiermee kan ik ook controleren of het spel is afgelopen.

Ik heb dit concept toegepast in de run-functie van de board-module. In deze functie controleer ik of er een winnaar is, of het gelijkspel is en of het spelbord vol is. Als een van deze eisen waar is, wordt het spel beëindigd. Zo niet, dan wordt de run-functie opnieuw aangeroepen met een zet van de volgende speler.

3.3.4 Pattern Matching

Ik kan pattern matching gebruiken om een functie te maken waarmee de speler een zet kan doen. Bovendien kan ik een functie maken die controleert of de speler heeft gewonnen door te controleren of er vier cellen op een rij zijn bezet door deze speler.

Dit concept heb ik toegepast in de checkWin functie van de Board-module. De checkWin functie controleert of er op een rij, kolom of diagonaal vier cellen zijn bezet door dezelfde speler. Als dit het geval is, wordt True geretourneerd.

Daarnaast wordt pattern matching gebruikt in de showCell functie van de printBoard functie. De showCell-functie controleert of een cel leeg is of bezet is door een speler en retourneert op basis daarvan een string die de cel voorstelt.

3.3.5 Higher-order functions

Door gebruik te maken van deze concept kan ik een functie maken die een andere functie als parameter accepteert zodat ik bijvoorbeeld en het spel board kan uitprinten. Ik kan ook hiermee een functie maken die een higher-order- fucntion zoals any om te bepalen of een patroon aanwezig is in de lijst met lijnen om te controleren.

Ik heb dit concept toegepast in de checkWin-functie om te controleren of er op een rij, kolom of diagonaal vier cellen zijn bezet door dezelfde speler.

Daarnaast heb higher-order- fucntion in de isBoardFull gebruikt om te kijken of alle cellen door een speler bezet zijn.

3.3.6 Immutability

Deze concept laat me de waarde van een bestaande niet kunnen veranderen zoals een bestande board invullen. Daarom als ik een cel wil in vullen moet ik een new board teruggeven zodat ik een ingevulde board kan uitprinten ipv van een lege board die gemaakt wordt tijdens het opstarten van het spel.

3.3.7 Lambda functie

Ik kan een lambda-functie gebruiken om de patronen van vier aangrenzende cellen te controleren, zodat dit patroon dynamisch kan worden gegenereerd op basis van de huidige cel positie en de richting waarin wordt gecontroleerd.

3.4 Verschil tussen FP en OOP

Het belangrijkste verschil tussen object georiënteerd programmeren (OOP) en functioneel programmeren (FP) is hoe met data en functies omgaan.

OOP is een programmeerparadigma waarbij data en functies zijn georganiseerd in objecten. Deze objecten beschrijven hun gedrag door middel van methoden. Het doel van OOP is het maken van een computermodel van de werkelijkheid, waarbij objecten gebruikt worden om de abstracties van de echte wereld te weerspiegelen. Een OOP-programma is een verzameling objecten die met elkaar communiceren.

FP stelt daarentegen dat functies de meest waardevolle elementen van een programma zijn. Het is niet nodig de werkelijkheid te modelleren, maar wel de berekening uit te leggen. FP ziet gegevens als een passieve entiteit die je gebruikt als input voor functies. De nadruk ligt op het transformeren van data door middel van functies en het combineren van bestaande functies om nieuwe functies te creëren.

Een ander verschil is dat OOP meestal veranderlijke status toestaat, waarbij je objecten kunt wijzigen terwijl het programma draait. FP produceert altijd dezelfde output voor dezelfde input en werkt met data die niet verandert en met functies zonder neveneffecten.

OOP is gebaseerd op het modelleren van de realiteit met behulp van objecten, terwijl FP berekeningen definieert als functies die geen neveneffecten hebben en de toestand niet veranderen. Beide paradigma's hebben hun sterke en zwakke punten.

Bronnen:

- *Functional programming vs OOP: Which paradigm to use?*
- *Difference between Functional Programming and Object Oriented Programming*

4 CONCLUSIE

In dit onderzoek heb ik onderzocht hoe ik zoveel mogelijk kan leren over Haskell door het implementeren van een Connect spel met behulp van Haskell en het relateren van het spel aan Haskell concepten.

Uit mijn onderzoek blijkt dat Haskell een functionele programmeertaal is die is ontworpen om logica en extreem gecompliceerde berekeningen te verwerken door middel van functies. Haskell is een moderne en veelgebruikte functionele programmeertaal die volledig losstaat van andere talen. Het heeft een uitgebreid type systeem en expressieve syntaxis, waardoor ontwikkelaars meer flexibiliteit en controle hebben bij het bouwen van software.

Ik heb de belangrijkste Haskell-concepten geïdentificeerd, waaronder higher-order functions, algebraic data types (ADT), pattern matching, lambda functies, immutability, recursie, laziness, type inference, layout sensitivity en garbage collection. Deze concepten zijn belangrijk voor het begrijpen van Haskell en het toepassen ervan bij het bouwen van software.

Vervolgens heb ik deze concepten geïmplementeerd in het Connect spel. Door gebruik te maken van deze concepten heb ik een spel gecreëerd dat efficiënt en dynamisch is. Zo kon ik gebruik maken van hogere-orde functies om bijvoorbeeld een functie te creëren die een ander functie als parameter accepteert, bijvoorbeeld een functie om te bepalen of een patroon aanwezig is in de lijst met lijnen om te controleren.

Ik heb ook ADT's gebruikt om het spelbord en de spelers te definiëren. Door middel van pattern matching kan ik bepalen of een speler heeft gewonnen door te controleren of er vier cellen op een rij zijn bezet door deze speler. Ik heb lambda functies gebruikt om patronen van vier aangrenzende cellen te controleren, zodat dit patroon dynamisch kan worden gegenereerd op basis van de huidige celpositie en de richting waarin wordt gecontroleerd.

Tot slot heb ik het verschil tussen OOP en FP besproken. Het belangrijkste verschil is hoe gegevens en functies worden georganiseerd. Bij OOP worden gegevens en functies georganiseerd in objecten die hun gedrag beschrijven door middel van methoden. Bij FP zijn functies de meest waardevolle elementen van een programma en worden gegevens beschouwd als passieve entiteit die wordt gebruikt als input voor functies.

In conclusie, door het maken van een Connect spel met Haskell heb ik veel geleerd over Haskell-concepten en hoe ze kunnen worden toegepast bij het bouwen van software. Haskell biedt ontwikkelaars veel flexibiliteit en controle, waardoor het een ideale keuze is voor het bouwen van complexe software. Haskell is een krachtige taal die een diepgaand begrip vereist, maar de investering in tijd en moeite betaalt zich zeker uit in de vorm van efficiënte, robuuste en dynamische software.

5 LITERATUURLIJST

1- *Lecture notes and assignments*. (n.d.). Penn Engineers University. Retrieved February 2, 2023, from https://www.seas.upenn.edu/%7Ecis1940/spring13/lectures.html

2- Kovanikov, D. (2022). *GitHub - haskell-beginners-2022/course-plan: Haskell course info, plan, video lectures, slides*. GitHub. Retrieved February 2, 2023, from https://github.com/haskell-beginners-2022/course-plan

3- GeeksforGeeks. (2022, March 2). *What is Haskell Programming Language*. https://www.geeksforgeeks.org/what-is-haskell-programming-language/

4- Lipovača, M. (n.d.). *Chapters - Learn You a Haskell for Great Good!* http://learnyouahaskell.com/chapters

5- Khandaker, S. (2022, January 7). *Why Haskell? - Geek Culture*. Medium. https://medium.com/geekculture/why-haskell-a9117c42da12

6- Thelin, R. (2021, February 25). *Haskell Tutorial: get started with functional programming*. Educative: Interactive Courses for Software Developers. https://www.educative.io/blog/haskell-tutorial

7- Dorman, T. (n.d.). *Functional programming vs OOP: Which paradigm to use*. Educative: Interactive Courses for Software Developers. https://www.educative.io/blog/functional-programming-vs-oop

8- GeeksforGeeks. (2022, May 12). *Difference between Functional Programming and Object Oriented Programming*. https://www.geeksforgeeks.org/difference-between-functional-programming-and-object-oriented-programming/