Pmst1T

Realisatie Document

Never Board

Contents

[Software 2](#_Toc517160472)

[Transities 2](#_Toc517160473)

[UI Componenten 2](#_Toc517160474)

[Schaakspel 3](#_Toc517160475)

[Media 4](#_Toc517160476)

[FUN 5](#_Toc517160477)

# Technology

De onderstaande code laat zien hoe de arduino de stappen motoren aanstuurt.

#include <Servo.h>

#define 1 // Y positie

#define 2 // Y positie

#define 3 // Y positie

#define 4 // Y positie

#define 5 // Y positie

#define 6 // Y positie

#define 7 // Y positie

#define 8 // Y positie

#define A // X positie

#define B // X positie

#define C // X positie

#define D // X positie

#define E // X positie

#define F // X positie

#define G // X positie

#define H // X positie

#define Down 180 // Z Positie

#define Up 0 // Z Positie

Servo X;

Servo Y;

Servo Z;

void setup() {

X.attach(PIN);

Y.attach(PIN);

Z.attach(PIN);

Serial.begin(115200);

}

int aantal=0;

char data[50];

void loop() {

// Voorbeeld commando parameter vanuit Visual Studio:

// Z-18-AH-Offset||18-AH-Offset

// Als Z wordt doorgegeven, betekent dat dat de arm het schaakstuk eerst moet pakken, bij sequentiele commandos is dit dus niet nodig

// 18 staat voor een cijfer tussen 1 en 8, welke gedefined zijn

// AH staat voor een letter tussen A en H, welke gedefined zijn

// Het eerste deel betreft punt 1, het tweede deel punt 2

// Wanneer 1 opdracht is uitgevoerd stuurd de Arduino een bericht terug naar Visual Studio, waarna Visual Studio een nieuwe commando stuurt

while(Serial.available())

{

data[aantal]=Serial.read();

if(data[aantal]=='%')

{

aantal=-1;

for(int y=0;y<50;y++)

{

data[y]=' ';

}

}

aantal++;

for(int u=0;u<50;u++)

{

Serial1.print(data[u]);

}

Serial1.println("Woord");

}

void Beweging(Z,18a,AHa,OffXa,OffXb,18b,AHb,OffXa,OffXb)

{

if(Za=!Z) //Als Z niet doorgegeven is

{

Z.write(Down);

delay(tijd);

X.write(AHa);

Y.write(18a);

delay(anderetijd);

X.write(AHa+OffXa);

Y.write(18a+OffYa);

Z.write(Up);

}

X.write(AHb+OffYb);

delay(tijd);

y.write(18b+OffXb);

delay(tijd);

}

# Software

### Transities

Transities op het never board werken met een gekleurde “Fade” animatie. De werking achter deze animatie is erg simpel. Er blijft namelijk op de achtergrond constant een form open. Als er een transitie plaats vindt dan Fade het voorste form weg om plaats te maken voor het dan ingekleurde voorste Form. Als het voorste form compleet weg gevaagd is dan opent het achtergrond form opnieuw een ander form voor op de voorgrond. De kleur die gebruikt wordt tijdens de transitie wordt direct uit de kleur voor de knop gehaald door 5 pixels vanaf de middel bovenste pixels te kijken welke kleur er gebruikt wordt. Als het Form in transitie is wacht het eerst totdat alle knoppen geladen zijn voordat het de transitie verlaat.

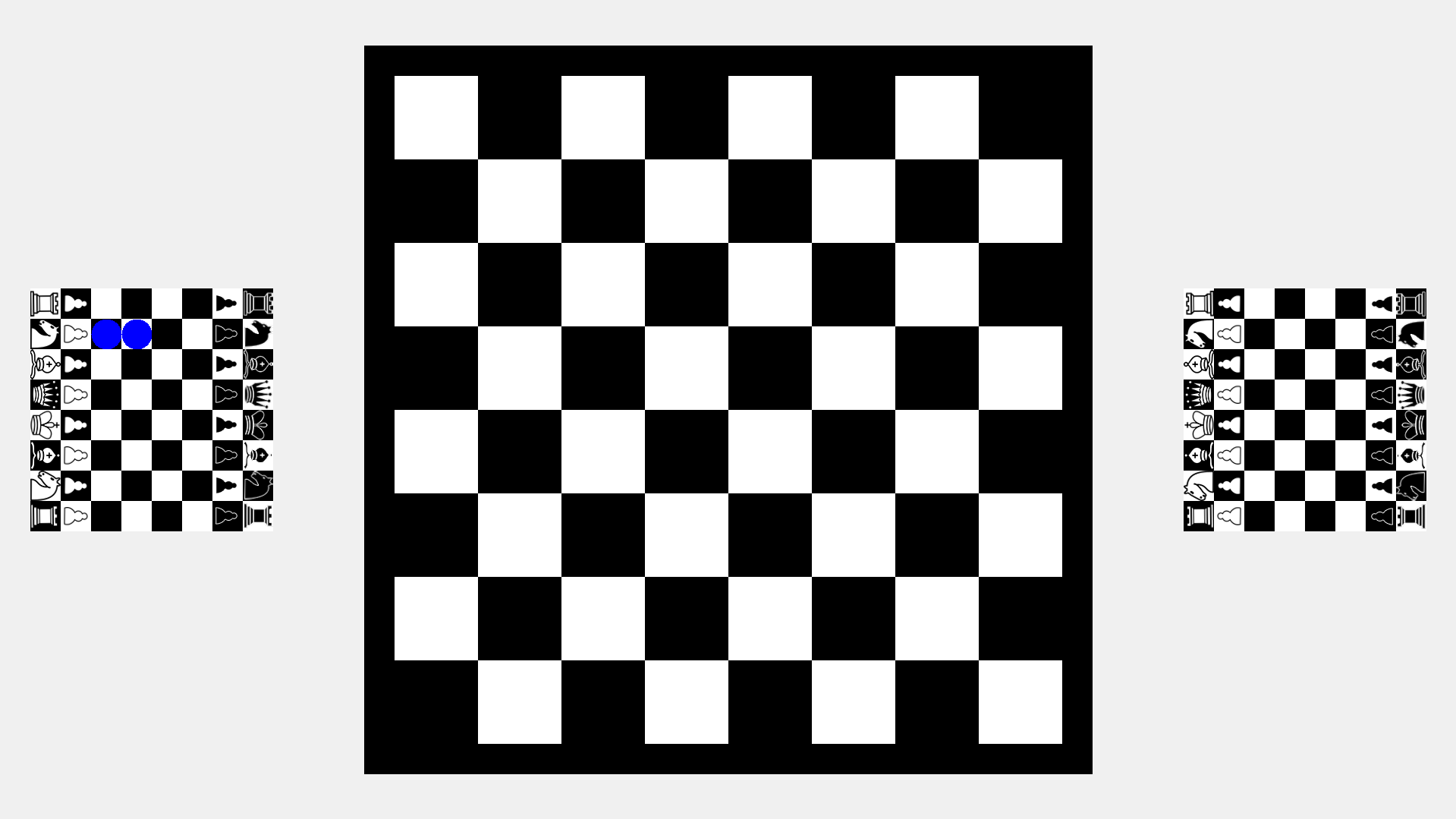
### UI Componenten

De knoppen die gebruikt worden in het Never-Board zijn zo gedesigned dat elke persoon rondom de tafel kan lezen wat er staat. Voor een dynamisch effect zijn de knoppen zo geprogrammeerd dat het icoon in het midden tegen de klok in draait en de letters met de klok mee. Omdat het te zwaar is voor het programma om dit 50 keer per seconden te doen wordt er een kwart van de afbeeldingen ge-preload. Dit houdt in dat er 90 afbeeldingen voor elke graad ge load worden. De rest van de afbeeldingen wordt dan geflipt met de rotateflip method. De afbeeldingen worden vervolgens in een picturebox geladen. Als er op de picturebox geklikt wordt wordt er via de stelling van Pythagoras berekend of er ook op de knop wordt gedrukt in plaats van ernaast. Een knop kan aan een form toegevoegd worden door in een list een knop object toe te voegen. Het click event en de picturebox wordt dan vervolgens dynamisch toegevoegd.



### Schaakspel

Het Schaakbord wordt compleet automatisch gegenereerd met draw functies. Vervolgens worden er speelstukken op de twee controlepanelen getekend. Deze kunnen door de speler geselecteerd worden waardoor alle mogelijke zetten getoond worden. Met een tweede klik op het bord kan de speler het speelstuk bewegen. Dit zorgt ervoor dat er een signaal naar de arduino wordt gestuurd met instructies over het bewegen van speelstukken. Ook wordt er een animatie afgespeeld op het bord dat de mogelijke zetten laat zien. De mogelijke zetten worden per stuk via algoritmiek berekend.



# Media

Het realiseren van ons product was met veel trial en error, niemand had met dit formaat gewerkt en het onder de knie ter krijgen duurde wel even. Nadat we de keuzen hadden gemaakt om de knoppen te laten ronddraaien ging de rest van lay-out best gemakkelijk.

Met de animaties hebben we lang moeten experimenteren, lex en ik hadden bijna weinig ervaring met after effects en om tot een goed resultaat te komen hebben we veel onderzoek moeten doen naar de beste manier op het toe te kunnen passen in ons product. Er zijn veel veranderingen door gebracht aan de stijl van animeren, dit is voornamelijk gedaan voor de halbaarheid en mogelijkheden van het product. Ook een reden was dat we er achter kwamen dat in het video format GIF de transparantie is gelimiteerd tot: volledig transparant of volledig zichtbaar. Dit heeft er voor gezorgd dat de stijl van de animaties drastisch is veranderd. De stijl is van oorlog naar cartoon/ arcade gegaan, de geluiden passen hier gelukkig nog wel bij.

# FUN

We hebben de database gemaakt volgens het database diagram dat we hebben gemaakt in de ontwerpfase. Alle tables zijn gemaakt en de we hebben ook de juiste relaties tussen de verschillende tables. Om de applicatie goed te kunnen runnen zullen we ook wat met de database moeten gaan doen. Hier hebben we dus ook query’s voor nodig om met de database te kunnen communiceren. Dit zijn onder andere de query’s die we gaan gebruiken.

Om de ID’s van elke groep op te vragen:

SELECT ID

FROM Groep;

Deze is om de gegevens op te vragen van een bepaalde groep:

SELECT Naam, Kleur

FROM Groep

WHERE ID = [int GroepID];

Om de ID’s van elke gebruiker op te vragen:

SELECT ID

FROM Gebruiker;

Deze querie vraagt de gegevens op van een bepaalde gebruiker:

SELECT Naam, Kleur, Foto

FROM Gebruiker

WHERE ID = [int GebruikerID]

Dit is om alle spellen te sorteren op genre:

SELECT \*

FROM Spel

WHERE genre = “[String genrenaam]”;

Dit is de query om alle gebruikers op te vragen die in een bepaalde groep zitten:

SELECT \*

FROM Groep

INNER JOIN Gebruiker ON Gebruiker\_Groep.GebruikerID = Gebruiker.GebruikerID

INNER JOIN Groep ON Gebruiker\_Groep.GroepID = Groep.[GroepID]

WHERE Gebruiker.GebruikerID = Groep.[GroepID];

Dit is de query die alle gebruikers selecteert die niet in een bepaalde groep zitten:

SELECT \*

FROM Groep

INNER JOIN Gebruiker ON Gebruiker\_Groep.GebruikerID = Gebruiker.GebruikerID

INNER JOIN Groep ON Gebruiker\_Groep.GroepID = Groep.[GroepID]

WHERE Gebruiker.GebruikerID NOT Groep.[GroepID];

Voor het fatsoenlijke gebruik maken van de database maken we gebruik van een nieuwe class die alle functies voor de database heeft. In deze class staan alle query’s die gemaakt zijn. Hierdoor kunnen we op een overzichtelijke manier alle functies gebruiken zonder overzicht te verliezen. Verder hebben we ook een nieuwe class gemaakt voor alle gebruikers en groepen om te gebruiken in de applicatie zelf. Deze zijn nodig voor het maken van objecten zodat we die kunnen gebruiken in listen.