REQUIREMENTS EN ONTWERP

Vier op een rij casus

Datum: 8 oktober 2021

Opleiding: HBO-ICT, Zuyd Hogeschool

Auteurs: Manon Schüle(1805193)

Clara Schoelitsz(1427709)

Anthony Schuman(1939319)

Docent: Drs. Snijders

Inhoudsopgave

1. Inleiding

Als er beslissingen moeten worden gemaakt, of als data moet worden gekozen, worden hier meestal loopjes zoals while of if loopjes voor gebruikt. Bij een ingewikkeld programma zoals een spel of een programma met een grote database, kan zo loop heel groot worden. Om onnodige problemen in het programmeren te voorkomen, kunnen voor dit soort toepassingen een algoritme worden ingezet. Een algoritme is een “recept” om een wiskundig of informatie probleem op te lossen. Een algoritme is een stukje code die informatie op een efficiënte manier kan sorteren. Hierbij moet er rekening worden gehouden met het geheugen van de machine.

1.1. Doel.

Een spel programmeren met behulp van een algoritme die voorruit denkt en werkt als een computer speler.

1.2. Doelstelling.

Voordat “4 op een rij online” live wilde gaan, wilde zij eerst een Proof of Concept zien van de projectgroep. Dit hoefde nog geen mooie interface te hebben. Maar het spel moest wel werken. Zo moet het spel gespeeld kunnen worden door twee spelers of één speler tegen de computer. De computer mag niet altijd winnen, maar moet wel slim genoeg zijn om de winnende zet van spelers te blokkeren. Verder moet het bord moest 6 bij 7 groot zijn.

Het was voor de projectgroep dan ook belangrijk om een werkend algoritme op te leveren doormiddel van een Proof of Concept. 4 op een rij online zou dan zelf dit Proof of Concept uitwerken. Zij zullen hier een website voor creëren met een mooie grafische interface.

De Proof of Concept moest demonsteren dat het spel door één speler te spelen is tegen de computer, of door twee spelers tegen elkaar. Het moest duidelijk zijn wie welk steentje heeft. Spelers moesten hun spelersnaam kunnen opslaan zodat ze deze later weer konden kiezen. Hierdoor zouden er ook resultaten bijgehouden kunnen worden zodat spelers kunnen zien hoe vaak zijn gespeeld hebben en hoe vaak zij gewonnen, verloren of gelijk gespeeld hebben.

Verder moet het spel de originele regels volgen zoals stoppen bij 4 op een rij of gelijkspel en moet het steentje altijd in het onderste vrije vakje van de gekozen rij vallen zodat deze niet ergens blijft zweven.

1.3. Opdrachtomschrijving.

Voor de casus moet een spel (4 op een rij) ontworpen worden. Deze casusgroep zal dit in C# programmeren. In het spel moet de mogelijkheid bestaan om tegen de computer te spelen, of om tegen een andere persoon te spelen via hetzelfde interface. Om de optie “tegen te computer spelen” te laten werken, zal er een algoritme worden ontworpen en ontwikkeld, waardoor de computer zelf strategisch goede keuzes kan maken.

Daarnaast is het de bedoeling dat aan het programma ook een database gekoppeld wordt. Deze zal dienen voor de opslag van spelgegevens (resultaten), en de gegevens van spelers.

Om dit te realiseren wordt er een onderzoek uitgevoerd om informatie te verzamelen over bestaande oplossingen, over hoe spel algoritmes werken en over hoe het spel werkt. Deze kennis is noodzakelijk om de requirements op een rij te krijgen en te begrijpen waar het spel en het algoritme aan moet voldoen. Vervolgens wordt het programma ontwikkelt aan de hand van de gevonden literatuur. De voortgang wordt geground door de code te vergelijken met bestaande oplossingen.
Aan het eind wordt de code getest met een testplan om vast te stellen hoe complex en werkend het programma is.

2. Research en informatie verzameling.

2.1. Gekozen onderzoeksmethodes en onderbouwing.

Er kunnen veel requirements gebaseerd worden op het originele spel. Daarom zullen de spelregels als basis dienen van het onderzoek. Aan de hand van de spelregels kan bepaald worden wat de code moet kunnen doen en welke beslissingen moeten worden gemaakt. Dit kan de scope van de code bepalen en de eerste requirements maken van de functies van het spel. Daarnaast is het onderzoek nodig om bestaande oplossingen en ideeën over een spel algoritme te vinden.

Om de juiste informatie te vinden, wordt de Kitchenham methode toegepast op de zoekresultaten. Aan de hand van een lijst van zoektermen, wordt een lijst met bronnen zoals artikelen, filmpjes en literatuur verzameld. Op gezond verstand worden er een aantal bronnen naar de exclusie lijst gestuurd. Dit zijn bronnen die, ondanks dat ze naar voren kwamen met onze zoektermen, duidelijk niet van toepassing zijn op het project.
Met een verkorte inclusie lijst, wordt de inhoud, titel en samenvatting bekeken en kunnen deze punten scoren. Aan de hand van deze punten, krijgt de bron een rank. Er wordt per gevonden bron gelijk geanalyseerd, omdat dat efficiënter is dan eerst een hele lijst te maken en die dan weer uit te moeten pluizen. Ook wordt relevante informatie meteen gefilterd en apart genoemd, om zo te voorkomen dat dit later misschien gemist wordt.

Veel van deze bronnen zijn grijze literatuur. De informatie is dan aan de hand van andere literatuur gevalideerd. Aan de hand van de grijze literatuur wordt gekeken hoe dit spel in het verleden is gerealiseerd. Het is niet nodig dat deze literatuur reviewed is, omdat het voornamelijk ingezet wordt ter inspiratie van de structuur van de code. De validatie zal uit de tests komen die tijdens dit project plaatsvinden. Als, met behulp van de gevonden bronnen, het project goed uit te test komt, dan valideert dit de waarde van de bronnen. Met een groter project kan dit gevaarlijk zijn omdat er dan laat in het project naar nieuwe oplossingen gezocht moet worden. Echter, is de volwassenheid van de oplossingen zodanig hoog dat het risico dat dit gebeurd klein is.
Aan de hand van het DSR matrix principe, kan vastgesteld worden dat de volwassenheid hoog ligt omdat er veel zoekresultaten zijn met oplossingen voor vier op een rij, zoals filmpjes, artikelen en thesis uitwerkingen van het algoritme principes.

Er vind ook een interview plaats met de opdrachtgever, zodat de wensen van de opdrachtgever in kaart kan worden gebracht. Dit was zeker nodig voor het database, om te begrijpen welke meerwaarde de database moet hebben voor het functioneren van het programma. De opdrachtgever kan aangeven welke eisen er verder gesteld worden aan het spel, buiten de normale spelregels.

Bij het opstellen van de requirements worden de requirements geprioriteerd aan de hand van de 100 punten methode. Deze methode is in dit project wenselijk omdat het aantal requirements relatief laag is, en van nature in categorieën vallen. Deze categorieën corresponderen met de twee hoofdonderwerpen van het onderzoek: het algoritme en de spelregels.

Ter ondersteuning van de requirements wordt er een Use-case diagram ontworpen die de functies verdeelt tussen speler en computer. Dit draagt bij aan het overzicht van wat het algoritme moet kunnen, en wat aan de speler overgelaten kan worden.
Voor het ontwerpen van de code begint het ontwerp met een klassediagram. Samen met een ERD wordt zo de structuur van de code en de connectie met de database bepaald. De klassediagram kan een aantal keer veranderen tijdens het proces. Het kan gebeuren dat tijdens het programmeren de conclusie wordt getrokken dat een andere structuur beter is voor het functioneren van de functies en het algoritme.

2.2. Resultaten.

Spelregels.

Iedere speler heeft 21 stenen. Deze stenen heb meestal de kleuren rood of geel. Het doel is om als eerste 4 stenen op een rij te krijgen. Dit kan diagonaal, verticaal of horizontaal. Het spel eindigt als alle vakjes vol zijn en er geen vier op een rij zijn gemaakt. Het bord wordt dan weer leeggehaald. Degene met de gele (lichtere) stenen begint altijd als eerst. Een boord heeft 6 rijen en 7 kolommen, dus in totaal 42 vakjes. Je mag niet twee keer achter elkaar een steen leggen.

Bron: https://123bordspellen.com/hoe-speel-je-4-op-een-rij/

“np.zeros( ) function is used to create a matrix full of zeroes. This function in Python can be used when you initialize the weights during the first iteration in TensorFlow and other statistic tasks. ((6,7)) are the dimensions. 6 rows and 7 columns. Then we just return that board.

We will begin writing the main game loop now. We’re going to create a loop as while not game_over. A while not loop repeatedly executes the body of the loop until the condition for loop termination is met. Our loop is going to run as long as this game_over variable is false. We will initialize the game_over as False. The only time its going to switch to true is if someone gets 4 circles in a row.”

We will begin writing the main game loop now. We’re going to create a loop as while not game_over. A while not loop repeatedly executes the body of the loop until the condition for loop termination is met. Our loop is going to run as long as this game_over variable is false. We will initialize the game_over as False. The only time its going to switch to true is if someone gets 4 circles in a row.

To increase the turn by 1 we will use turn += 1. To make it switch between player 1 and 2 alternatively, we use turn = turn % 2.”

Het artikel van askpython laat een methode van programmeren zien waarbij het spel wordt doorgezet met een Boolean en een while loop. De Boolean is game_over = false, en de code geeft aan dat het spel

door moet gaan zolang de Boolean op false staat. Binnen deze while loop, kan het spel plaatsvinden.

De beurten kunnen worden bepaald aan de hand van een integer. Als de integer op 0 staat, is speler een aan de beurt, als de integer op 1 staat, is de andere speler aan de beurt, en dan wordt de integer teruggezet naar nul.

De speler kan kiezen tussen de 7 kolommen, waardoor zijn steentje kan worden toegevoegd aan een array, en standard naar de eerst volgende beschikbare index.
Er moet voorkomen worden dat indexen worden overschreven; het programma zou dan een foutmelding moeten geven dat de array vol zit.
Er moet een gelijkspel melding komen als alle array's vol zijn.
bron: https://www.askpython.com/python/examples/connect-four-game

```python
import numpy as np

def create_board():
    board = np.zeros((6,7))
    return board

#initialize board
board = create_board()
#We will initialize the game_over as False.
game_over = False
turn = 0

while not game_over:
    #Ask for player 1 input
    if turn == 0:
        selection = int(input("Player 1, Make your Selection(0-6):"))

    #Ask for player 2 input
    else:
        selection = int(input("Player 2, Make your Selection(0-6):"))

    turn += 1
    turn = turn % 2
```

```python
def create_board():
    board = np.zeros((6,7))
    return board

def drop_piece(board,row,col,piece):
    board[row][col]= piece

def is_valid_location(board,col):
    #if this condition is true we will let the use drop piece here.
    #if not true that means the col is not vacant
    return board[5][col]==0

def get_next_open_row(board,col):
    for r in range(ROW_COUNT):
        if board[r][col]==0:
            return r

def print_board(board):
    print(np.flip(board,0))
```

Aan de hand van een dictionary kan het systeem nakijken of een van de conditions voor een winnend spel van toepassing is. Elke keer als er een speler is geweest, dan kijkt het systeem de dictionary na.

Winnende strategieën.

" Place in the middle column

If the player can play first, it is better to place it in the middle column. Since the board has seven columns, placing the discs in the middle allows connection to go up vertically, diagonally, and horizontally. In total, there are five possible ways.

· Make traps for the opponent

One typical way of not losing is to try to block the opponent's paths toward winning. For example, preventing the opponent from getting a connection of three by placing the disc next to the line in advance to block it. This strategy also prevents the opponent from setting a trap on the player.

· Make a "7."

A 7 trap is a name for a strategic move where one positions his disks in a configuration that resembles a 7. With three horizontal disks connected to two diagonal disks branching off from the rightmost horizontal disk. The 7 can be configured in any way, including right way, backward, upside down, or even upside down and backward. This disk formation is a good strategy because it gives players multiple directions to make a connect-four."

Het algoritme.

Behalve het minimax algoritme, verteld dit artikel ook over wat winnende strategieën zijn in het spel, of je nou tegen een speler of een computer speelt. Het artikel spreekt over de "7" techniek, en over de winkansen als je het eerste steentje in het midden plaatst. Vervolgens verteld dit artikel hoe dit toepasbaar is voor een minimax algoritme.
Bron:https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f

"The topic of this thesis is Monte-Carlo Tree Search (MCTS), a technique for making decisions in a given problem or domain by constructing an internal representation of possible actions, their effects, and the possible next actions that result. This representation takes the form of a tree, and it is grown and improved over time so as to find the best action to take. During the construction of the tree, MCTS focuses on the actions that currently seem most promising, where promising actions are identified through the so-called Monte-Carlo simulations. The basic idea behind these simulations is estimating the quality of an action by repeatedly sampling its possible consequences, the possible futures resulting from it."

https://project.dke.maastrichtuniversity.nl/games/files/phd/Baier_thesis.pdf

2.3 Conclusie.

Spelregels.
Iedere speler heeft 21 stenen. Deze stenen heb meestal de kleuren rood of geel. Het doel is om als eerste 4 stenen op een rij te krijgen. Dit kan diagonaal, verticaal of horizontaal. Het spel eindigt als alle vakjes vol zijn en er geen vier op een rij zijn gemaakt. Het bord wordt dan weer leeggehaald. Degene met de gele (lichtere) stenen begint altijd als eerst. Een boord heeft 6 rijen en 7 kolommen, dus in totaal 42 vakjes. Je mag niet twee keer achter elkaar een steen leggen.

Het Hasbro bord, zoals meest bekend, gebruikt 6 en 7 voor kolommen en rijen, terwijl meeste andere bronnen het hebben over 8 en 8. Wel hebben ze het beide over 21 steentjes per persoon. Het verschil is dat bij het grotere bord, het bord dan nooit vol zou kunnen raken. Het is dan voor een algoritme makkelijker om het kleine bord te gebruiken.

Het meest gebruikte algoritme voor bordspellen is het zogenaamde minimax algoritme. Dit idee is gebaseerd op de monte carlo tree search algoritme, waarbij het principe is dat er scores hangen aan de keuzes die het algoritme kan maken, en dat de computer de pad met keuzes afgaat met de hoogste score. In het minimax algoritme, is het idee dat de computer erop doelt zijn eigen score te verhogen.

3. Requirements en prioritering.

Code	Eis	Prioritering 100 punten methode	FE (functioneel) of N-FE (niet functioneel)	Bron	Eigenaar
Spel-0001	Het spel kan door twee spelers gespeeld worden.	9	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Spel-0002	het spel kan door één speler gespeeld worden tegen de computer.	9	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Speler-0001	De spelers gebruiken verschillende stenen.	8	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0002	Het systeem stop het spel als er 4 op een rij is gedetecteerd door steentjes van een speler.	8	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0006	Het systeem moet de mogelijkheid geven om een nieuwe speler aan te maken.	7	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0007	Het systeem moet de mogelijkheid geven om een bestaande speler te kiezen.	7	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Spel-0003	Een speler kan maximaal 21 zetten maken.	4	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Spel-0004	Een speler kan maximaal 1 steen plaatsen per zet.	4	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0003	Het systeem controleert op vier op een rij verticaal.	4	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0004	Het systeem controleert op vier op een rij diagonaal.	4	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0005	Het systeem controleert op vier op een rij horizontaal.	4	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Speler-0002	Speler 1 begint als eerste.	3	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Bord-0001	Het spel moet 42 vakjes hebben.	3	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders

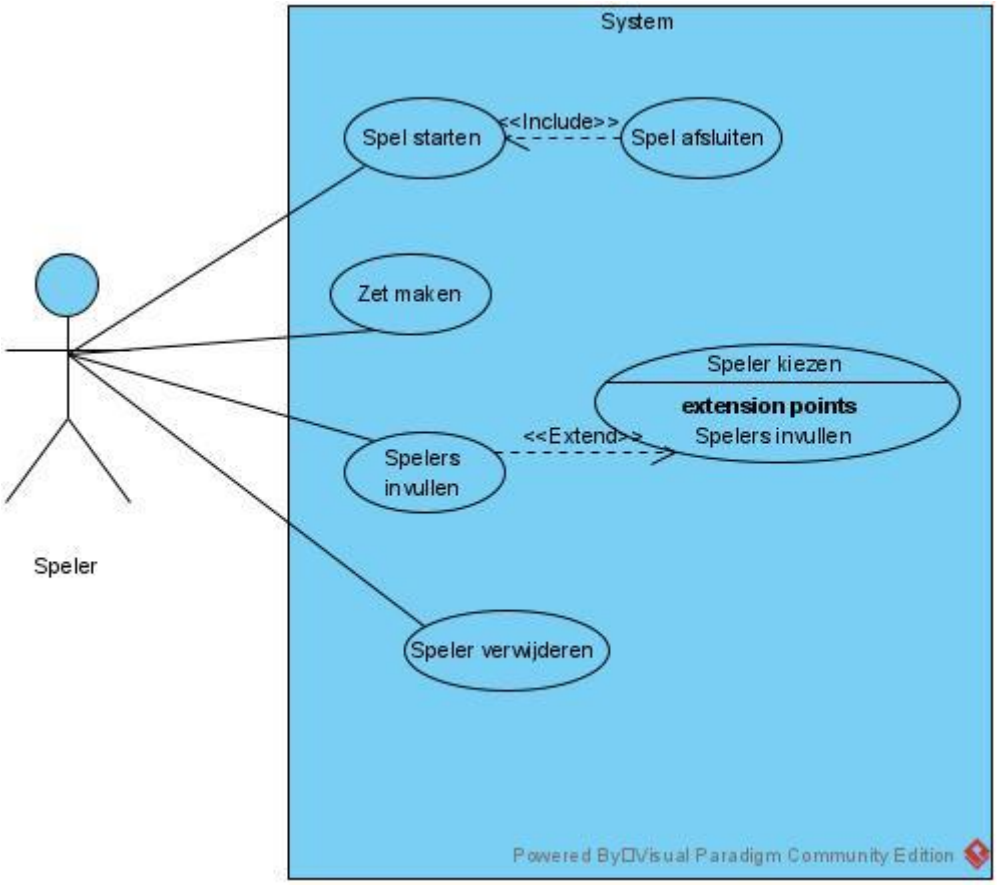
Code	Eis	Prioritering 100 punten methode	FE (functioneel) of N-FE (niet functioneel)	Bron	Eigenaar
Speler-0003	De speler mag geen vakje kiezen waar al een steen in zit.	3	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Speler-0004	De speler moet altijd het onderste vakje van een rij kiezen die leeg is.	3	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0001	Het systeem stop het spel als alle 42 vakjes gevuld zijn.	2	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0008	Het systeem moet de scores per speler opslaan.	2	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0009	Het systeem moet de 10 hoogste scores kunnen aantonen.	2	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0010	Het systeem moet per speler de score kunnen laten zien.	2	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Computer-0001	De computer kiest alleen lege vakjes om steentjes te plaatsen.	2	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Computer-0002	De computer moet een speler blokkeren met een steentje als de speler bijna wint.	2	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0014	Het systeem moet een melding weergeven als er gewonnen is door een speler.	2	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0015	Het systeem moet een melding weergeven met "verloren" als de computer van een speler wint.	2	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0016	Het systeem moet een melding geven wanneer er een nieuwe speler wordt gemaakt maar deze naam al in gebruik is.	1	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0011	Het systeem moet nadat het spel is gestopt de keuze geven om opnieuw te spelen.	1	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders
Systeem-0012	Het systeem geeft een melding als een speler een vakje kiest die al vol is.	1	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders

Code	Eis	Prioritering 100 punten methode	FE (functioneel) of N-FE (niet functioneel)	Bron	Eigenaar
Systeem-0013	Het systeem moet een melding weergeven bij gelijkspel.	1	FE	Interview 06-10-2021 met Hans Snijders.	Hans Snijders

Onderbouwing en argumentatie eisen.

Vooraan de prioriteit staan de eisen die het spelen van het spel mogelijk maken. Dit slaat op functies en eisen die de regels van het spel bewaren, en de spelers in staat stellen om een spel te starten. Dit slaat minder op eisen aan het database, of eisen die de doorloop van het programma bepalen. Als er een werkend bord is en een werkend algoritme, dan kan de speler het spel spelen, ook als het spel aan het eind spontaan afsluit.
Requirements aan de computer geven een idee van wat er van het algoritme verwacht wordt, en waar de prioriteit ligt bij het schrijven van de code.
De laagste requirements zijn overige functionaliteiten die te maken hebben met de userflow en user design van het programma. Dit gedeelte van het design wordt verder niet uitgewerkt in het ontwerp omdat er geen harde eisen aan hangen die hier noodzaak aangeven. De requirements impliceren alleen het bestaan van de functies.

4. Use-case

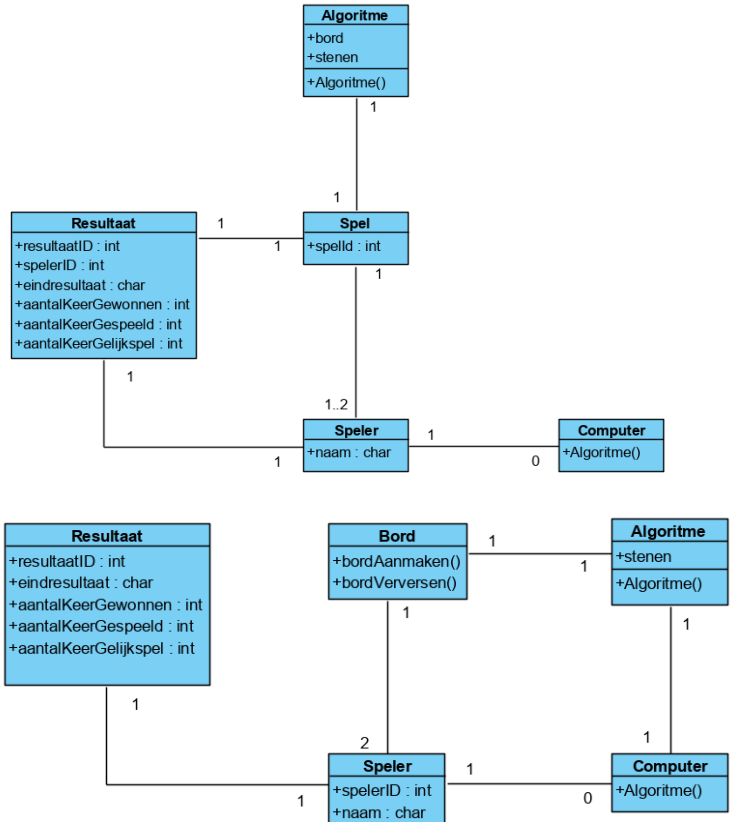


In dit diagram staan de hoogste geprioriteerde eisen aan het spel weergegeven, om zo te bepalen wat de speler moet kunnen om het spel te kunnen spelen. Bij het invullen van de spelers, kan de speler aangeven of speler twee een computer wordt. Als dit aangegeven wordt, dan wordt als speler automatisch de naam “Computer” ingevuld, wat het programma prompt om het algoritme aan te roepen. Ter extensie kunnen spelers, mits deze opgeslagen zijn, teruggevonden worden met hun score. Door op een rij te hebben wat de speler moet kunnen, wordt zo ook duidelijk welke functies er aan het programma en het algoritme overgelaten moet worden. Zo wordt de scope van wat er in het algoritme moet al iets kleiner.

5. Onderbouwing (grounding)

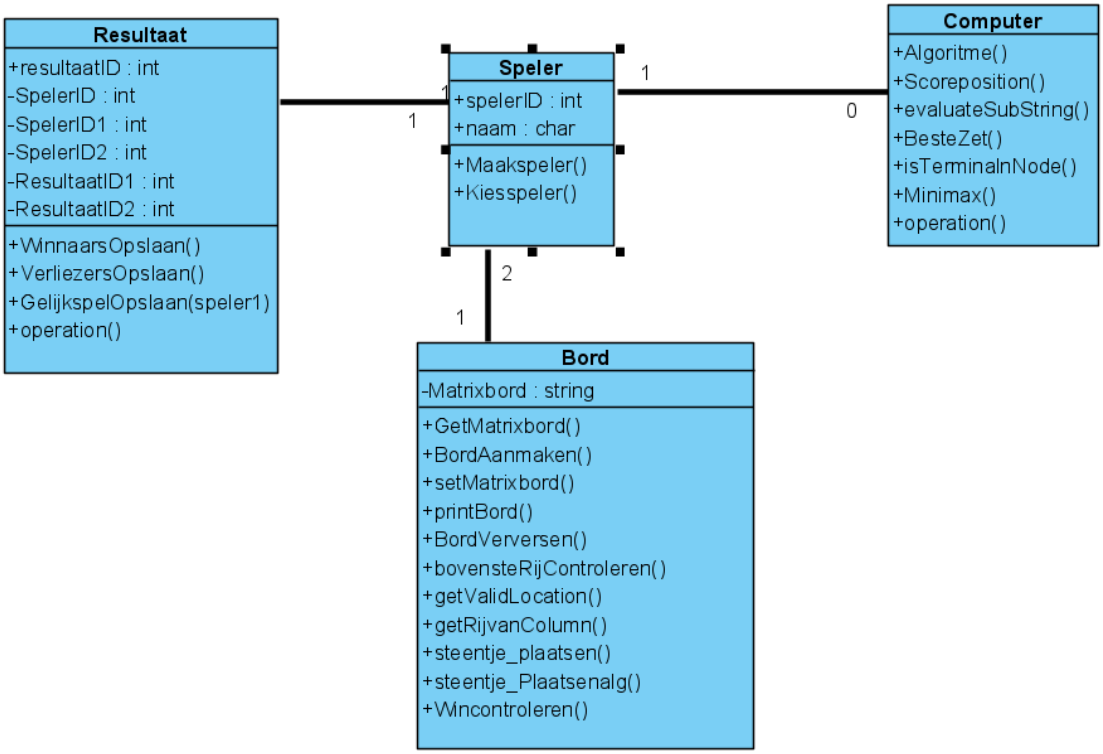
Op de eerste instantie was er bedacht om een dictionary te maken waarin alle win situaties opgeslagen stonden. De computer zou dan, na elke ronde deze dictionary afgaan om te kijken of er al een winsituatie is. Na verder onderzoek, door de documentatie te lezen van anderen die het spel ontworpen hebben, bleek dat er veel efficiëntere manieren zijn. Het algoritme kan alle nodes afgaan van de matrix, en controleren of er ergens vier op een rij staan.

Er was veel twijfel over de klasse structuur. De voornaamste overweging was welke klasse verantwoordelijk is voor het algoritme, en of speler en “computer” als speler, onder dezelfde klasse zouden moeten vallen. Uiteindelijk is ervoor gekozen om deze twee apart te programmeren. Als de code de spelernaam als “computer” invult, wordt er een object van de klas computer aangeroepen, en worden zo de algoritmes ingezet.



6. Design

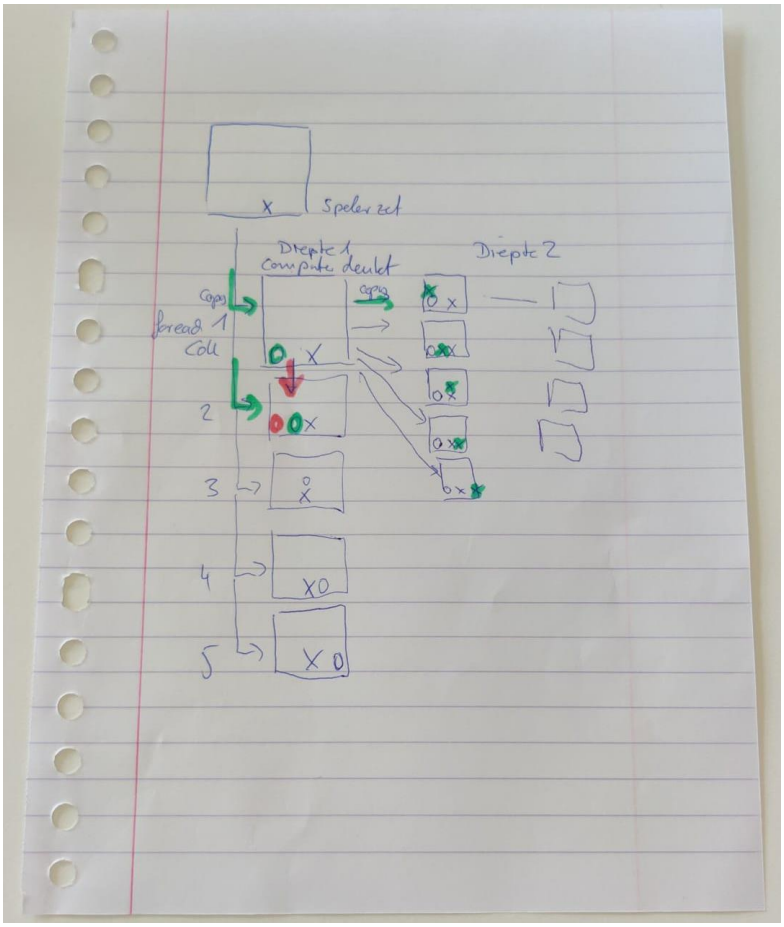
Class diagram:



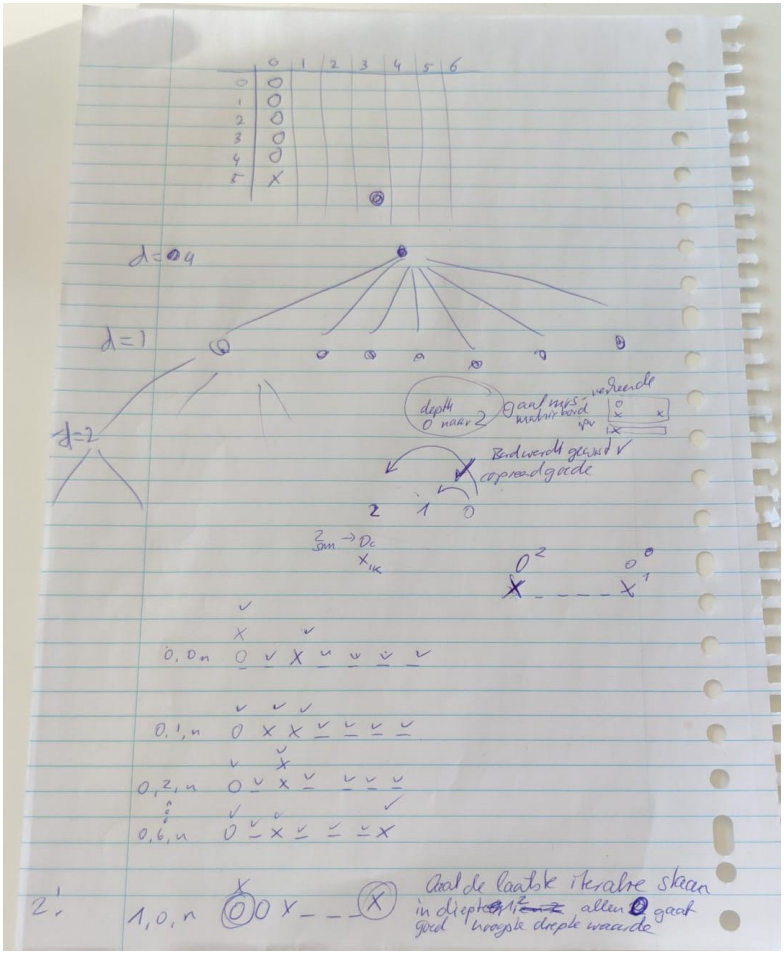
Het programma “spel” dient als het spel waar alle objecten en functies worden opgeroepen. De computer wordt aangeroepen op het moment dat de speler als computer wordt genoemd. De klasse resultaten corresponderen met de database, om het zo de resultaten van de spelletjes makkelijk op te slaan. In de klasse speler worden nieuwe spelers aangemaakt en ongeroepen. Deze klasse bevat ook functies om spelers op te slaan in de database. De klasse computer bevat vervolgens alle functies om beslissingen te maken.

Voor de database is een ERD ontworpen om een idee te krijgen wat de meest efficiënte manier is om de gegevens op te slaan en welke meerwaarde dat heeft voor het functioneren van het spel. De database wordt simpel gehouden met twee tabellen. Een tabel houdt alle spelers bij, en het andere slaat de resultaten op aan de hand van de “spelersID”. Er staat een totaal spellen, gelijkspellen en gewonnen spellen. Er staan geen verloren spellen, omdat dit herleidbaar is. Ook is voor een eventuele score bord het aantal verloren spellen niet interessant.

Algoritme ontwerp.



De diepte setting bepaald hoe diep het algoritme gaat nadenken over welke stappen er gezet moeten worden. Na het vaststellen van een eerstvolgende zet, gaat hij binnen die tree verder zoeken naar de volgende beste stap. Zo probeert hij een aantal stappen voorruit te denken, afhankelijk van hoe diep hij is ingesteld. Tijdens het zoeken naar de beste stap, zal hij, binnen zijn eigen processen, voor de tegenstander een steentje zetten om daar de mogelijkheden van te bekijken. Als hij alle dieptes af is gegaan, slaat hij daar het resultaat van op. Vervolgens gaat hij een andere diepte af, om daar de totaal score van te bereken. Deze worden vergeleken waarvan de beste gekozen wordt. Dit gaat allemaal op een kopie van het spelbord. Zo gaat het algoritme door totdat die op alle dieptes heeft gekeken welke zet het beste is. dit geeft die door en zal hij in het echte bord plaatsen.



Bovenaan de pagina is het bord getekend met de juiste indexen. Dit is gedaan om makkelijk te kijken of het controleren en het tellen juist gedaan wordt. Ook is nog een keer de boom getekend om de doorloop weer te geven. Er is handmatig gemaakt wat de computer zou moeten doen, om daarna te controleren of de computer ook dat gene doet. Klopt het niet, dan moet dit aangepast worden, klopt dit wel dan is dat onderdeel juist.

Een belangrijk optimalisatie punt is dat als hij ziet dat een van de opties een lage score geeft, dat hij andere leaves binnen die branch ook niet meer gaat bekijken.

Bronnen en referenties.

https://123bordspellen.com/hoe-speel-je-4-op-een-rij/

https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f

https://www.askpython.com/python/examples/connect-four-game