Race Simulatie met Artificiële intelligentie

Brys Tim - Seberechts Ruud

Academiejaar 2007 - 2008

Toegepaste Informatica - Erasmushogeschool Brussel
Promotor: Dhr. P. Van Laethem

Synopsis

Dit verslag behandelt de ontwikkeling van een software simulatie voor autoracen. De nadruk ligt op de onconventionele artificiële intelligentie, specifiek ontworpen om correct om te gaan met eender welk circuit of racessituatie. De technologieën die gebruikt zijn om dit te bereiken, zijn onder andere A* pathfinding voor het vinden van de ideale lijn, neurale netwerken om de auto te besturen en flocking om obstakels te ontwijken.

Naast de AI is er veel aandacht besteed aan de physics engine, die onmisbaar is in een degelijke simulatie. Verschillende engines zijn uitgetest en technieken voor geoptimaliseerde collision detection zijn ontwikkeld en geïntegreerd.

Het weergeven van beelden en opvangen van invoer wordt verwezenlijkt met de hulp van de Gosu library. Hiervoor werd ook een GUI framework ontwikkeld, wat ervoor zorgt dat de interactie met de gebruiker op een gebruiksvriendelijke wijze wordt afgehandeld.

Een laatste onderdeel is circuit- en beeldgeneratie, met als toepassing het creëren van willekeurige circuits en bijbehorende achtergronden. Via een handige editor wordt ook de gebruiker de mogelijkheid gegeven om zelf circuits te ontwerpen.

Voorwoord

Dit project in verband met race simulaties is geschreven als eindwerk in het derde en laatste jaar toegepaste informatica. De bedoeling van dit eindwerk was, als individu of in een kleine groep, een welbepaald probleem aan te pakken en verder uit te diepen. Het onderwerp moest tegelijk uitdagend en motiverend zijn.

Aan de hand van twee wekelijkse bijeenkomsten, vele uren samen brainstormen en nog meer uren coderen is dit dan het resultaat van een half jaar noeste arbeid.

Het is niet minder dan gepast onze promotor, Dhr. Philippe Van Laethem, te bedanken voor zijn interesse en hulp bij dit eindwerk. Daarnaast bedanken we ook de enthousiastelingen die dit verslag hebben nagelezen. Ten slotte willen we ook elkaar bedanken voor de inzet en passie waarmee aan dit eindwerk is gewerkt.

Inhoudsopgave

1 Inleiding **1**

2 Gebruikte technologieën **2**

2.1 C++ 2

2.2 Gosu Library 2

2.2.1 Window klasse 2

2.2.2 Image klasse 3

2.2.3 Font klasse 3

2.2.4 Color klasse 3

2.2.5 Bitmap klasse 3

2.3 Adobe Flash 4

2.4 C#.NET 4

2.5 SAX en DOM 5

3 Het circuit **6**

3.1 Beschrijving 6

3.2 De RaceSimTrack file 9

3.2.1 DTD 9

3.3 Inlezen 9

3.3.1 XML parsing met Xerces-C 10

3.3.2 Reconstructie level 10

4 Artificiële intelligentie **11**

4.1 AI in een racesimulatie 11

4.2 AI in deze racesimulatie 12

4.3 Pathfinding 12

4.3.1 A* . . . 13
4.3.2 Rasterizatie . . . 18
4.3.3 Eigen implementatie . . . 20
4.4 Besturing . . . 23
4.4.1 Neuraal netwerk . . . 23
4.4.2 Trainingsset . . . 30
4.5 Machine learning . . . 31
4.6 Flocking . . . 33
4.6.1 Toepassing . . . 33

5 Physics **34**
5.1 PhysicsWorld en PhysicsObject klassen . . . 34
5.1.1 Collision Detection . . . 35
5.2 CarPhysics klasse . . . 36
5.3 Andere geprobeerde engines . . . 38
5.3.1 Open Dynamics Engine . . . 38
5.3.2 2D Car Physics by Matt Kincaid . . . 38

6 Grafische gebruikersomgeving **39**
6.1 Werking . . . 39
6.2 GUI Listeners . . . 40
6.3 GUI Componenten . . . 40
6.4 De verschillende GUI layouts . . . 41
6.4.1 Container layout . . . 42
6.4.2 Button layout . . . 42
6.4.3 Menu item layout . . . 43
6.4.4 Tabbed text button layout . . . 43

6.5 De verschillende GUI componenten 43
6.5.1 Image component 43
6.5.2 Background component 44
6.5.3 Button component 44
6.5.4 Label component 44
6.5.5 Listbox item component 45
6.5.6 Listbox component 45
6.5.7 Menu item component 46
6.5.8 Menu component 46
6.5.9 Mouse component 46

7 Globale werking van de applicatie 47

8 Verloop van een race 49
8.1 Algemene werking 49
8.2 Visuele aspecten 49
8.2.1 Basis 49
8.2.2 Tekstuele informatie 50
8.2.3 Pauzemenu 50
8.2.4 Bandensporen 50
8.2.5 Extra Informatie 52
8.3 Achterliggende code 52

9 Achtergrond generatie 54
9.1 Algemene werking 54
9.2 Belangrijke technieken 55
9.2.1 Noise (ruis) 55
9.2.2 Interpolatie 56

9.2.3 Mixen van afbeeldingen 57
9.2.4 Blur filter 58
9.2.5 Anti-aliasing 59
9.3 De verschillende stappen 60
9.3.1 Rasterizatie 60
9.3.2 Generatie van wolken 62
9.3.3 Inkleuring van het circuit 63
9.3.4 Plaatsing en generatie van bomen 64

10 INI Bestand **67**
10.1 Algemene werking 67
10.2 CIniFile klasse 67

11 Programmastructuur **69**

12 TrackEditor **70**
12.1 Algemene werking 70
12.2 Werking Flash gedeelte 71
12.2.1 Het eigenschappen paneel 73
12.2.2 Tekenen van een circuit 73
12.2.3 Wijzigen van een circuit 74
12.2.4 Verplaatsen van een circuit 74
12.2.5 Omdraaien van een circuit 75
12.2.6 Creëren van een willekeurig circuit 75
12.2.7 Opslaan en inladen van een circuit 77
12.3 Integratie in C#.NET applicatie 78
12.3.1 Communicatie tussen Flash en C# 78
12.3.2 Werking 80

13 Gebruikte wiskunde **83**

13.1 Afstand tussen twee punten 83

13.1.1 Afleiding van de formule 83

13.1.2 Praktisch gebruik 84

13.2 Verschuiving punt over een afstand en een hoek 84

13.3 Afgeleide van de sigmoid functie 85

14 Retrospectie **87**

14.1 Artificiële intelligentie 87

14.1.1 Verbeteringen 87

14.1.2 Alternatieve pathfinding 87

15 Besluit **89**

16 Literatuurlijst **90**

1 Inleiding

In dit verslag wordt het verloop en resultaat van dit eindwerk, 'Race Simulatie met Artificiële intelligentie', beschreven. Zoals de naam doet vermoeden is het doel van dit eindwerk software te schrijven die autoraces simuleert.

De keuze voor dit eindwerk heeft twee redenen, namelijk de interesse in artificiële intelligentie en de passie voor autosport. Bij de ontwikkeling zal de meeste aandacht gaan naar het onderzoeken en ontwikkelen van een AI engine[1]. Het is namelijk de bedoeling dat een computergestuurde wagen elk mogelijk circuit op een competitieve wijze kan afleggen. De AI zal geschreven worden met enkele basisregels in het achterhoofd. Zo zal een computergestuurde bestuurder weten dat de wagen zal versnellen wanneer hij gas geeft en zal vertragen wanneer hij remt. De AI kan verder uitgedaagd worden door bijvoorbeeld verschillende weersomstandigheden, en het gebruik van achtvormige circuits.

De physics zijn in dit project van minder belang, al zullen deze natuurlijk wel enig niveau moeten bereiken. Om niet te veel tijd te verliezen aan dit component zal er gezocht worden naar bestaande physics engines.

Ook de grafische voorstelling wordt zo eenvoudig mogelijk gehouden, zonder dat deze daardoor onaantrekkelijk wordt. Er zal gezocht worden naar een engine die op eenvoudige manier kan omgaan met twee dimensionale beelden en invoer van de gebruiker. Voor het gebruik van de applicatie zal er een compact framework geschreven worden, dat het gebruik van menu's en knoppen moet ondersteunen.

Tenslotte zal er nog een eenvoudige circuit editor geschreven worden. Dit is een aparte applicatie en heeft als doel een gebruiksvriendelijke, maar ook volledige oplossing te bieden voor het creëren van circuits. Deze circuits zullen opgeslagen worden als bestanden, en in te lezen zijn door de andere applicatie.

In dit verslag is een uitgebreide beschrijving van alle onderdelen te vinden, alsook enkele extra's die in de loop van het project zijn toegevoegd.

[1] Softwarecomponent dat instaat voor de berekeningen omtrend artificiële intelligentie

2 Gebruikte technologieën

2.1 C++

C++ is een hogere programmeertaal die onder andere imperatief, object georiënteerd en generiek programmeren ondersteunt. Voor het programmeren van de simulatie is gebruik gemaakt van de Microsoft Visual C++ implementatie.

2.2 Gosu Library

Gosu is een gratis 2D game ontwikkeling library voor de programmeertalen Ruby en C++. De library is beschikbaar voor Mac OSX, Windows en Linux. Zoals vermeld in het vorige hoofdstuk werd er gekozen voor Microsoft Visual C++. De library zelf is nog in volle ontwikkeling, wat op sommige vlakken zorgt voor wat beperkingen, maar door gebruiksgemak en functionaliteit is er toch voor gekozen.

Gosu kan gebruikt worden voor het weergeven van beelden en tekst, afspelen van geluid en beheren van invoer van de gebruiker. Het weergeven van 2D beelden en tekst wordt verwezenlijkt via openGL. Dit zorgt bijvoorbeeld voor zeer snelle rotaties, herschaling en kleurmodulatie[2] van afbeeldingen.

Het vervolg van dit hoofdstuk geeft een kort overzicht van de belangrijkste klassen uit de Gosu library.

2.2.1 Window klasse

De Window klasse van Gosu definiëert een zichtbaar venster en linkt dit meteen aan drie belangrijke aspecten van game programming: graphics, gebruikersinvoer en geluid. Wanneer een Gosu applicatie gemaakt dient te worden zal de basis klasse hoogstwaarschijnlijk deze klasse implementeren. Een window definiëert ook een bepaalde framerate. Binnen elk frame zal eerst de *update* methode uitgevoerd worden, en daarna de *draw* methode. In update moet alle niet visuele code uitgevoerd worden. De draw methode daarentegen mag enkel functionaliteit bevatten die rechtstreeks te maken heeft met de visuele weergave.

[2]Kleurmodulatie maakt het mogelijk afbeeldingen van kleur te veranderen. Een afbeelding met een witte auto kan zo bijvoorbeeld in het rood weergegeven worden.

2.2.2 Image klasse

Een object van de Image klasse bevat afbeeldingsinformatie. Deze klasse wordt doorheen het project gebruikt voor het tonen van afbeeldingen. Bij het creëren van een Image is het nodig het graphics object (dit is beschikbaar in de Window klasse) en een bestandsnaam mee te geven. Enkel BMP en PNG bestanden kunnen ingeladen worden. Er zijn verschillende tekenfuncties beschikbaar van een Image. Hiermee kan de afbeelding bijvoorbeeld geroteerd weergegeven worden.

2.2.3 Font klasse

De Font klasse bevat informatie over een bepaald lettertype in een bepaalde grootte. Net als bij een Image is het nodig het graphics object mee te geven aan de constructor. Via de draw functie kan er een bepaalde tekst op een bepaalde plaats op het scherm weergegeven worden.

2.2.4 Color klasse

Een object van de Color klasse bevat kleurinformatie. Een kleur is 32 bits groot en bevat vier componenten, namelijk rood, groen, blauw en alpha. De eerste drie componenten bepalen de eigenlijke kleur. De alpha waarde bepaalt de transparantie. Elke component krijgt een 8 bits waarde, met andere woorden een waarde van 0 tot 255 ($2^8 = 256$). Hoe groter de waarde van een bepaalde component, hoe meer hij aanwezig zal zijn in het resultaat. Wanneer bijvoorbeeld de rode component de waarde 255 heeft, en de blauwe en de groene slechts 0, zal de uiteindelijke kleur rood zijn. Hoe hoger de alpha waarde, hoe minder transparant de kleur is.

2.2.5 Bitmap klasse

De Bitmap klasse bevat een twee dimensionaal raster gevuld met Color objecten. In tegenstelling tot een Image kan hier elke pixel (plaats in het ratser) van kleur veranderd worden. Wanneer een Bitmap moet worden weergegeven, is het helaas nodig eerst de omzetting te maken naar een Image object. Dit proces neemt voor grote afbeeldingen veel tijd in beslag. Het werken met Bitmaps die moeten weergegeven worden is dus enkel mogelijk met kleine afbeeldingen, of in projecten waar snelheid geen rol speelt.

2.3 Adobe Flash

Adobe Flash is een applicatie voor het ontwikkelen van (hoofdzakelijk) animaties en webapplicaties. Een flash filmpje bestaat uit verschillende frames. Deze frames kunnen, door ze achter elkaar af te spelen, een animatie vormen. Per frame kan er echter ook bepaalde code worden uitgevoerd. Deze code maakt het dan bijvoorbeeld mogelijk om invoer van de gebruiker op te vangen en objecten op het scherm te laten bewegen.

De taal die gebruikt wordt voor het schrijven van code in Flash heet ActionScript. Momenteel zit deze scripttaal aan haar derde editie, voor dit project is er echter gebruik gemaakt van ActionScript 2.0.

Zoals gezegd bevat een flashfilmpje dus verschillende frames. Het is echter mogelijk om op een bepaald frame te stoppen en hier verschillende animaties op uit te voeren. Een belangrijk object hierbij is de *MovieClip*. Een movieclip is een soort van container die verschillende frames en code kan bevatten. Technisch gezien is het flash filmpje dus zelf een movieclip. Via Actionscript is het mogelijk om movieclips aan te spreken, en bijvoorbeeld te verplaatsen.

2.4 C#.NET

C#, uitgesproken in het Engels als "C sharp", is een hogere, object-georienteerde programmeertaal. Ze werd ontwikkeld door Microsoft en werd samen met het .NET framework geïntroduceerd. C# gelijkt qua syntax zeer hard op Java, een andere populaire object-georiënteerde programmeertaal.

Het .NET framework is gemaakt om verschillende applicaties en libraries te laten samenwerken met verschillende programmeertalen. Wanneer bijvoorbeeld C# of J# (een andere .NET taal) code gecompileerd wordt zal deze eerst omgezet worden naar een tussentaal. Deze tussentaal is dan gelijkwaardig en zorgt dus voor een naadloze samenwerking met de verschillende facetten van het framework.

Visual Studio, een IDE[3] voor C#, biedt naast een gebruikelijke code editor ook een grafische editor. Via het betere klik- en sleepwerk kan zo een heuse grafische applicatie gemaakt worden, volledig met menu's, knoppen en tekstvelden.

[3]Integrated Development Environment - De omgeving waarin computerprogramma's geschreven worden.

2.5 SAX en DOM

SAX en DOM zijn twee verschillende aanpakken om XML te parsen.

Bij het SAX model wordt de informatie onmiddellijk doorgegeven wanneer ze wordt ingelezen door de parser (push), terwijl bij DOM heel het document wordt ingeladen vooraleer de informatie eruit kan gehaald worden (pull).

Enkele redenen om voor SAX te kiezen:

- Het document past niet in het geheugen. (Dankzij de 'streaming' is er steeds maar een klein deel van het document in het geheugen)
- Het document kan stuk per stuk verwerkt worden. Het is niet nodig dat het gehele document beschikbaar is om verder te werken.

Redenen om voor DOM te kiezen:

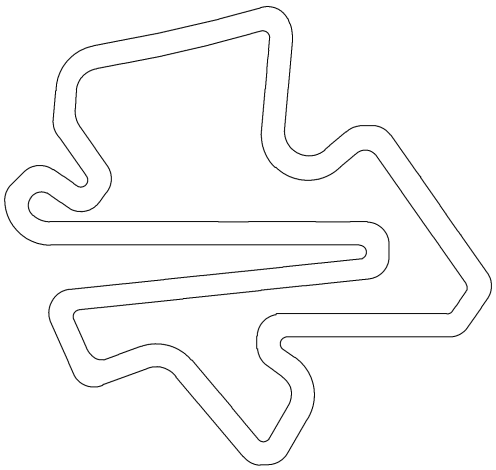
- Voor de verwerking van het document zijn stukken informatie uit het gehele document of zelfs andere documenten nodig.
- De datastructuur van het document is zeer gecompliceerd.
- Het document moet verscheidene malen aangepast worden.
- Het document is voor lange tijd nodig in het programma.

3 Het circuit

3.1 Beschrijving

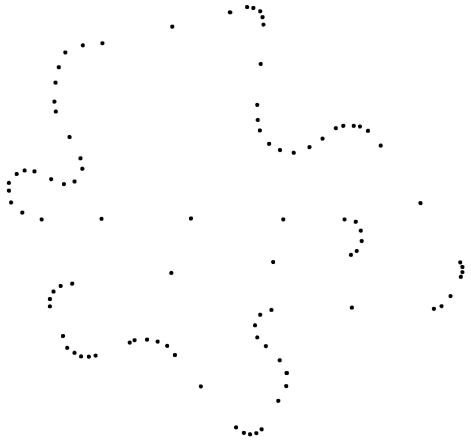
Een belangrijke vraag die we ons moesten stellen is: hoe kan een circuit beschreven worden, zodat het verstaanbaar en gemakkelijk bruikbaar is voor de simulatie. Verschillende mogelijkheden, waaronder Bézier curves, zijn overwogen.

De gekozen oplossing kan uitgelegd worden in volgende stappen:



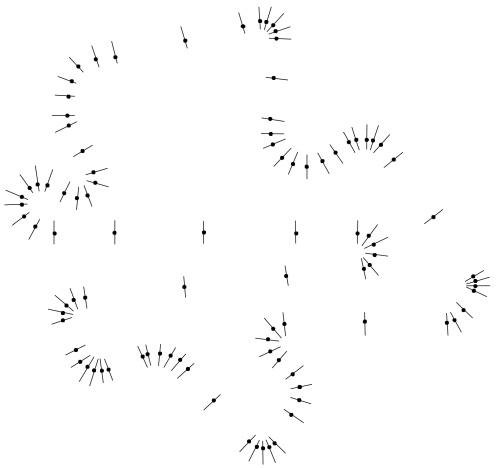
Figuur 1: Te beschrijven circuit

- Een reeks punten (vanaf nu trackpoints genoemd) wordt over de lengte van de baan uitgezet.



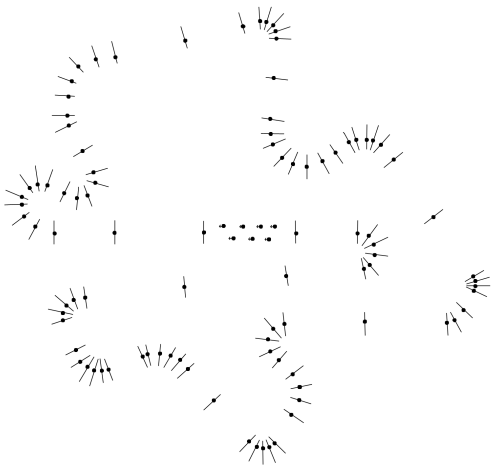
Figuur 2: Trackpoints

- Voor elk punt wordt een baanbreedte en offset op die baanbreedte vastgelegd.



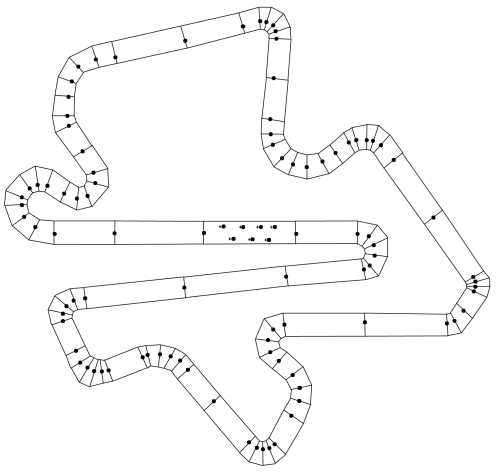
Figuur 3: Trackpoints met breedte

- Startposities zijn eveneens punten, ditmaal met een rotatie



Figuur 4: Startposities

- Wanneer nu intern een volledige representatie nodig is van het circuit (zie hoofdstuk 4.3.2), kan deze bekomen worden door de zijkanten van de trackpoints te verbinden.



Figuur 5: Reconstructie

Uiteraard hangt de nauwkeurigheid van de reconstructie sterk af van de hoeveelheid trackpoints, zoals bij elk bemonsteringsproces[4].

[4]Het bemonsteringstheorema stelt: om aliasing te voorkomen moet bemonsterd worden op een snelheid van minstens 2 maal de hoogste frequentie van het signaal. In dit geval worden de frequenties bepaald door de scherpte van bochten: zeer scherpe bochten moeten met veel trackpoints gedefiniëerd worden om niet schokkerig gereconstrueerd te worden.

3.2 De RaceSimTrack file

Aangezien de circuits worden opgesteld in TrackEditor (zie hoofdstuk 12), maar worden gebruikt in de simulatie, moet er een manier zijn om de informatie over te brengen. De meest logische en gemakkelijkste manier is het gebruik van files. Om de informatie duidelijk op te slaan wordt gebruik gemaakt van XML markup. Deze files heten RaceSimTrack- of RST files.

3.2.1 DTD

Specificatie voor de RaceSimTrack file.

```
<!-- Root element -->
<!ELEMENT track (info, startgrid, points)>

<!-- Track info -->
<!ELEMENT info (title, background)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT background (#PCDATA)>

<!-- Stargrid info -->
<!ELEMENT startgrid (position+)>
<!ELEMENT position (x, y, rotation)>
<!ELEMENT rotation (#PCDATA)>

<!-- Lijst trackpoints -->
<!ELEMENT points (point+)>

<!-- TrackPoint -->
<!ELEMENT point (x, y, width, offset)>
<!ELEMENT width (#PCDATA)>
<!ELEMENT offset (#PCDATA)>

<!ELEMENT x (#PCDATA)>
<!ELEMENT y (#PCDATA)>
```

3.3 Inlezen

Om banen beschikbaar te maken in de simulatie moeten natuurlijk de trackfiles ingeladen worden. Omdat deze opgesteld zijn in XML is een XML parser

nodig om ze in te laden. De gebruikte library heet Xerces-C[5].

3.3.1 XML parsing met Xerces-C

Xerces-C is een XML parser geschreven in C++. Zowel DOM als SAX (zie hoofdstuk 2.5) zijn beschikbaar, maar SAX komt het meest tegemoet aan de noden van de simulatie:

- hoge snelheid
- laag geheugengebruik (trackfile moet niet in geheugen blijven)

Dankzij de simpele interface is het mogelijk met zeer weinig eigen code al de nodige informatie te winnen uit de in te lezen XML files. Enkel de nodige Handler klasse moet overgeërfd worden, waarbij de drie belangrijkste te implementeren methodes: *startElement*, *endElement* en *characters* zijn. Deze klasse wordt door Xerces gebruikt om de overeenkomstige events door te geven aan de programmeur, zodat deze de informatie kan gebruiken in zijn eigen code.

Op deze manier wordt een trackfile ingelezen en de trackpoints en startposities bijgehouden wanneer ze tegengekomen worden.

3.3.2 Reconstructie level

Wanneer de file is geparsed wordt het level gereconstrueerd aan de hand van de gewonnen informatie. Een Level object wordt aangemaakt, de trackpoints worden overlopen en verder verwerkt (hoekberekeningen voor rasterizatie, zie hoofdstuk 4.3.2) en worden toegevoegd aan het object, samen met de startposities.

[5] http://xerces.apache.org/xerces-c/

4 Artificiële intelligentie

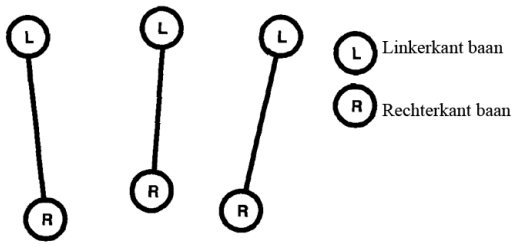
4.1 AI in een racesimulatie

Er moet onderscheid gemaakt worden tussen twee types AI wanneer men spreekt over AI voor races. Namelijk AI die probeert realistisch te *lijken*, en AI die probeert realistisch te *zijn*. Dit wil zeggen dat er bij het eerste type volop vals gespeeld wordt door de AI informatie te verschaffen, die in het echt niet beschikbaar is. Deze AI zal vrij gelimiteerde mogelijkheden hebben op gebied van uitbreidbaarheid en zal bijvoorbeeld geen circuits kunnen berijden die deze extra informatie niet bevatten; er kan zelfs geargumenteerd worden dat dit geen echte AI is.

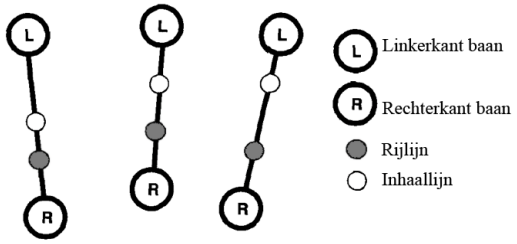
Het tweede type daarentegen, zal zó gemaakt zijn dat hij beslissingen maakt aan de hand van de informatie die hij zelf vindt. Ter illustratie, zie Figuren 6 en 7, waarbij Figuur 6 onze manier van werken is.

Deze twee types zijn de extremen; de meeste AI engines zullen ergens daartussen liggen, afhankelijk van kleine nuances.

De AI engine die we zelf ontwikkeld hebben doelt erop om realistisch te zijn.



Figuur 6: Circuit met enkel basis informatie



Figuur 7: Circuit met extra informatie voor de AI

4.2 AI in deze racesimulatie

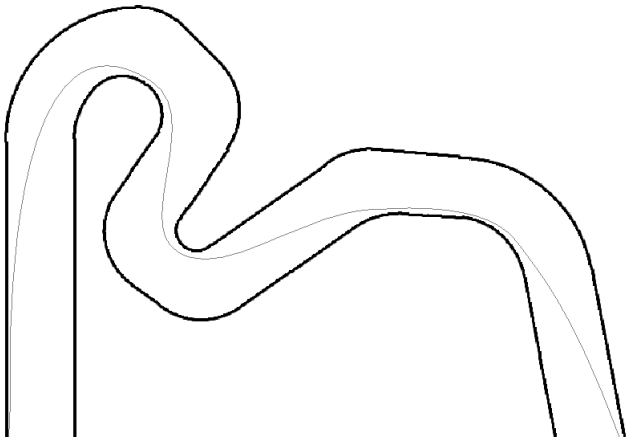
De artificiële intelligentie in deze simulatie staat in voor het competitief doen rijden van auto's over een racebaan. Ze kan opgedeeld worden in enkele belangrijke onderdelen:

- Pathfinding; om de ideale lijn te bepalen
- Neuraal netwerk; om de gevonden lijn te volgen
- Flocking; om obstakels te ontwijken
- Machine learning; om het rijgedrag te verbeteren

Deze worden verduidelijkt in volgende hoofdstukken.

4.3 Pathfinding

Alvorens een autobestuurder een beslissing kan maken in een gegeven situatie, moet hij eerst weten waarnaartoe hij wil rijden. Deze regel geldt ook voor deze simulatie. Er moet een zeker pad gevonden worden (liefst op de baan), om over een circuit te navigeren. Aangezien het om een racesimulatie gaat, is dit pad de rijlijn die ook echte coureurs gebruiken: de zogenaamde ideale lijn. Deze fictieve lijn definiëert het snelste pad langs bochten en rechte stukken, waarbij de auto zoveel mogelijk zijn snelheid kan behouden (ter illustratie, zie Figuur 8). Om deze lijn te vinden is er gebruik gemaakt van het A* pathfinding algoritme.

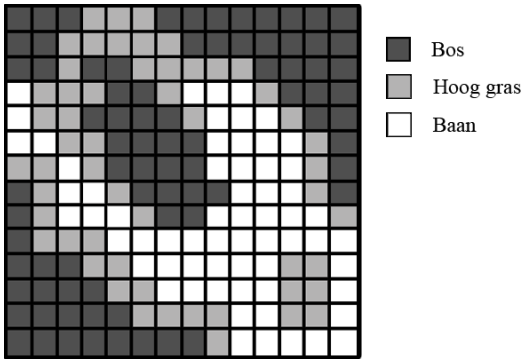


Figuur 8: Ideale lijn

4.3.1 A*

Het A* algoritme is een van de meest gebruikte pathfinding algoritmes vandaag de dag. Het resulteert altijd in het snelste pad tussen twee punten en is relatief efficiënt. Let hier op, het gaat niet om het kortste pad, maar wel degelijk om het snelste pad, aangezien er kan rekening gehouden worden met factoren als vertragend terrein.

De basis van het A* algoritme is het zoekgebied. Dit is een set van nodes waarlangs er kan gereisd worden. Meestal komt dit neer op een raster met nodes die een bepaald terrein voorstellen.



Figuur 9: Voorbeeld raster

Wanneer de zoekset gedefinieerd is, kan er gezocht worden naar het snelste pad.

Hierbij wordt gebruik gemaakt van twee lijsten: de open lijst en de gesloten lijst. De eerste bevat de gevonden nodes die nog moeten onderzocht worden, de tweede bevat deze die al onderzocht zijn.

Bij de eerste stap wordt de startnode toegevoegd aan de gesloten lijst. Er wordt onderzocht welke nodes kunnen bereikt worden vanuit deze node en deze worden toegevoegd aan de open lijst om later zelf onderzocht te worden. Voor elk van deze gevonden nodes wordt hun kostprijs berekend. Dit is de belangrijkste factor voor de kwaliteit van het pad: de kostprijs van een node staat voor de moeite om te reizen naar deze node en de voorspelde moeite nodig om van die node verder te reizen naar het einddoel. Hierdoor kan de ligging van het pad beïnvloed worden.

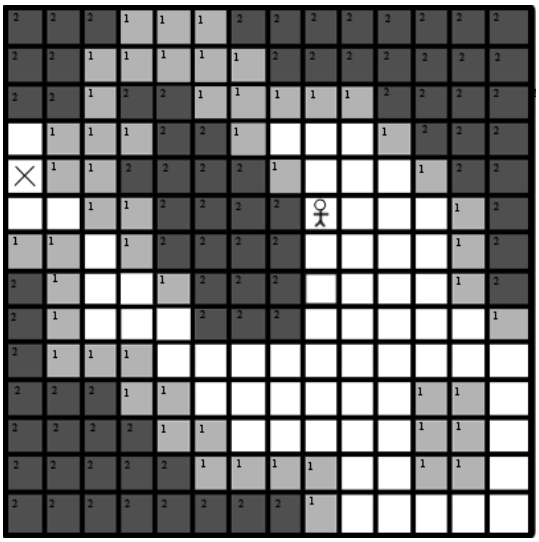
Wanneer nu de volgende iteratie begint, wordt de node uit de open lijst met de laagste kost gekozen om te onderzoeken. Opnieuw worden gevonden nodes toegevoegd aan de open lijst en wordt hun kostprijs berekend. Doordat telkens de node met de laagste kostprijs wordt gekozen zal het pad relatief

snel gevonden worden en zal het bestaan uit de nodes die samen de laagste kost hebben.

In pseudocode:

```
voeg de begin node toe aan de open lijst
zolang open lijst niet leeg is
  huidige node = vind node met laagste kost in open lijst
  als huidige node = doel node
    pad gevonden
  anders
    voeg huidige node toe aan gesloten lijst
    onderzoek elke node bereikbaar vanaf de huidige node
    voor elke gevonden node
      als de node niet in de gesloten lijst zit
         en niet in de open lijst
           bereken dan de kost en voeg hem toe aan de open lijst
```

Ter illustratie Figuur 10 toont de beginsituatie. De zoekset met als startnode het mannetje en als doel het kruis.



Figuur 10: Stap 1: Zoekset met startpositie en doel

Figuur 11. De startnode wordt toegevoegd aan de gesloten lijst en er wordt gekeken welke nodes bereikbaar zijn. De acht gevonden nodes worden toegevoegd aan de open lijst en hun kostprijs wordt berekend. Deze kostprijs bestaat uit de kost om tot die node te reizen van de startnode (s), de eventuele terreinkost (t) en de voorspelde kost om tot het einddoel te komen (v).

Vb kostberekening:

- de node onder de startnode:

  s: 1

  t: 0

  v: 8 [6]

  totaal: 9

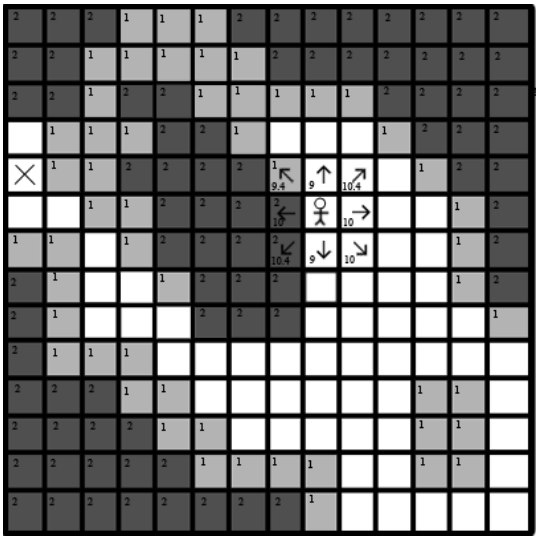
- de node links onder de startnode:

  s: 1.4 [7]

  t: 2

  v: 7

  totaal: 10.4



Figuur 11: Stap 2: 8 Gevonden nodes met kostprijs

Figuur 12. De node in de open lijst met de laagste kost wordt gekozen en naar de gesloten lijst verhuisd. Aangezien er twee met kost 9 zijn, wordt er willekeurig één genomen. Opnieuw wordt er gekeken welke nodes bereikbaar zijn en hun kost wordt berekend. Ditmaal worden er maar drie gevonden, omdat de andere 5 reeds in de open of gesloten lijst zitten.

Vb. kostberekening:

[6] Om het simpel te houden: het aantal nodes tot het doel. Rekening houden met schuine nodes die 1.4 kosten zou een nog efficiënter algoritme opleveren.

[7] Afronding $\sqrt{1+1} = 1,4142135....$ Voor berekening, zie hoofdstuk 13.1

- de node onder de huidige node:

  s: $1 + 1 = 2$

  t: 0

  v: 8

  totaal: 10

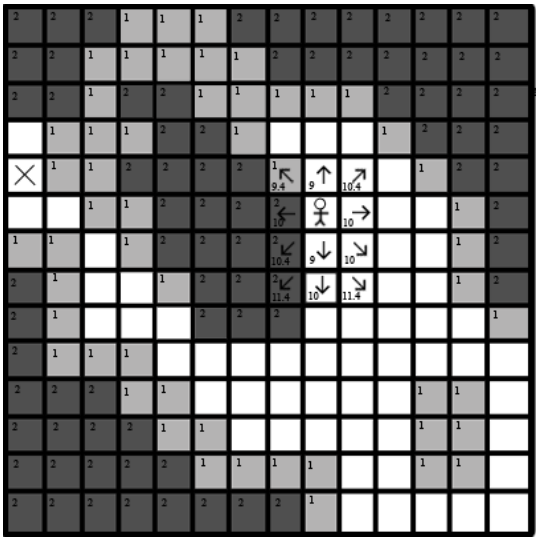
- de node links onder de huidige node:

  s: $1 + 1.4 = 2.4$

  t: 2

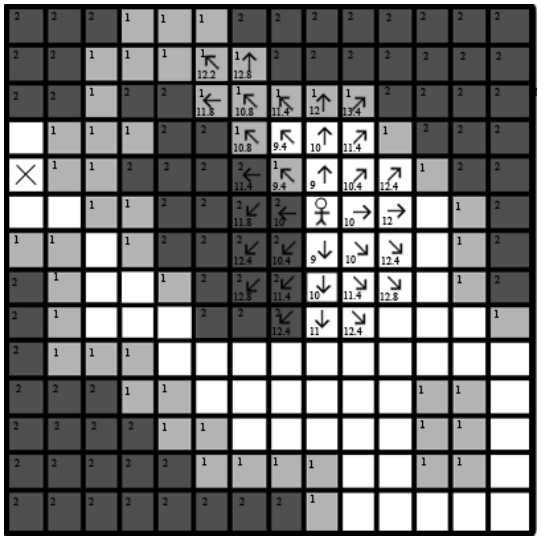
  v: 7

  totaal: 11.4



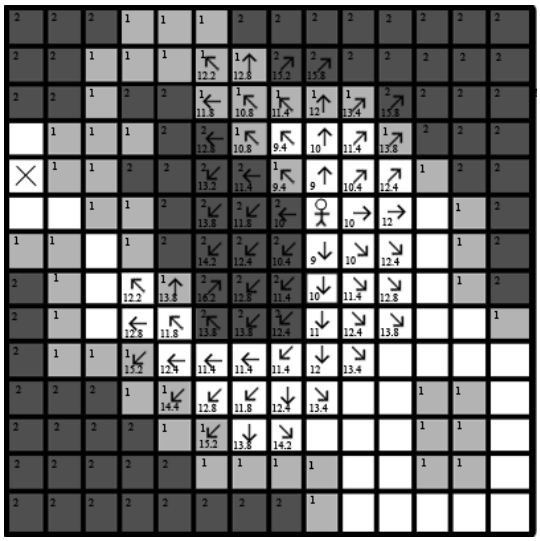
Figuur 12: Stap 3: 3 Gevonden nodes met kostprijs

Figuur 13. Dezelfde stappen worden herhaald. Het beste pad lijkt langs boven te liggen.



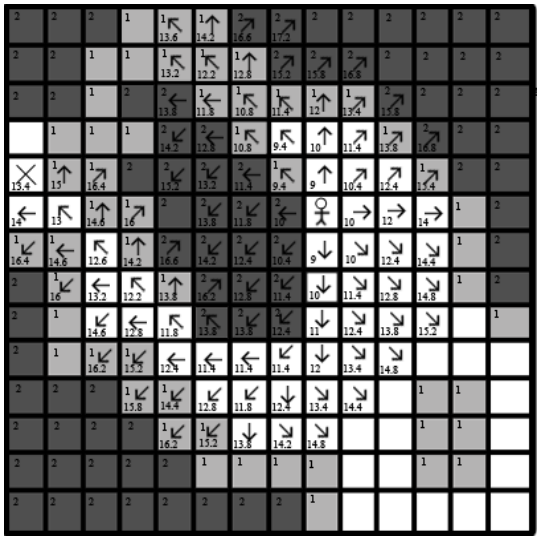
Figuur 13: Stap n

Figuur 14. Dezelfde stappen worden herhaald. Boven wordt op moeilijker terrein gestoten zodat de kosten van de nodes te hoog liggen, een beter pad lijkt langs de onderkant van het bos te liggen.



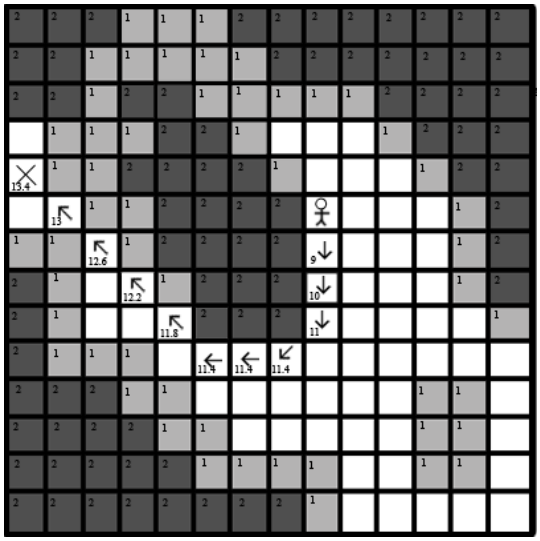
Figuur 14: Stap m

Figuur 15. Uiteindelijk bevindt de eindnode zich onder de gevonden nodes. Het pad is gevonden en moet nu gereconstrueerd worden door terug te lopen vanaf de laatste node.



Figuur 15: Laatste stap: Doel gevonden

Figuur 16. Het gevonden pad.



Figuur 16: Gevonden pad

4.3.2 Rasterizatie

Om A* pathfinding te kunnen uitvoeren op de baan, moet deze van een continue omgeving omgezet worden naar een tile-based omgeving[8]. Hiervoor

[8]Vergelijkbaar met het nemen van een foto: het omzetten van een continu beeld naar een reeks pixels.

wordt een zelf ontwikkeld rasterizatie algoritme gebruikt, dat aan de hand van de informatie over het level (zie hoofdstuk 3) een raster met terrein informatie opstelt.

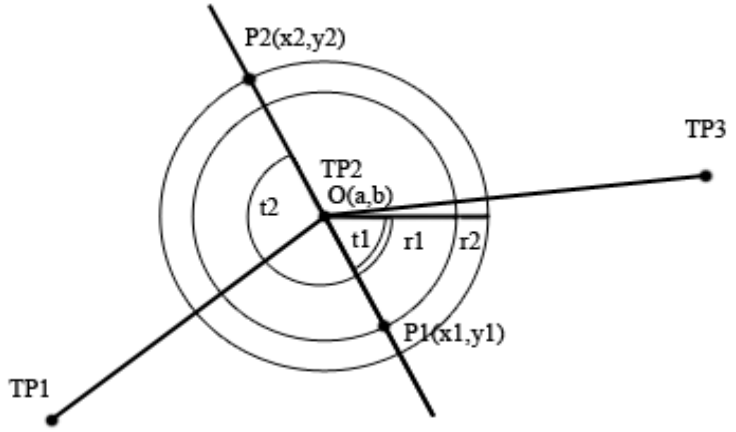
De punten gedefiniëerd in de trackfile worden overlopen en telkens worden de twee bijbehorende punten op de zijkant van het circuit berekend. Dit gebeurt aan de hand van het principe beschreven in hoofdstuk 13.2 en uitgetekend in Figuur 17. TP_1, TP_2 en TP_3 zijn opeenvolgende trackpoints, P_1 en P_2 zijn de zijkanten van de baan bij TP_2. P_1 en P_2 worden respectievelijk berekend met straal r_1 en hoek t_1 en straal r_2 en hoek t_2.

$$r_1 = \frac{breedteBaan}{2} - offsetPunt$$

$$r_2 = \frac{breedteBaan}{2} + offsetPunt$$

$$t_1 = \frac{((TP_1 \measuredangle TP_2) + (TP_2 \measuredangle TP_3))}{2}$$

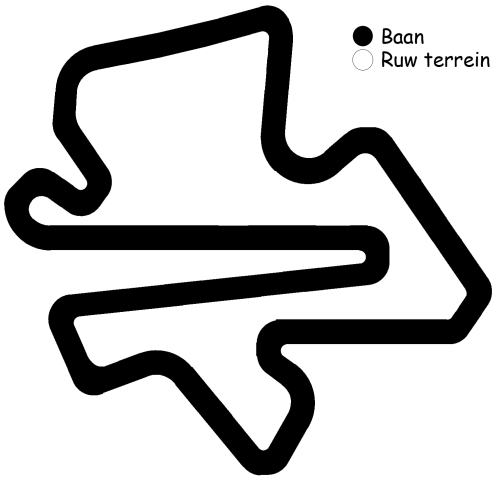
$$t_2 = t_1 + \pi$$



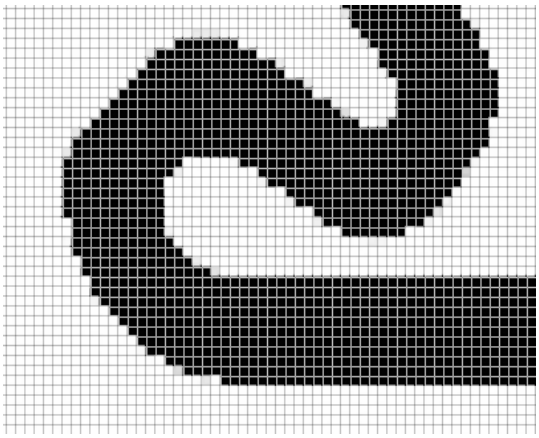
Figuur 17: Berekenen zijkant baan

De berekende punten worden verbonden met de vorige punten met behulp van Bresenham's lijn algoritme[9] om op deze manier een vierhoek te creëren. Nadien wordt deze vierhoek opgevuld door op elke rij (indien van toepassing), twee punten met elkaar te verbinden. Gevisualiseerd resulteert dit in Figuren 18 en 19.

[9]Een efficiënt algoritme om een lijn te trekken tussen twee punten met het minst mogelijke pixels of nodes



Figuur 18: Visualisatie raster



Figuur 19: Closeup raster

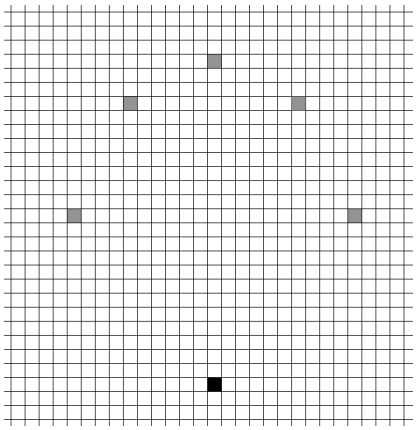
4.3.3 Eigen implementatie

Bereikbare nodes Ten eerste zijn nodes die niet op het circuit liggen volledig onbereikbaar. Er zal geen rekening mee gehouden worden, hetgeen logisch is en daarenboven het zoeken versneld.

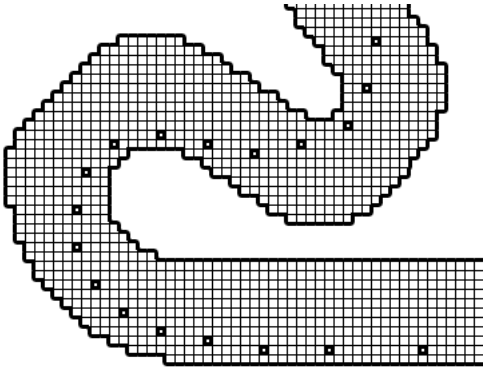
Ten tweede. Meestal zal bij A* pathfinding tussen aanliggende nodes gereisd worden, zoals getoond in het voorbeeld in hoofdstuk 4.3.1. Dit resulteert in een nauwkeurigheid van amper 45° bij het draaien[10]. Zo'n nauwkeurigheid is onacceptabel bij pathfinding voor auto's die in theorie $\frac{1}{\infty}$ nauwkeurig zijn.

[10] $\frac{360°}{8\ nodes}$

Om dit probleem te voorkomen, wordt bij het zoeken naar bereikbare nodes rekening gehouden met een echte racesituatie. Dit wil zeggen dat met elke node een snelheid en hoek geassociëerd wordt. Wanneer nu wordt gezocht naar volgende nodes zal er gekeken worden waar de auto uitkomt wanneer hij in de situatie bij de huidige node over een bepaald tijdsinterval gas geeft, remt, draait of een combinatie doet van de drie[11]. Dit levert nodes op als getoond in Figuren 20 en 21.



Figuur 20: Bereikbare nodes



Figuur 21: Toepassing pad

Ten derde wordt voor elke gevonden node gecontroleerd of alle nodes tussen deze en de vorige circuitnodes zijn en er in een cirkel van bepaalde grootte ook enkel circuitnodes zijn. Dit om respectievelijkte vermijden dat bij twee aanliggende baanstukken er over de scheiding kan 'gesprongen' worden en dat er niet te dicht bij de rand gereden wordt.

[11] Deze berekeningen volgen het principe beschreven in hoofdstuk 13.2

Kost en gewicht Zoals reeds vermeld is bij A* pathfinding de keuze van kosten voor nodes zeer belangrijk voor de ligging van het uiteindelijke pad. Om in de richting van het doel te blijven zoeken is het belangrijk rekening te houden met de afstand tot het doel. Daarnaast zijn er de specifieke kosten voor deze simulatie. Er wordt rekening gehouden met het verschil in snelheid tussen deze en de vorige node en het verschil in hoek. De verhouding tussen deze twee kan aangepast worden met een gewicht, zie hoofdstuk 4.5. Dankzij deze gewichten wordt het pad gevonden waar volgens de berekeningen de gemiddelde snelheid het hoogst is.

Performantie De duurste operatie bij A* pathfinding is het overlopen van de lijsten. Om dit op te vangen is er gebruik gemaakt van een tweede raster waar de status van nodes wordt opgeslagen, ter vervanging van de gesloten lijst. Dit zorgt ervoor dat voor elke node met één enkele operatie kan gekeken worden of hij in de gesloten lijst zit. Het enige nadeel hiervan is het grotere geheugengebruik, aangezien er een tweede raster van dezelfde grootte als het gerasteriseerde level moet worden gealloceerd.

Een tweede unieke feature is de mogelijkheid om de pathfinding over verschillende keren te spreiden. Met het oog op realtime pathfinding is dit zeker nuttig. Zo kan de pathfinding gespreid worden over enkele frames, in plaats van om dz zoveel tijd een frame langer te laten duren dan voorzien. Dit wordt geïmplementeerd door een grens te leggen aan het aantal nodes dat per keer wordt onderzocht. Indien het pad niet gevonden is voor de grens bereikt is, wordt de huidige status bijgehouden, zodat die later kan verdergezet worden.

Nog een belangrijke maatregel is het schalen van het raster. Door de grootte van het raster 4, 8 of 16 keer kleiner te maken, zal de pathfinding enorm veel sneller gaan. Toch moet hier opgepast worden, er moeten namelijk nog genoeg nodes aanwezig zijn om een goed pad te kunnen vinden. Hierbij was het dus nodig de juiste balans te vinden tussen performantie en kwaliteit.

Enkele andere kleine wijzigingen ten voordele van de performantie:

- Variabelen op voorhand declareren; dit maakt enorm verschil in methodes die duizenden al dan niet tienduizenden keren worden opgeroepen.
- Het limiteren van het aantal te vinden nodes; hier moest dus ook een goede balans gevonden tussen de performantie en de kwaliteit van de pathfinding.

4.4 Besturing

Nadat het te berijden pad is bepaald, moet de AI beslissen welke handelingen te ondernemen in de gegeven situatie en in functie van het pad. Om zo'n flexibiliteit te bekomen is er gebruik gemaakt van een neuraal netwerk.

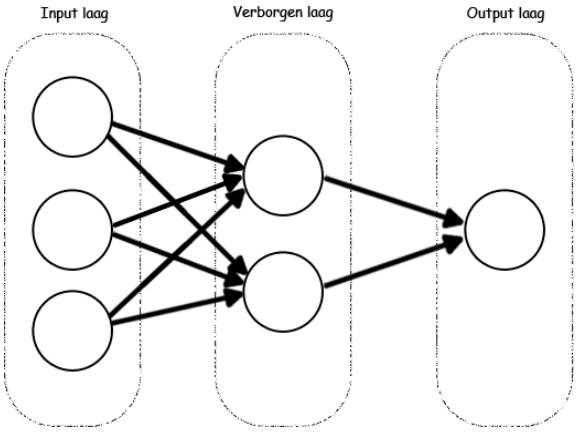
4.4.1 Neuraal netwerk

Beschrijving Een neuraal netwerk gaat voort op de samenstelling en werking van de hersenen van de mens. Een belangrijk concept hierbij is de neuron: een deeltje in het brein waarvan er miljarden bestaan, elk met duizenden anderen verbonden. Deze neuronen zorgen voor de gigantische rekenkracht van de hersenen. 'Informatie' wordt overgebracht van een geactiveerde neuron naar de andere via een axon door middel van een zeker voltage. Neuronen worden geactiveerd wanneer er genoeg geconnecteerde neuronen zelf hun geactiveerd signaal ernaar gestuurd hebben. Dit wil zeggen dat een neuron een reeks andere neuronen heeft van dewelke hij signalen ontvangt en een reeks neuronen naar dewelke hij zelf zijn signaal stuurt. Samen coördineren ze acties, nemen ze beslissingen, enz.

Een artificiëel neuraal netwerk implementeert dit concept, weliswaar op veel kleinere schaal. Toch is het zeer goed in het verwerken van zeer niet-lineaire problemen die bijna onmogelijk op te lossen zijn met meer conventionele methodes. Dit kan er wel voor zorgen dat het niet altijd duidelijk is wat het neuraal netwerk doet en hoe het tot zijn resultaten komt. Zeker bij het testen is dit een struikelblok.

Evenals een echt brein opgeleerd wordt tijdens de jeugd (en later), moet een artificiëel neuraal netwerk getraind worden alvorens het gebruikt kan worden. Deze training is specifiek aan het probleem en de kwaliteit ervan zal sterk de werking van het netwerk beïnvloeden.

Structuur en werking Het gebruikte type neuraal netwerk heet: een three-layered, feedforward network. Dit wil zeggen dat het netwerk uit drie lagen bestaat, namelijk een input laag, een verborgen laag en een output laag. Het aantal neuronen in elke laag mag vrij gekozen worden. Het aantal inputneuronen hangt af van het aantal variabelen in de probleemstelling en het aantal outputneuronen hangt af van het aantal benodigde antwoorden. De keuze van het aantal neuronen in de verborgen laag is ook zeer belangrijk, aangezien hier de hoeveelheid neuronen bepaald hoeveel 'features' het netwerk zal aankunnen. Dit zal verderop geïllustreerd worden.



Figuur 22: Voorbeeld neuraal netwerk

Elke neuron in de input laag is verbonden met elke neuron in de verborgen laag. Evenzo zijn alle neuronen in de verborgen laag verbonden met alle neuronen in de ouput laag. Daarenboven heeft elke neuron, behalve in de input laag, een bijkomende input, de bias genaamd.

Wanneer er een berekening dient gemaakt te worden, moeten de inputwaarden doorgegeven worden aan de overeenkomstige inputneuronen[12]. Aan de hand van deze waarden zal de ouput van elke inputneuron bepaald worden en deze wordt dan doorgegeven aan de neuronen in de verborgen laag. Hetzelfde gebeurt voor de neuronen in de verborgen en output lagen. De outputwaarden van de ouputneuronen zijn de uiteindelijke resultaten van het proces. Dit geheel heet het feed-forwarden van het netwerk.

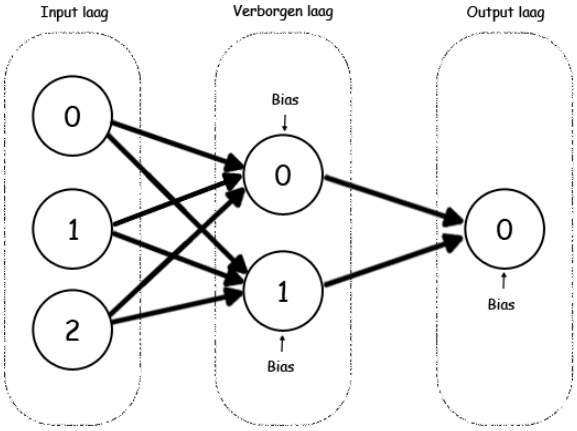
Het berekenen van de input van een neuron is niet zo simpel als het nemen van de som van alle ouputs van de neuronen in de vorige laag. Om het netwerk de kans te geven zelf het onderling belang van neuronen uit te werken voor een bepaald gedrag, wordt er gebruik gemaakt van gewichten bij connecties tussen neuronen. Deze waarden beïnvloeden de grootte en dus het uiteindelijk belang van deze connecties. Tijdens het trainen van het netwerk worden deze gewichten aangepast om de relaties tussen neuronen aan te passen, zodat het gewenste gedrag kan bekomen worden.

De input voor een neuron j kan berekend worden met de inputs van een set inputneuronen i volgens volgende functie:

$$n_j = \sum n_i w_{ij} + b_j w_j$$

[12]Deze waarden kunnen onder verschillende vormen aangeboden worden, doch het moeten steeds reële getallen zijn. Er zal vanuitgegaan worden dat deze waarden steeds in het bereik 0.0 en 1.0 zullen liggen.

Elke input(n_i) voor een neuron(n_j) is vermenigvuldigd met het gewicht van de connectie tussen de twee neuronen(w_{ij}). De som hiervan plus de bias(b_j) met het biasgewicht(w_j) vormt de totale inputwaarde van een neuron.

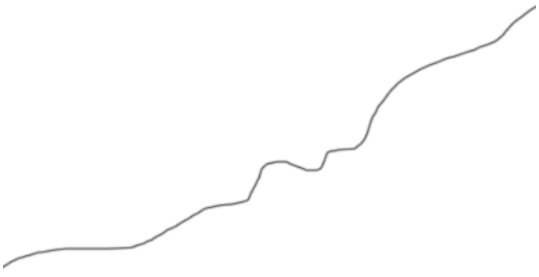


Figuur 23: Voorbeeld neuraal netwerk

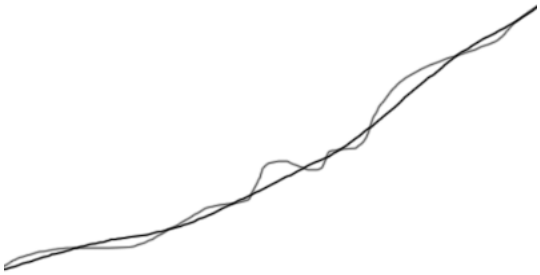
Bekijken we het specifieke geval van de eerste neuron in de verborgen laag, dan krijgen we deze formule:

$$n_0^v = n_0^i w_{00} + n_1^i w_{10} + n_2^i w_{20} + b_0^v w_{b0}$$

Nadat de input van de neuron berekend is moet deze geactiveerd worden om zijn output te berekenen. De manier waarop heeft veel invloed op het gedrag van het netwerk als functie benaderaar. De functie die benaderd moet worden is deze van het gewenste gedrag in functie van de inputwaarden. Het netwerk zal proberen het gedrag van deze functie te repliceren. Dit moet echter niet te goed gedaan worden, anders zal het netwerk niet meer in staat zijn waarden die niet in de trainingsset voorkomen te generaliseren.



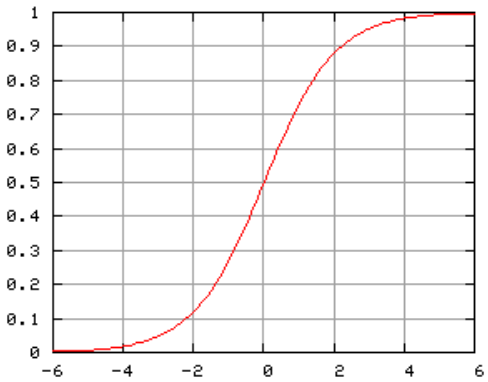
Figuur 24: Functie met veel ruis



Figuur 25: Slechte benadering functie: de algemene trend wordt beschreven, maar locale maxima en minima worden gemist

Om niet-lineaire functies als in Figuur 24 te benaderen is een niet-lineaire neuronactivatie nodig. Hiervoor wordt de zogenaamde sigmoid functie gebruikt, gevisualiseerd in Figuur 26 en beschreven met deze formule:

$$f(x) = \frac{1}{(1 + e^{-x})}$$



Figuur 26: Sigmoid functie

Het laatste belangrijk onderdeel van het neuraal netwerk is de training. De bedoeling ervan is aan de hand van een trainingsset een set gewichten te vinden voor het netwerk die het gewenste gedrag repliceren. Deze trainingsset bestaat uit inputwaarden en de gewenste outputwaarden. Het trainingsproces bestaat uit volgende stappen:

1. Initialiseer de gewichten in het netwerk met kleine random waarden

2. Voor elke set inputdata, voedt het netwerk en bereken de output

3. Vergelijk de bekomen output met de gewenste output en bereken de afwijking

4. Pas de gewichten aan zodat de afwijking kleiner wordt en herbegin met stap 2

Dit heet 'back-propagation training'.

Er bestaan verschillende manieren om de afwijking te berekenen. De gekozen manier doet dit door het gemiddelde te nemen van het kwadraat van het verschil tussen bekomen en gewenste output: $\epsilon = \frac{\sum (n_g - n_b)^2}{m}$. ϵ is de algemene afwijking voor het netwerk, n_b en n_g zijn respectievelijk de bekomen en de gewenste outputs en m is het aantal outputneuronen.

Alvorens de gewichten van elke neuronconnectie aan te passen, moeten nog enkele berekeningen gedaan worden. De afwijking voor een specifieke outputneuron wordt berekend met deze formule: $\delta_i^o = \Delta n_i^o f'(n_{bi}^o)$. δ_i^o is de afwijking voor de neuron i in de ouput laag, Δn_i^o het verschil tussen bekomen en gewenste output en $f'(n_{bi}^o)$ is de afgeleide van de activatiefunctie (sigmoid). Deze afgeleide wordt berekend volgens het principe beschreven in hoofdstuk 13.3. Herschreven is dit $\delta_i^o = (n_{gi}^o - n_{bi}^o) n_{bi}^o (1 - n_{bi}^o)$.

Voor het berekenen van de afwijking van een neuron in de verborgen laag moet een licht aangepaste formule gebruikt worden: $\delta_i^v = \left(\sum \delta_j^o w_{ij}\right) f'(n_{bi}^v)$. De afwijking van een verborgen neuron is een functie van de afwijkingen van de neuronen in de output laag vermenigvuldigd met het gewicht van de connectie. De afgeleide wordt op dezelfde manier verkregen als in de vorige formule.

Nu de afwijkingen voor elke neuron berekend zijn, kunnen de gewichten aangepast worden. Dit gebeurt met deze formule: $\Delta w = \rho \delta_i n_i$. ρ is de leersnelheid, δ_i de afwijking voor neuron i en n_i is de waarde van neuron i. Het nieuwe gewicht is het oude plus Δw. De leersnelheid bepaalt hoe snel en óf de optimale gewichten zullen bereikt worden tijdens het trainen.

Logica De logica achter het netwerk is vrij simpel en toch krachtig genoeg om op alle mogelijke situaties waarin de auto zich kan bevinden correct te reageren. Deze logica gaat uit van twee punten langs het berekende pad, één dichtbij en één verderaf. Aan de hand van de hoek tussen de auto en de punten en de afstand tot die punten wordt ingeschat hoe het pad ligt. Met deze informatie en de huidige snelheid van de auto kan het netwerk beslissen hoe te reageren.

Hierbij zijn deze twee regels van groot belang:

1. Hoe kleiner de hoek met het punt, ...
2. Hoe verder het punt, ...

... des te rechter het pad en des te meer gas mag gegeven worden. Bij het omgekeerde geval zal er geremd en/of gedraaid worden.

Ter illustratie:

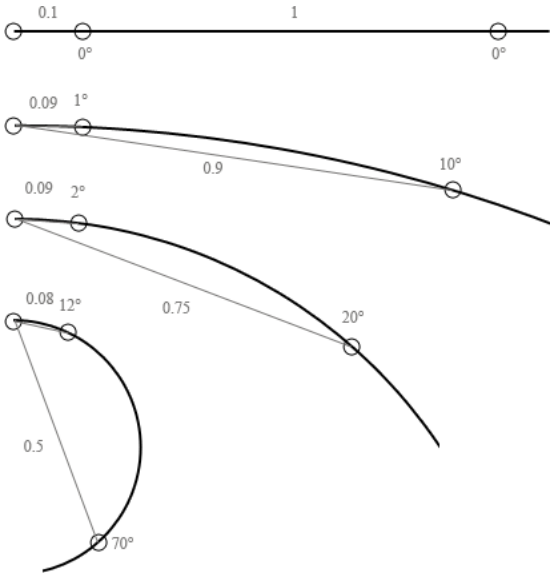
Beschouw Figuur 27. Vier voorbeeldpaden worden hierin gevisualiseerd. (De zwarte lijn stelt het pad voor, de eerste cirkel de positie van de auto en de twee volgende cirkels de twee punten waarmee rekening gehouden wordt)

Bij het eerste pad staan de twee berekende punten beiden onder een hoek van 0° ten opzichte van de auto en liggen zo ver als ze kunnen zijn[13]. Daaruit kan met de logica geconcludeerd worden dat het pad in een rechte lijn ligt.

Bij paden twee en drie is de hoek met het tweede punt al beduidend groter, terwijl de afstand tot het punt slinkt. Met deze informatie stelt de logica respectievelijk een flauwe bocht en een iets sterkere bocht vast.

Het laatste pad is de uiterste vorm die kan tegengekomen worden: punt twee ligt onder een zeer grote hoek (70°) en ligt zeer dicht bij de auto (0.5 - de helft van de maximale lengte). Dit duidt op een scherpe bocht.

[13]De punten worden berekend op een vaste afstand langs het pad. Als nu deze afstand gelijk is aan de rechtstreekse afstand van de auto tot het punt, wil dit zeggen dat het pad een rechte lijn is (er bestaat geen kortere weg tussen twee punten dan een rechte lijn).



Figuur 27: Verschillende paden

Toepassing Het neuraal netwerk gebruikt in de simulatie is een netwerk met drie lagen die bestaan uit vijf inputneuronen, tien verborgen en drie outputneuronen. De betekenis van de neuronen is als volgt:

Input

1. Huidige snelheid
2. Hoek met het eerste punt
3. Afstand tot het eerste punt
4. Hoek met het tweede punt
5. Afstand tot het tweede punt

Output

1. Hoek die auto moet draaien
2. Hoeveel de gaspedaal moet ingedrukt worden
3. Hoeveel de rem moet ingedrukt worden

De code van het neuraal netwerk steunt sterk op die gevonden in "AI for Game Developers", zie literatuurlijst.

4.4.2 Trainingsset

De trainingsset voor een neuraal netwerk is van cruciaal belang voor de resultaten ervan. Het netwerk is maar zo goed als de trainingsset. Dit wil zeggen dat een zo groot en gevariëerd mogelijke set vereist is.

Origineel werd geprobeerd de trainingsset te laten genereren. Dit gebeurde door op een bepaald circuit, met genoeg variatie qua bochten en rechte stukken, een pad uit te zetten en er zelf over te rijden. Ondertussen werden dezelfde berekeningen (hoek, afstand tot twee punten) gemaakt als wanneer de AI zou rijden en bijgehouden als inputwaarden, terwijl de acties (remmen, sturen, gas geven) gemaakt door de tester als outputwaarden werden bijgehouden. Dit zou de perfecte trainingsset moeten opleveren. Twee problemen stonden dit scenario in de weg:

1. Het is zeer moeilijk om perfect over een uitgetekende lijn te rijden, hetgeen veel verkeerde waarden oplevert.
2. De persoon kan via het toetsenbord enkel binaire waarden sturen, met als gevolg dat er enkel volledig naar links of rechts gestuurd kan worden, of volledig geremd of gas gegeven kan worden. Dit vernietigt volledig de subtiliteit van het netwerk.

Hierna is geopteerd voor een trainingsset die de vooraf beschreven logica volgt. Enkele voorbeelden uit de set:

Tabel 1: Alle waarden zijn geschaald naar het bereik tussen 0.0 en 1.0
Bij hoeken is 0.5 rechtdoor, 0 links en 1 rechts

$Snelheid$	$HoekP_1$	$AfstandP_1$	$HoekP_2$	$AfstandP_2$	$Draaihoek$	Gas	Rem
Stilstand, beide punten rechtdoor ⇒ gas geven en niet draaien							
0	0.5	0.2	0.5	0.7	0.5	0.9	0
Hoge snelheid, onmiddellijk scherpe bocht rechts ⇒ rechts draaien en remmen							
1	0.8	0.1	1	0.6	0.85	0	1
Hoge snelheid, flauwe bocht links ⇒ licht links draaien en remmen							
1	0.4	0.2	0.3	0.7	0.38	0	1
Lage snelheid, scherpe bocht rechts ⇒ rechts draaien en remmen							
0.2	0.8	0.1	1	0.3	0.85	0	1

4.5 Machine learning

Een belangrijk en indrukwekkend onderdeel van een AI engine is machine learning[14]. Bij deze simulatie wordt geprobeerd de AI beter te leren rijden.

Eerst was het de bedoeling gebruik te maken van de trainingscapaciteiten van het neuraal netwerk. Hierbij moeten nieuwe trainigssets gegenereerd worden tijdens de simulatie door het evalueren van situaties. Met deze sets kan het netwerk dan bijgetraind worden. Het grote probleem hierbij is de moeilijkheid om goede trainingssets te genereren. De gegeven situatie moet correct ingeschat worden door na te gaan of de laatst genomen beslissing van het netwerk een goede beslissing was. Hierbij spelen zoveel factoren een rol en meestal kan pas na bepaalde tijd beslist worden of een beslissing correct was. Door de complexiteit van dit probleem is er gezocht naar een andere oplossing.

Een eerste alternatief was bijtrainen van het netwerk gebaseerd op waarden verkregen van een menselijke deelnemer, op dezelfde manier als beschreven in hoofdstuk 4.4.2, maar ditmaal realtime. Omwille van dezelfde redenen als in dat hoofdstuk heeft deze oplossing afgedaan.

De derde en finale aanpak is zowel simpel als effectief. Zoals reeds geschreven, worden de waarden geschaald alvorens naar het neuraal netwerk gevoerd te worden. Deze gewichten kunnen aangepast worden om zo het observatievermogen aan te passen. De verschillende gewichten zijn van toepassing op:

- De snelheid van de auto
- De afstand tot een punt
- De hoek met een punt
- De nodekosten bij pathfinding

Ter illustratie: door het verhogen van het snelheidgewicht ziet de auto zichzelf trager rijden en is de kans daardoor groter dat hij zal versnellen.

$$snelheid = 150$$

$$sgewicht = 100$$

$$\Rightarrow inputNeuraalNetwerk = \frac{150}{100} = 1.5 \Rightarrow 1.0$$

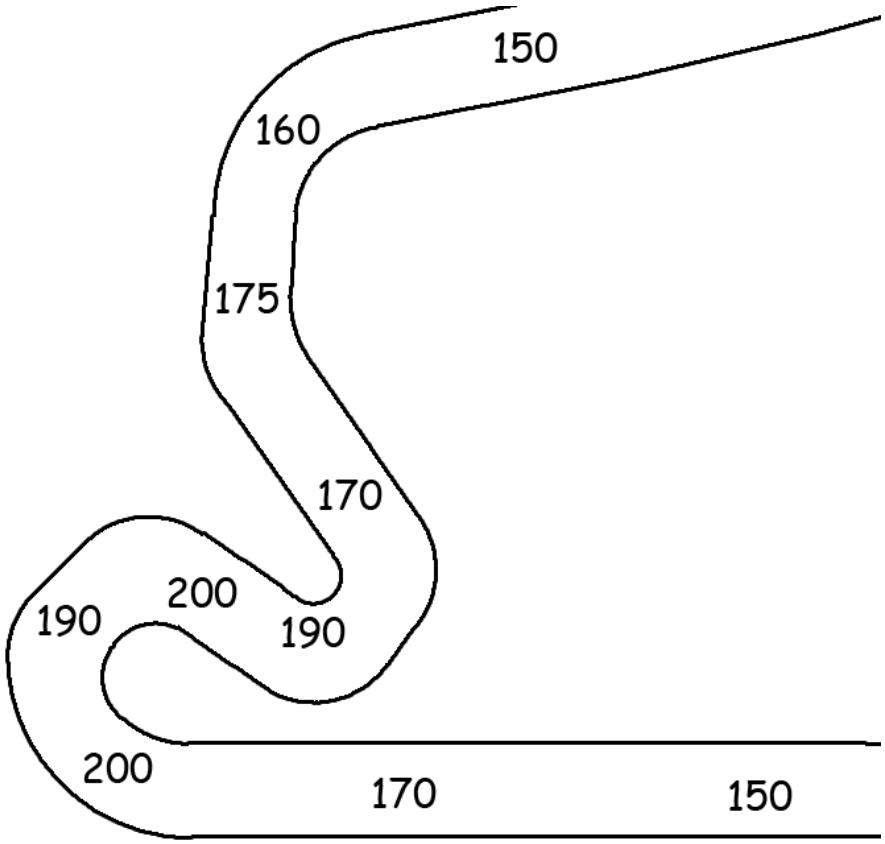
[14]Component dat instaat voor de mogelijkheid tot bijleren

$$snelheid = 150$$

$$sgewicht = 200$$

$$\Rightarrow inputNeuraalNetwerk = \frac{150}{200} = 0.75$$

Door deze gewichten te verspreiden over het circuit kan de AI leren een bepaald circuit beter te berijden. De gewichten worden in normale situatie zo aangepast dat de auto sneller zal rijden, terwijl wanneer de auto teveel slipt, of uit de bocht vliegt de lokale gewichten zullen veranderd worden zodat hij op dat stuk voortaan voorzichtiger zal rijden.



Figuur 28: Afstandsgewichten uigelegd over het ciruit. Hoe kleiner het gewicht, hoe verder de AI een afstand inschat en hoe sneller hij dus zal rijden

4.6 Flocking

Flocking is een techniek die het mogelijk maakt op een zeer simpele manier bewegende eenheden in een coherente groep te laten bewegen. Dit wordt gebruikt om bijvoorbeeld de beweging van een groep vogels of een massa mensen te simuleren. Flocking gaat uit van drie simpele regels:

Cohesie elke eenheid stuurt naar de gemiddelde positie van zijn buren in de groep

Gelijklopen elke eenheid stuurt zó dat hij gaat naar de gemiddelde richting van zijn buren

Scheiding elke eenheid stuurt zó dat hij botsingen met buren vermijdt

Samen leveren deze regels (correct geïmplementeerd) realistische simulaties op.

4.6.1 Toepassing

De toepassing van flocking in een racesimulatie lijkt niet onmiddelijk duidelijk, aangezien er geen groepen eenheden zijn die samen moeten bewegen. Toch kunnen de principes zeer nuttig zijn in dit type simulatie. De regels van gelijklopen en scheiding kunnen namelijk gebruikt worden om obstakels en andere auto's op een realistische manier te ontwijken. De derde regel wordt vervangen door de eigen wil van de auto, namelijk het rijden langs het gevonden pad.

Samen zorgt dit ervoor dat auto's hun pad proberen te volgen, maar wanneer ze een andere auto te dicht naderen zullen ze, indien mogelijk, erlangs scheren.

5 Physics

Physics in simulaties zorgen ervoor dat de wetten uit de fysica worden nageleefd. Denk hierbij bijvoorbeeld aan objecten die elk een bepaalde massa hebben, en onderhevig zijn aan de zwaartekracht. De physics zorgen ervoor dat een simulatie realistischer oogt. Vaak zijn de gebruikte physics slechts benaderingen van de werkelijkheid, grotendeels voor performantie redenen.

In een race simulatie zijn physics traditioneel een belangrijk, zo niet het belangrijkste, onderdeel. Ze bepalen namelijk het rijgevoel. In deze simulatie zijn ze echter meer een ondersteunde factor. De complexiteit van de AI moet hier zoveel mogelijk op andere vlakken gecompenseerd worden. Hierdoor zijn de physics bewust vrij eenvoudig, doch functioneel genoeg gehouden.

Voor dit project zijn er verschillende testen geweest met physics engines en voertuig-simulerende code. Uiteindelijk is er gekozen voor de 'Monstrous Software Car Physics'. Deze physics zijn geschreven door een enthousiaste programmeur, gedeeld met de rest van de wereld door middel van een tutorial. De implementatie van deze physics is zeer eenvoudig en de resultaten ervan zijn, mits enkele kleine aanpassingen, bevredigend.

De eenvoud is natuurlijk niet altijd een voordeel, complete physics engines als de Open Dynamics Engine (later in dit hoofdstuk besproken) bevatten bijvoorbeeld standaard collision detection. Nu moest dit dus zelf geïmplementeerd worden.

5.1 PhysicsWorld en PhysicsObject klassen

De PhysicsWorld klasse beheert alle objecten waar bepaalde fysica op moet worden uitgeoefend. Zo'n object moet dan de PhysicsObject klasse implementeren.

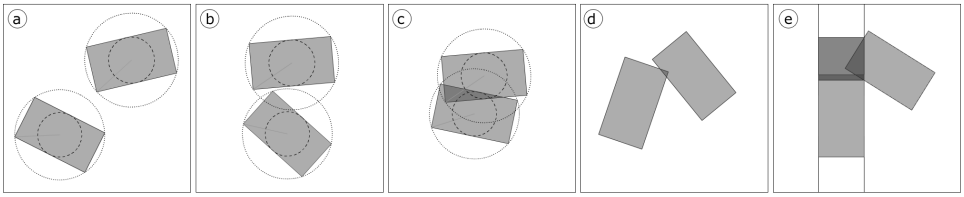
Een PhysicsObject moet verschillende gegevens bevatten. Een twee dimensionale positie, een lengte en een breedte, een rotatie, een massa en een bepaalde acceleratie. Ook moet er een update methode beschikbaar zijn om op te roepen. Hiermee moeten er voor een bepaalde tijd physics op het object worden uitgeoefend.

De PhysicsWorld heeft buiten de methode om objecten toe te voegen, één belangrijke functie: *update*. Hierin worden alle update methodes van de aanwezige objecten opgevraagd, en wordt er collision detection op toegepast. In een normaal programma wordt deze functie dus elk frame opgeroepen.

5.1.1 Collision Detection

De collision detection werkt enkel met rechthoeken, de lengte is hierbij de langste zijde, de breedte de kortste. Het proces gebeurt binnen een lus waarbij elk object getest wordt op aanraking met alle andere objecten. Zo een test bestaat uit twee delen. Eerst wordt de afstand tussen het middelpunt van de twee objecten berekend. Indien deze groter is dan de som van de halve diagonalen van de twee objecten, kan er in geen geval een aanraking zijn (op figuur 29 voorgesteld door de grote cirkels). Indien ze kleiner is dan dan de som van de halve breedtes van de twee objecten (voorgesteld door de kleine cirkels), is er altijd een botsing.

Figuur 29a toont twee objecten waarbij de afstand te groot is om nog te kunnen botsen. Op de volgende figuur (29b) bestaat deze kans wel, de buitenste cirkels overlappen elkaar namelijk. Op figuur 29c is er zeker een botsing, hier overlappen de binnenste cirkels elkaar wel.



Figuur 29: Collision Detection

Wanneer de afstand tussen de twee ligt moet het tweede gedeelte van de collision detection uitsluitsel geven. Figuur 29d toont hier een voorbeeld van.

Eerst en vooral wordt er voor gezorgd dat de eerste rechthoek een hoek van 0 graden beschrijft. Hiervoor moeten zowel de eerste als de tweede rechthoek (met een hoek gelijk aan de negatieve hoek van de eerste rechthoek) gedraaid worden. Hierdoor krijgt rechthoek 1 enkele eigenschappen. Zijn linker- en rechterzijde worden elk apart door dezelfde x-waarden beschreven, en de boven- en onderzijde elk door dezelfde y-waarden.

Er wordt dan onderzocht of de tweede rechthoek het gebied tussen de twee x-waarden, de horizontale zone, raakt. Indien dit niet het geval is, is er al zeker geen sprake van een botsing. Op figuur 29e wordt dit verduidelijkt. De twee doorlopende verticale lijnen bepalen hier de horizontale zone. De donkergrijze zone (raakzone) wordt bepaald door het hoogste en het laagste snijpunt van rechthoek 2 met de horizontale zone.

In het makkelijkste geval zitten zowel het hoogste als het laagste punt van de tweede rechthoek in de horizontale zone. Dan wordt de raakzone enkel door deze twee waarden mee bepaald. In andere gevallen moet er getest worden

of de punten uiterst links en rechts van rechthoek 2 voor een overlapping met de horizontale zone kunnen zorgen. Dan moet via snijpunten de raakzone bepaald worden.

Hierna is het eenvoudig te bepalen of de twee rechthoeken elkaar raken. Dit is namelijk het geval wanneer de raakzone en de eerste rechthoek elkaar raken. Er moet dus enkel gezocht worden naar een overlapping bij de y-waarden door het feit dat de zijden van de zone steeds loodrecht op de y-as staan.

Wanneer er een botsing gedetecteerd wordt, is het natuurlijk belangrijk dat er ook een reactie op volgt. Ook hier is de oplossing eenvoudig gehouden. Aan de hand van de acceleratie en de massa van een object wordt de potentiële kracht berekend die dat object in zijn bewegingsrichting uitoefend. Normaal gezien moet dan de richting van acceleratie van de twee objecten in rekening worden gebracht, en speelt de factor waar de objecten elkaar raken een belangrijk rol.

Hier wordt de berekening echter sterk vereenvoudigd. Ten eerste wordt er geen rekening gehouden met het verschil in richting van acceleratie. Wanneer er een botsing voorvalt wordt via de middelpunten van de objecten berekent onder welke hoek er gebotst wordt. De kracht zelf wordt een stuk verkleind. Ze gaat er immers van uit dat het andere object zich in de richting van het andere object voortbeweegt. Dit is echter meestal niet het geval.

Aan de hand van de hoek en de overblijvende kracht wordt dan een acceleratie bepaald. Deze wordt dan bij de volgende update van het physics object in rekening gebracht met de werkelijke acceleratie van het object.

5.2 CarPhysics klasse

De uiteindelijke physics zijn gebaseerd op de car physics tutorial[15] van 'Marcus Monstrous'. Deze physics zijn gekozen omdat de code die zorgt voor de uitvoering vrij compact is, en het resultaat toch zeer realistisch oogt. De compactheid van de code heeft tot gevolg dat niet alle facetten van de fysica geïmplementeerd zijn, maar voor dit project toch al meer dan voldoende. Dit hoofdstuk geeft een korte schets hoe de physics in elkaar zitten.

De physics engine steunt op twee soorten krachten die helpen om tot een geslaagd eindresultaat te komen: longitudinale en laterale krachten. Longitudinale krachten zijn actief in de lengte, denk hierbij bijvoorbeeld aan

[15] http://web.archive.org/web/20040803175849/home.planet.nl/~monstrous/tutcar.html

accelereren en remmen. Laterale of zijdelingse krachten zorgen dan weer voor het realistisch draaien.

De physics zijn gemaakt voor auto's met achterwielaandrijving.

Wanneer een auto in een rechte lijn vooruit rijdt, treden de longitudinale krachten in werking. Een eerste kracht hierbij is logischerwijs de kracht waarmee de achterste wielen draaien. Deze kracht zorgt er namelijk voor dat de auto vooruit zal gaan. Belangrijk is ook de resistentie die de lucht veroorzaakt. Naarmate een auto sneller gaat rijden zal deze een belangrijkere rol spelen. Een andere vorm van resistentie is die tussen het rubber van de band, en de baanoppervlakte.

Wanneer we al deze krachten willen optellen is het belangrijk te weten dat wanneer de beweging veroorzaakt door de wielkracht positief is, de resistentiekrachten negatief zijn.

Nadat de totale kracht is berekend kan de acceleratie bepaald worden. De tweede wet van Newton zegt namelijk het volgende: $F = m * a$. Waarbij F de kracht is, m de massa en a de acceleratie. Indien deze formule hervormd wordt, is dit het resultaat: $a = \frac{F}{m}$. De acceleratie wordt dus bepaald door de berekende kracht te delen door het gewicht van de auto.

Wanneer er gedraaid moet worden, wordt het rekenen al wat ingewikkelder. Eerst en vooral is er een verschil tussen draaien tegen lage snelheid en tegen hoge snelheid. Bij lage snelheid gaat de auto zonder probleem in de richting naar waar de wielen wijzen. Aan de hand van de hoek van de voorwielen ten opzichte van de auto is het vrij simpel om de gehele draaicirkel te tekenen. Wanneer het project nu bijvoorbeeld een parkeer-simulatie was geweest had deze werkwijze volstaan.

Bij hogere snelheden kan er echter niet vanuit gegaan worden dat de richting van de auto enkel en alleen bepaald wordt door de richting van de wielen. Een makkelijk in te beelden voorbeeld is de wijze waarop een rally coureur zijn wagen in een bocht 'gooit'. De wagen ligt dan schuin ten opzichte van de richting waar hij in werkelijkheid naartoe gaat. Om zo een situatie te verkrijgen moet de kracht die de auto uitoefent groter zijn dan de maximale grip die de banden bieden.

Een probleem bij deze (en andere) physics is het feit dat de auto nog zeer lang blijft rijden wanneer de gaspedaal gelost wordt. Hiervoor wordt er, wanneer er geen gas gegeven wordt, aan de physics gezegd dat er zacht geremd wordt. Ook de kracht van het remmen zelf is vergroot.

Andere aanpassingen aan de code dienen voor het rekening houden met de ondergrond. Deze bepaalt namelijk hoeveel grip en weerstand er wordt geboden door het baanoppervlak.

5.3 Andere geprobeerde engines

5.3.1 Open Dynamics Engine

De Open Dynamics Engine of ODE is een krachtige real-time physics engine. Ze is open source en platformonafhankelijk, geschreven voor C/C++. Deze engine levert zeer goede rigid body physics[16] en collision detection[17]. Ze is dan ook zeer geschikt om uiterst realistische simulaties met voertuigen te schrijven.

Om deze redenen was ODE de eerste keuze om de physics in de racesimulatie te leveren.

Bij het integreren van de engine werd echter duidelijk dat met de kracht van de engine een zekere complexiteit bij het programmeren meekomt. Simpele acties als het rondbewegen van een kubus in de 3D physics wereld zijn relatief gemakkelijk te schrijven, maar het modelleren van een auto met wielen die weerstand ondervinden van de ondergrond en de lucht én die realistisch botst, is reeds een minder vanzelfsprekende opgave. De leercurve voor het programmeren met deze engine werd te hoog bevonden voor de tijd die ons eindwerk bestrijkt en daarmee had de engine ook afgedaan voor de simulatie.

5.3.2 2D Car Physics by Matt Kincaid

Na de moeilijkheden met een uitgebreide physics engine als ODE, is er gezocht naar eenvoudigere alternatieven. Een eerste interessante oplossing was te vinden in het 2D Car Physics artikel van Matt Kincaid, geprogrammeerd in C#. Veelbelovend aan deze brok code waren de afzonderlijke werking van de vier wielen. (In de huidige engine is er ruim gezien enkel een voor- en een achterkant.). Hierdoor trekt de auto bijvoorbeeld automatisch naar rechts wanneer de rechterkant zich juist in gras bevindt, omdat de weerstand daar hoger is en er dus minder snelheid ontwikkeld kan worden. De uiteindelijke werking stelde echter teleur. Zelfs na veel sleutelen aan verschillende zaken als motorkracht, grip en weerstand bleven de auto's te veel 'glijden' over het circuit.

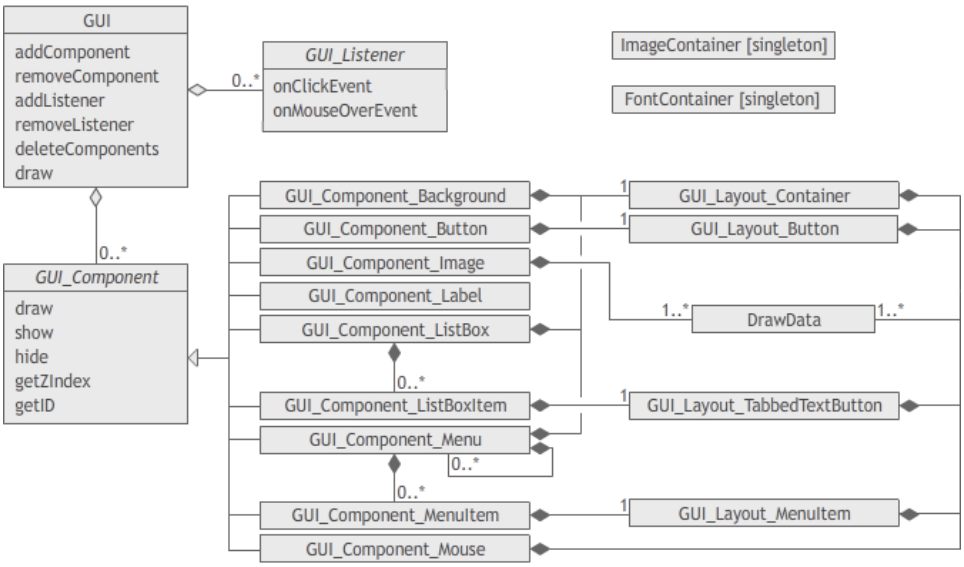
Enig voordeel aan het werken met deze engine was het feit dat er reeds een vector klasse voor gemaakt moest worden, die dan later hergebruikt werd door de uiteindelijke physics. Deze vector klasse is in principe niet meer dan de x en y datamembers. Bijkomend zijn er ook wiskundige functies om vectoren op te tellen, af te trekken, vermenigvuldigen, ...

[16] Bewegingen van vaste objecten
[17] Detecteren van collisies

6 Grafische gebruikersomgeving

6.1 Werking

Gosu bevat geen standaard GUI[18] klassen. Hierdoor moest alle interactiviteit zelf geprogrammeerd worden. Er is dan maar meteen besloten om te werken aan een bijna volwaardig GUI framework, volledig met componenten als menu's, listboxen, labels en knoppen. Een GUI object bevat verschillende GUI elementen en een lijst van listeners die worden verwittigd wanneer er bijvoorbeeld op een bepaald element geklikt wordt.



Figuur 30: UML klassendiagram van de GUI

Het is vanzelfsprekend dat een GUI object mogelijkheden biedt om elementen en listeners toe te voegen en te verwijderen.

De belangrijkste functie is echter de draw methode. Wanneer de GUI in zijn geheel zichtbaar moet zijn, is het nodig dat op elk frame deze methode wordt aangeroepen.

Ze rangschikt eerst alle componenten volgens z-index. Hoe groter deze index, hoe hoger het component in de lijst zal verschijnen. Hierna gaat de methode de lijst af en tekent elk element.

Zoals straks zal blijken heeft de draw functie van elk component twee argumenten nodig en heeft het een string als return type. Het eerste argument is

[18] Graphical User Interface of Grafische gebruikersomgeving. Dit is het geheel van knoppen, tekstvelden, vensters, ... die interactie met de gebruiker vergemakkelijken.

een input object, en bevat dus informatie als de positie van de muis en ingedrukte knoppen op het toetsenbord. Het tweede argument is een booleaanse waarde die zegt of het element in kwestie actief is. De string die teruggegeven wordt door de draw functie van het component is ofwel leeg, ofwel bevat hij het ID van het element waar op dat moment de muis boven hangt. Dit is dus het ID van het element zelf, of in bijvoorbeeld menustructuren kan het dat van een submenu zijn.

Aan het begin van de draw methode van het GUI object zullen de elementen altijd actief getekend worden. Het tweede argument van elke element draw methode is dus *true*. Dit gebeurt tot een draw methode van zo'n element een ID teruggeeft. Vanaf dan weet de GUI dat dit het (in de diepte) hoogste element is waar de muis zich op bevindt. De elementen eronder kunnen dus nooit nog actief worden.

6.2 GUI Listeners

GUI Listeners zijn klassen die GUI_Listener overerven. Hun functie is te 'luisteren' naar de GUI. Dit wilt zeggen dat wanneer er bijvoorbeeld op een knop geklikt wordt, de listener hiervan op de hoogte zal gebracht worden. De werking hiervan is gebaseerd op het observer design pattern. De GUI is hier het subject en de listeners zijn de observers.

Listeners zijn verplicht twee methodes te implementeren, namelijk *onClickEvent* en *onMouseOverEvent*. Beide methodes hebben een ID-string argument dat slaat op het GUI element waarover het gaat. De eerste methode zal worden opgeroepen in het geval dat de gebruiker zijn linker muistoets heeft losgelaten op een bepaald element, de laatste altijd wanneer de muis zich op een bepaald GUI element bevindt.

6.3 GUI Componenten

De elementen die de GUI voorstellen erven elk de klasse GUI_Component over. Deze interface zorgt ervoor dat de GUI klasse een lijst kan bijhouden met daarin verschillende typen GUI componenten, die allemaal gelijksoortige basisfunctionaliteit hebben.

De *draw* methode van een component tekent een element op het scherm, *show* en *hide* zorgen voor het weergeven en verstoppen van een element. Het is ook belangrijk dat elk element een z-index heeft, dit bepaald namelijk de diepte van het element. Een element met een hogere z-index wordt steeds getoond boven een element met een lagere z-index. Elk element heeft ook een ID, dit is meestal een korte omschrijving van het type element en zijn

functionaliteit. Aan de hand van een ID kunnen GUI listeners opmaken welke reactie er moet zijn op bepaalde events.

De draw methode van een component heeft twee argumenten, namelijk een input object en een booleaanse waarde die duidt op het feit of het component actief is of niet.

Het input object is een belangrijk Gosu element en bevat informatie over verschillende input apparaten als muis, toetsenbord en spelcontroller. Aan de hand van zo een object kan dus bijvoorbeeld de huidige positie van de muis bepaald worden.

De waarde die duidt op het actief zijn van het component is niet bepalend voor het eigenlijke actief zijn. Ze bepaalt enkel of het component actief mág zijn. Een element is meestal pas echt actief als dit argument *true* is én bijvoorbeeld de positie van de muis binnen de rand van het element ligt. In dit laatste geval zal de methode de ID van het element teruggeven, om er op te wijzen dat dit element dus actief is. In het andere geval stuurt deze functie gewoon een lege string terug.

6.4 De verschillende GUI layouts

Veel componenten maken gebruik van een bepaald soort layout. Deze bepaalt de algemene vormgeving van het element. Terugkomend bij layouts is een soort van initialisatie methode. Hierin worden aan de hand van bijvoorbeeld tekstuele inhoud de grootte en posities van gebruikte afbeeldingen en tekst berekend. Het gebruik van deze functie zorgt er dus voor dat deze posities niet elk frame opnieuw moeten berekend worden, wat zeer goed is voor de performantie.

Afbeeldingsinformatie wordt opgeslagen in een DrawData object. Zo een object bevat dan x, y, z waarden, grootte, bestandsnaam van de afbeelding en een referentie naar de Gosu-image zelf. Eenvoudige GUI elementen maken geen gebruik van een layout, maar gebruiken zelf een DrawData object.

Omdat bepaalde afbeeldingen meermaals gebruikt zullen worden is er een ImageContainer klasse geschreven. Deze singleton klasse[19] bevat alle afbeeldingen maximum één keer. De afbeelding in een DrawData object komt van het singleton ImageContainer object. De container kan worden leeggemaakt op elk gewenst moment, indien dit gewenst zou zijn. Ook de afbeeldingen van bijvoorbeeld de racewagens worden via de ImageContainer ingeladen.

[19]Het singleton design patroon zorgt ervoor dat er maar één instantie van dit object kan gemaakt worden. Het object is daardoor overal hetzelfde, en overal aan te roepen.

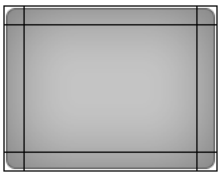
Naast de ImageContainer is er ook nog een FontContainer, deze heeft hetzelfde doel als de ImageContainer, maar dan voor het inladen van fonts. Dit verschilt namelijk per font en fontgrootte.

De layout bevat de laatste draw methode (na de GUI en de componenten). Hier wordt meestal pas echt berekend of de muis binnen de rand van het element valt en of het element dus actief is (indien het actief-argument niet $false$ is natuurlijk).

Wanneer een element getekend wordt, verschilt dit proces in veel gevallen van de status van het element. Er zijn over het algemeen drie verschillende statussen. Namelijk *mouseOut* wanneer de muis zich buiten het element bevindt, *mouseOver* wanneer de muis zich binnen het element bevindt, en *mouseDown* wanneer de muis zich binnen het element bevindt én ook de linkermuistoets ingedrukt is.

6.4.1 Container layout

De container layout is gemaakt voor elementen die kunnen verschillen in grootte van inhoud, maar allen dezelfde rand willen behouden zonder vervorming. Hij maakt gebruik van negen afbeeldingen (zie figuur 31). De afbeeldingen op de hoeken behouden altijd hun originele grootte. De andere afbeeldingen worden naargelang de inhoud uitgerokken of verkleind.



Figuur 31: Container layout

6.4.2 Button layout

De button layout is vergelijkbaar met de container layout qua indeling. Hij ondersteunt echter de drie verschillende statussen (*mouseOut*, *mouseOver* en *mouseDown*). Hierdoor wordt het aantal benodigde afbeeldingen met drie vermenigvuldigd. Elke status gaat namelijk gepaard met een nieuw uiterlijk, het moet immers duidelijk zijn dat een gebruiker boven een bepaalde knop hangt.

Naast de afbeeldingen bevat de layout ook een tekst object. Per status kan de kleur van de tekst veranderd worden. Net zoals vele andere teksten zal er een soort schaduw onder de tekst hangen. Dit is in feite niet meer

dan dezelfde tekst, maar dan in het zwart, die een pixel naar rechts en naar beneden is opgeschoven. De kleur van de schaduw kan niet veranderd worden.

6.4.3 Menu item layout

De menu item layout is eigenlijk een versimpelde knop. In plaats van negen afbeeldingen per status zijn het er slechts drie. Deze keuze is gemaakt omdat menu items allemaal dezelfde hoogte hebben.

Figuur 32: Menu Item layout

6.4.4 Tabbed text button layout

De tabbed text button layout tenslotte is een nog meer versimpelde vorm van een knop. Hij gebruikt per status slechts één afbeelding. Bijkomend is wel de tab-functionaliteit voor de tekst. Deze zorgt ervoor dat stukken tekst op een bepaalde horizontale positie kunnen worden geplaatst. Eigenlijk wordt er een soort van tabel aangemaakt met één rij en verschillende kolommen. Elke kolom krijgt een bepaalde grootte. Wanneer de inhoud (tekst) van deze kolom te groot wordt, wordt deze automatisch korter gemaakt en worden er aan het einde van de verkorte tekst twee puntjes toegevoegd.

Figuur 33: Tabbed Text Button layout

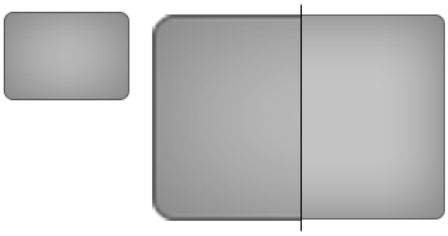
6.5 De verschillende GUI componenten

6.5.1 Image component

De image component is de eenvoudigste GUI component. Hij omvat namelijk slechts één afbeelding. De positie en de grootte van deze afbeelding zijn in te stellen en worden intern voorgesteld door een DrawData object.

6.5.2 Background component

De background component is eigenlijk niet meer dan een rechtstreekse vertaling van de container layout en kan, zoals de naam doet vermoeden, dienst doen als achtergrond. Het gebruik van deze klasse biedt ten opzichte van het image component vooral een voordeel bij herschaling. De 'border-afbeeldingen' gaan namelijk niet mee geschaald worden, waardoor de rand van het totaalbeeld steeds even scherp blijft.



Figuur 34: Verschil in herschaling bij Image en Background GUI Components

De figuur hierboven maakt dit duidelijk. De kleine rechthoek links is de originele achtergrond. De rechthoek rechts toont het verschil bij vergroting bij gebruik van een Image (links) en een Background (rechts).

6.5.3 Button component

De button component biedt de meest complete oplossing wanneer er een knop moet worden weergegeven. Logischerwijs gebruikt deze component de button layout. Buiten de verschillende afbeeldingen die de knop definiëren in elke fase (mouseOut, mouseOver en mouseDown) zijn er ook nog andere factoren aanpasbaar. Hieronder valt de tekst van de knop en de padding van de tekst (hoeveel de tekst van elke rand moet staan).

6.5.4 Label component

De label component bevat gewoon een tekst. Buiten de mogelijkheid om de te tonen tekst te veranderen, in te kleuren en positioneren biedt hij ook de optie om schaduw te tonen. Ook aan de schaduw is een andere kleur te geven.

6.5.5 Listbox item component

Een listbox item is zoals de naam voorspelt, gemaakt voor gebruik in een listbox. Al kan hij natuurlijk ook als simpele knop apart genomen worden. Zijn layout is de tabbed text button layout. Hiermee wordt het werken met verschillende kolommen in een listbox tegemoet gekomen.

Bij het creëren van een listbox item component moet het aantal kolommen ingesteld worden. Om de grootte van de kolommen te bepalen moet er een array van integers meegegeven worden. Deze array heeft dezelfde lengte als het aantal kolommen en bevat telkens de horizontale positie die het einde van de kolom bepaalt. Er wordt dus vanuit gegaan dat de eerste kolom start op 0. Door het gebruik van padding in de tabbed text button layout kan dit natuurlijk visueel gewijzigd worden.

6.5.6 Listbox component

De listbox component is in essentie een container die een hoopje listbox items bevat. Bij zijn creatie wordt het aantal kolommen vastgelegd, en ook de kolomafmetingen.

De listbox bevat een methode om eenvoudig listbox items toe te voegen. Er moet dan (buiten het ID) enkel een array van strings worden meegegeven, met de grootte van de array gelijk aan het aantal kolommen. Aan de hand hiervan en de reeds bekende gegevens als kolommen kan de listbox een nieuw item toevoegen. Deze methode stuurt de aangemaakte listbox item component terug zodat bijvoorbeeld de layout van het item kan worden gewijzigd.

Een soortgelijke functie bestaat er voor de titel rij van de listbox. Deze rij is op zich ook een listbox item, kenmerkend hiervoor is het feit dat hij altijd zichtbaar, bovenaan de lijst staat.

Er is ook functionaliteit voorzien om te scrollen in de listbox. Dit gebeurt niet aan de hand van een schuifbalk, maar moet anders geïmplementeerd worden. Er moet dus een knop gemaakt worden die dan de *setStartItem* methode van de listbox aanroept. Hieraan moet het nummer van de rij die bovenaan moet staan meegegeven worden. (Er is ook een methode *getStartItem* waardoor het niet moeilijk is een item naar boven of naar onder te scrollen.)

6.5.7 Menu item component

De menu item component is een zeer simpele button die steunt op de menu item layout. Net als de listbox item component is hij apart te gebruiken, maar speciaal aangemaakt voor een speciale houder. Hier is dat de menu component.

6.5.8 Menu component

Een menu component is visueel niets meer dan een container layout met daarin verschillende menu items. Het gebruik van dit component gaat evenwel iets dieper dan dat.

Met de *addMenuItem* methode kan er een nieuw menu item aan het menu toegevoegd worden. Visueel wordt er dus een knop in een container geplaatst. De uitlijning van verschillende menu items in de container hangt af van de menu item layout. Wanneer bijvoorbeeld alle menu items linksonder uitgelijnd worden, zullen ze onder elkaar staan. Wanneer ze rechtsboven uitgelijnd zijn, zullen ze horizontaal naast elkaar staan.

Het wordt echter pas interessant als er gewerkt wordt met submenu's en subsubmenu's. Hiervoor dient de methode *addSubMenu*. Het is noodzakelijk een submenu te binden aan een menu item van het menu. Als het menu item actief is, zal het bijbehorende submenu getoond worden. Een menu item is actief tot een ander menu item uit hetzelfde menu actief wordt, of er naast alle submenu's geklikt wordt. Het is actief vanaf de muis over het menu item zweeft (er moet dus niet eerst geklikt worden op het item). Ook bij de submenu's is uitlijning van de layout belangrijk voor de plaatsing. Een submenu is op zijn beurt een menu component dat verschillende items en submenu's kan hebben.

Omdat natuurlijk altijd minstens het basis menu zichtbaar moet zijn, is dit aan te geven via de *setAlwaysVisible* methode

6.5.9 Mouse component

De mouse component zorgt voor de visuele aanduiding van de huidige positie van de muis. Buiten het veranderen van de afbeelding is er niet veel aan dit component aan te passen.

7 Globale werking van de applicatie

Wanneer de applicatie in werking is, is het belangrijk dat de verschillende statussen van het programma gescheiden worden. De twee belangrijkste statussen van het programma zijn de menu status en de race status. Tijdens de menu status wordt het hoofdmenu getoond, de code die moet uitgevoerd worden is miniem en dienen vooral om interactie met de gebruiker te verzorgen. Tijdens de race status worden logischerwijs enkel berekeningen gemaakt in verband met de race, en zal deze ook op het scherm getekend worden.

De GameWindow klasse erft over van de Window klasse van Gosu. Deze werd eerder al beschreven in hoofdstuk 2.2.1. De status van het programma bepaalt welke functionaliteit er wordt uitgevoerd in zowel de update als de draw methodes.

De eerste status van het programma is de intro status. Tijdens de update functie wordt berekend waar de verschillende afbeeldingen moeten komen te staan aan de hand van het aantal frames sinds de start van het programma. De intro bestaat namelijk gewoon uit enkele afbeeldingen die van positie en/of transparantie veranderen. Tijdens de draw functie worden deze afbeeldingen dan op het scherm getekend.

Wanneer de intro is afgelopen wordt de status gezet op deze waar het hoofdmenu getoond wordt. Bij het toekennen van deze status is het nodig de verschillende GUI elementen toe te tevoegen aan het GUI object. Tijdens de draw functie wordt dit GUI object dan getekend. Tijdens de update functie gebeurt er hier niets. De functionaliteit nodig voor het reageren op invoer van de gebruiker wordt opgevangen doordat het GameWindow luistert naar GUI events.

De onMouseClick listener zorgt onder andere voor het toevoegen van nieuwe GUI elementen wanneer er een ciruit wordt gekozen, start andere programma's op wanneer er bijvoorbeeld wordt gekozen om een circuit aan te passen. Dit laatste gebeurt met de hulp van het ShellExecute commando. Van hieruit worden ook enkele andere statussen van het programma aangeroepen.

Een eerste andere status is deze waarbij er een achtergrond gegenereerd wordt voor een circuit. Hier wordt in de update functie steeds een stap van de generatie uitgevoerd. In de draw functie wordt dan de gebruikelijke GUI getekend, met daarbovenop een eenvoudige afbeelding en een tekstuele voorstelling van de huidige generatiefase.

De andere statussen hebben elk te maken met het racen en het laden van die race. Voor het starten van een race wordt een RaceData object aangemaakt. De draw functie toont vanaf dit moment het laadscherm, dit is niet meer

dan een eenvoudige afbeelding. Tijdens het volgende frame wordt nu de loadRace methode van het RaceData object opgeroepen, en wordt de race dus ingeladen. Meer informatie over de werking van de RaceData klasse is te vinden in het volgende hoofdstuk.

Wanneer de loadRace functie voltooid is komt het programma in de race status. De update functie zal bestaan uit een simpele aanroep van de updateRace methode van het RaceData object. De draw functie stelt het met een eenvoudige aanroep van de drawRace methode.

8 Verloop van een race

8.1 Algemene werking

De RaceData klasse zorgt voor het in goede banen leiden van een race. Ze bevat vier belangrijke functies: *loadRace*, *updateRace*, *drawRace* en *endRace*.

LoadRace zorgt voor het inladen van een race. Eerst wordt het gekozen circuit via de Level klasse volledig ingeladen. Hierna gebeurt de initialisatie van verschillende variabelen, waaronder de verschillende arrays die de verschillende rondetijden zullen opslaan. De achtergrondafbeelding wordt ingeladen en ook de verschillende deelnemers worden geïnitialiseerd. Voor elke auto wordt er hierna een PhysicsObject toegevoegd aan een PhysicsWorld object. Ten laatste worden de AI Driver instanties geïnitialiseerd. Deze krijgen een brein onder de vorm van een neuraal netwerk en dat wordt onmiddellijk getraind. Daarenboven berekent elke Driver zijn initiëel pad over het circuit.

UpdateRace zorgt voor alle berekeningen. De physics wereld wordt geupdated, er wordt berekend of er auto's een ronde hebben afgelegd, bandensporen worden berekend en input van de gebruiker wordt opgevangen. Daarnaast wordt de AI gevraagd de auto's te besturen en om eventueel hun paden te herberekenen.

DrawRace tekent de race op het beeldscherm. Hieronder vallen de achtergrond, de verschillende auto's, bandensporen, tekstuele informatie en eventueel een GUI object (het pauzemenu).

EndRace zorgt tenslotte voor het correct verwijderen van alle informatie.

8.2 Visuele aspecten

8.2.1 Basis

In principe zijn er niet bijster veel visuele aspecten aan een race, zeker niet wanneer ze wordt voorgesteld aan de hand van twee dimensionale graphics. De essentiële onderdelen zijn hier dan de achtergrond en de verschillende auto's. Deze worden elk frame van de race getekent op hun respectievelijke positities. Omdat ze worden voorgesteld door statische afbeeldingen valt er verder niet veel over te zeggen. Het zijn elk aparte Gosu Bitmaps, die aan het begin van de race worden ingeladen. De auto die bekeken wordt, wordt altijd centraal weergegeven. De achtergrond en de andere auto's worden gelijkwaardig verschoven.

8.2.2 Tekstuele informatie

Een ander aspect is de aanvullende informatie. Hieronder vallen de zaken die belangrijk zijn voor de auto waar de camera op gefixeerd is. Het gaat hier dan over de huidige snelheid, rondetijden, het aantal afgelegde ronden en de naam van de rijder. Al deze informatie wordt afgebeeld via Gosu Fonts, die worden gehaald uit de FontContainer klasse.

8.2.3 Pauzemenu

Wanneer de race gepauzeerd wordt of afgelopen is, wordt het pauzemenu getoond. Dit menu is een GUI object met daarin een listbox en drie knoppen. De listbox bevat informatie over de stand, de knoppen bieden de mogelijkheid te scrollen door het deelnemersveld of de race te beëindigen.

De werking van de knoppen is eenvoudig. De 'scroll up' knop zet de start positie van de listbox eentje lager in waarde, de 'scroll down' knop doet het omgekeerde. De knop om de race te verlaten roept de endRace methode op.

De listbox bevat standaard evenveel items als er deelnemers zijn. Wanneer het pauzemenu wordt aangeroepen, wordt de listbox aan de hand van de huidige stand van de race opnieuw opgebouwd. Opbouwen is hier een groot woord, aangezien enkel de tekstuele inhoud van de verschillende items verandert. Het aanpassen van deze informatie gebeurt aan de hand van een array die de posities bevat van de deelnemers.

8.2.4 Bandensporen

De bandensporen maken het visuele geheel al wat interessanter. Enerzijds is het een visuele verbinding tussen de auto's en het baanoppervlak. Anderzijds kan het gebruikt worden om de rijlijn van de artificiële intelligentie te bestuderen.

Bandensporen worden op elk frame, voor elke band van een auto gecreëerd. Afhankelijk van de snelheid, het feit of de rempedaal wordt ingedrukt en of de auto al dan niet aan het slippen is wordt de 'afdruk' donkerder of lichter. Het bepalen of een wagen al dan niet aan het slippen is, kan vrij eenvoudig worden verwezenlijkt. Het komt er op neer de richting waarin de wagen beweegt, te vergelijken met de richting naar waar de auto wijst. Naarmate dit verschil groter wordt, slipt de wagen harder, en zullen de sporen donkerder worden. Ook naarmate de snelheid van een auto groter is terwijl hij remt, zal de afdruk verdonkeren.

Om de bandensporen voor te stellen moest er dus een dynamische manier gevonden worden om een afbeelding te wijzigen. De Image klasse biedt deze functionaliteit niet, dus is er gekozen voor het gebruik van Bitmap objecten. Zoals gezegd in hoofdstuk 2.2.5 duurt het echter zeer lang om een Bitmap om te zetten naar een toonbare Image. Om dit te verhelpen worden er verschillende, kleinere, Bitmaps en Images gemaakt, en wordt er slechts één Bitmap per frame omgezet naar een Image. Er werd gekozen om een raster te vormen met daarin afbeeldingen van 100x100 pixels. De afbeelding die gevormd moet worden hangt elk frame van een andere auto af. Het gaat dan namelijk om die afbeelding waar het middelpunt van de wagen zich op dat moment bevindt.

Alvorens er afbeeldingen gevormd kunnen worden met bandensporen, is het natuurlijk noodzakelijk dat de verschillende Bitmaps aangevuld worden met nieuwe data. Op elk frame moet daarom een hele waslijst aan bewerkingen uitgevoerd worden. Er moeten namelijk voor elke band van elke auto sporen worden toegevoegd aan de Bitmaps. Voor elke auto wordt eerst de donkerte van het gewicht bepaald, op de hierboven beschreven wijze. Daarna wordt de ligging van de banden bepaald. Voor elke band wordt de passende Bitmap gezocht, en de relatieve positie van de band tenopzichte van deze Bitmap. Op deze positie wordt dan een klein bolletje bij getekend. De transparantie van het bolletje hangt af van de donkerte die daarvoor berekend is. Dit bolletje is niet meer dan een centraal punt, dat de werkelijke donkerte heeft, met daarrond een vierkante meer transparante pixels.

Het toevoegen van de extra kleur gaat vrij eenvoudig. Standaard wordt de kleur van de afbeeldingen op zwart gezet, met een alpha waarde van nul. De afbeeldingen zijn dus 100% transparant. Wanneer de kleur donkerder moet worden, wordt gewoon de alpha waarde een beetje verhoogd. Er wordt ook een maximale alpha waarde bepaald. Hierdoor wordt voorkomen dat er volledig zwarte stroken op het circuit komen te liggen.

De oplettende lezer zal nu de opmerking maken dat wanneer de positie van de band samenvalt met de rand van de Bitmap, niet het hele bolletje getekend kan worden. Hiervoor, en ook voor een andere reden, die zo meteen wordt aangekaart, worden de groottes van de Bitmaps en Images zowel in de breedte als in de lengte twee pixels vergroot. Er wordt met andere woorden een rand met een breedte van één pixel toegevoegd. De overige code blijft echter nog steeds van de vorige resolutie uitgaan.

De andere reden waarom de afbeelding groter moesten worden gemaakt, is omdat bij de weergave van de afbeeldingen soms problemen voorkwamen. Er ontstonden namelijk klein ruimtes tussen de verschillende afbeeldingen. Dit kwam voor omdat de positie van de achtergrond en afbeeldingen met bandensporen niet altijd wordt gevormd door mooie, ronde getallen.

Natuurlijk moet er nu wel voor gezorgd worden dat de randen niet opvallen doordat daar twee afbeeldingen over elkaar worden weergegeven. De maximum alpha waarde van de pixels aan de rand moet dus de helft zijn van de rest van de afbeelding.

Een laatste, belangrijke, stap in verband met de bandensporen gebeurt in de drawRace methode. Hier wordt namelijk bepaald welke Images werkelijk zichtbaar zijn. Enkel deze worden namelijk getekend. Ook dit gebeurt voor performantie redenen. Het aantal afbeeldingen loopt bij een groot circuit namelijk snel op, en hoe minder afbeeldingen er getoond moeten worden hoe beter natuurlijk.

8.2.5 Extra Informatie

Andere informatie is in of uit te schakelen door de gebruiker. Deze informatie is vooral bedoeld om de handelingen van de artificiële intelligentie beter te leren begrijpen. Aan de hand van tekst worden de verschillende gewichten, nodig voor het inschatten van afstand en snelheid weergegeven. (Meer informatie over deze gewichten is te vinden in hoofdstuk 4.5 op pagina 31.)

Verder worden aan de hand van enkele indicators de verschillende invoerwaarden van de bestuurder aan de auto getoond. Het gaat hier dus over hoe diep de gas- en rempedalen worden ingedrukt, en naar waar het stuur gedraaid wordt.

8.3 Achterliggende code

Voordat er rondetijden getoond kunnen worden, moeten ze natuurlijk op één of andere manier verkregen worden. Om te voorkomen dat er in grote mate vals gespeeld kan worden, wordt er automatisch een controlepunt in de helft van het circuit geplaatst. Een ronde begint en eindigt dus bij de finishlijn, met de voorwaarde dat er onderweg via het controlepunt gereden is.

Het opslaan van rondetijden gebeurt in een twee dimensionale array. In deze array wordt voor alle wagens per ronde de rondetijd opgeslagen. Ook is er een array die van elke auto zijn huidige ronde bijhoudt. Elk frame wordt er aan de rondetijd van de huidige ronde van een auto een tijd opgeteld van 1/60 (de framerate).

In de loadRace methode wordt de positie van de start- en controlelijn bepaald. Deze lijnen zijn in principe niet meer dan de breedtelijnen op het eerste en middelste trackpoint. Om te voorkomen dat wanneer een wagen,

die zich juist een beetje naast de baan bevindt, de lijn niet echt overschreidt, worden de punten geplaatst op een afstand die gelijk is aan het dubbel van de breedte van dat trackpoint.

Wanneer een wagen over de finishlijn rijdt, wordt er enkel een nieuwe ronde gestart wanneer tijdens die ronde de controlelijn reeds gepasseerd is. Er wordt namelijk maar één lijn gecontroleerd per auto. Bij het begin van de race zal de te controleren lijn steeds de controlelijn zijn. Elk frame van de race zal nu getest worden of de wagen de lijn voorbijgegaan is. Dit gebeurt door na te kijken of twee lijnstukken snijden. Deze lijnstukken zijn enerzijds de controlelijn en anderzijds het lijnstuk gevormd door de vorige en de huidige positie van de wagen. Wanneer de lijnstukken snijden, wordt er vanaf het volgende frame getest op een snijpunt met de finishlijn.

Wanneer een auto over de finishlijn rijdt zal logischerwijs zijn huidige ronde met één verhogen.

9 Achtergrond generatie

9.1 Algemene werking

Omdat in de ideale wereld natuurlijk geen werk besteed moet worden aan het maken van een mooie achtergrond voor een circuit, is het de gebruiker gemakkelijk gemaakt doordat het programma zelf een achtergrond kan genereren. Dit proces gebeurt volledig proceduraal, er worden dus geen bestaande afbeeldingen gebruikt om te komen tot een resultaat.

Een acceptabel resultaat wordt verkregen door een opeenvolging van enkele stappen.

De eerste stap is de rasterizatie van het circuit. Deze stap geeft ons een raster met hierop de circuit en grasgedeeltes.

Hierna worden aan de hand van deze gegevens nieuwe zones gecreëerd. Deze zones omvatten de grindbakken[20] en plekken op het gras die ver genoeg van het circuit liggen om er bomen op te zetten. De laatste zones zijn de verschillende wegmarkeringen voor startposities en de finishlijn.

Vervolgens wordt een laag gecreëerd met een wolkenpatroon. Deze heeft niet als doel echte wolken te maken, maar wordt later wel gebruikt om een laag schaduw over het circuit te leggen, die doet denken aan de schaduw van bewolking.

Het uiteindelijke tekenen van het circuit gebeurt in de volgende stap. Hier worden alle vakjes van het raster afgegaan en ingekleurd naarmate de zone waar het vakje zich in bevindt. In de zone waar bomen komen te staan wordt voorlopig nog gras getekend.

De bomen worden namelijk achteraf toegevoegd. Op gedeeltelijk willekeurige basis worden vakjes afgegaan op het raster. Indien de zone van het vakje voor bomen bedoeld was, wordt er een boom gegenereerd en op die plaats in het gras geplakt.

Deze stappen worden verder in dit hoofdstuk nauwkeuriger besproken. Voordat al die informatie losgelaten wordt, worden eerst de belangrijkste technieken in verband met dit proces besproken.

[20] Bij circuits wordt dit soort ruw zand gebruikt aan de buitenkant van bochten met als doel de snelheid van een auto te doen afnemen wanneer deze door verlies van controle van de weg gaat.

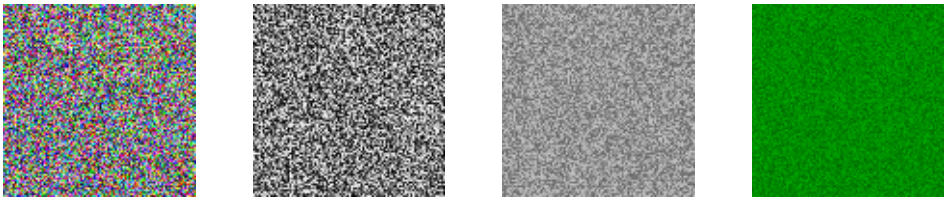
9.2 Belangrijke technieken

9.2.1 Noise (ruis)

Noise is in feite niet meer dan een opeenvolging van willekeurige waarden. In de muziek is dit te herkennen als ruis, op afbeeldingen door willekeurige kleurwaarden. Denk voor dit laatste aan zogenaamde 'sneeuw' op uw televisiescherm. De toepassingen van ruis zijn echter eindeloos. Er kan gras mee gemaakt worden, wolken, marmer patronen, ...

De eenvoudigste implementatie van noise in de wereld van afbeeldingen is de volgende: Bepaal een bepaalde grootte (in pixels) van een afbeelding, en geef vervolgens aan elke pixel een willekeurige kleur (meer over kleur is te vinden in hoofdstuk 2.2.4). De alpha waarde zal in dit deel steeds 255 zijn, wat er dus op neer komt dat alle kleuren geen enkele vorm van transparantie zullen bevatten.

Een mogelijk noise algoritme zal dan bij elke pixel aan elke kleurcomponent (rood, groen en blauw) een willekeurige waarde geven van 0 tot en met 255. Figuur 35a toont het resultaat van het werken met deze methode. Meestal wordt er echter grijze noise gecreëerd. Hier krijgen de rode, groene en blauwe componenten elk dezelfde willekeurige waarde (figuur 35b).



(a) **(b)** **(c)** **(d)**

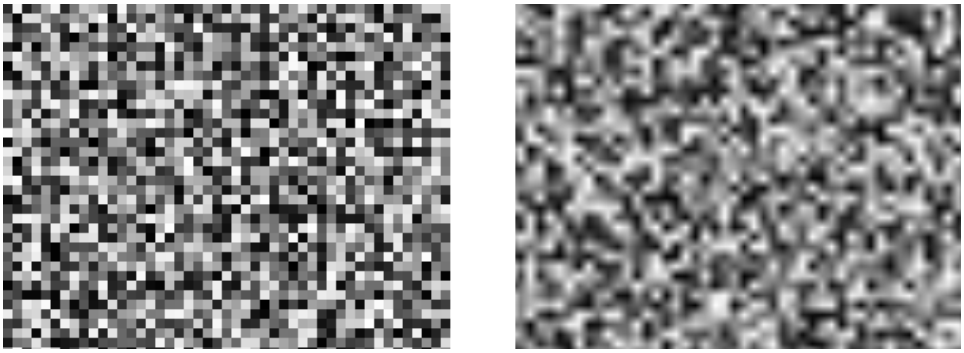
Figuur 35: Eenvoudige voorbeelden van noise

Het werken met noise wordt eigenlijk pas interessant wanneer er bepaalde limieten gesteld worden aan de willekeur waarmee de waarden van de kleurcomponenten gekozen worden. Voor figuur 35c werd er een variatie gebruikt op het algoritme verantwoordelijk voor de grijze noise. De waarde van een component ligt hier tussen 100 en 200, wat ervoor zorgt dat de grijswaarden dichter bij elkaar gaan aanleunen. Het laatste voorbeeld van eenvoudige noise is te zien op figuur 35d en zou gebruikt kunnen worden voor de creatie van gras. Zowel de rode als de blauwe component zijn hier 0. De groene component varieert hier van 100 tot 180.

9.2.2 Interpolatie

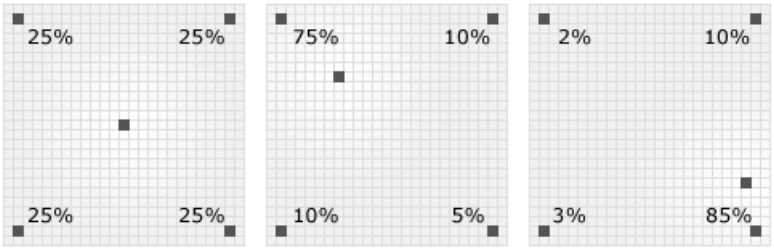
Interpolatie is in de ruime betekenis het bepalen van een onbekende waarde tussen andere, gekende waarden. Net als noise is interpolatie toepasbaar op vele domeinen. Op het gebied van afbeeldingen wordt dit meestal in verband gebracht met het vergroten van een bepaalde figuur. Hier worden nieuwe pixels toegevoegd aan de afbeelding, berekend uit omringende pixels.

Figuur 36 toont twee vergrote afbeeldingen met noise. De linkse afbeelding is vergroot zonder het gebruik van interpolatie. Hierdoor lijkt het dat de pixels gewoon groter zijn geworden. (Het schaal algoritme dat hier gebruikt wordt, heet het nearest-neighbour scaling algorithm.). De rechtse afbeelding daarentegen maakt wel gebruik van dit fenomeen.



Figuur 36: Uitvergrote noise mét en zonder interpolatie

Het algoritme voor interpolatie gebruikt in dit project, is vrij eenvoudig gehouden. Onbekende pixels worden berekend aan de hand van de 4 omringende pixels. Hoe dichter een omringende pixel bij de onbekende ligt, hoe meer de kleur van de onbekende pixel hier zal op gelijken. Figuur 37 stelt dit grafisch voor.



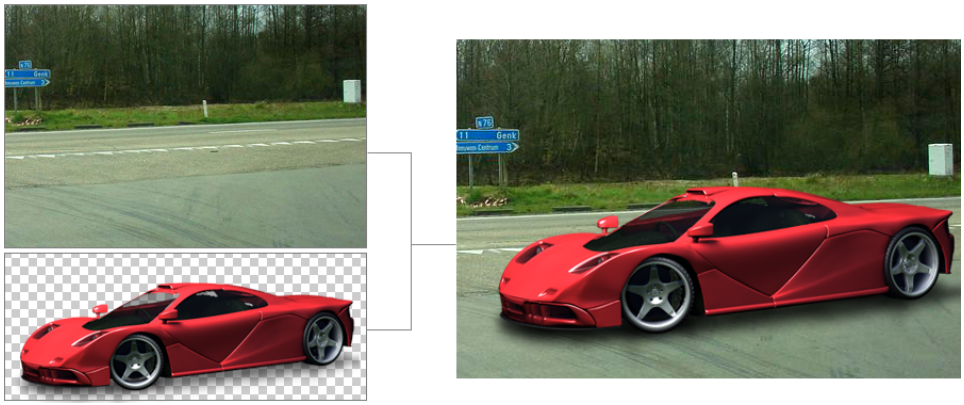
Figuur 37: Invloed van omringende pixels

9.2.3 Mixen van afbeeldingen

Wanneer er gewerkt wordt met afbeeldingen, komt het vaak voor dat twee afbeeldingen gecombineerd moeten worden. In het eenvoudigste geval wordt de ene afbeelding dan gewoon over de andere geplaatst, waardoor de onderste afbeelding enkel zichtbaar is daar waar de bovenste transparant is.

De algoritmes voor de verschillende soorten blendmodes (de verschillende manieren waarop afbeeldingen gemixt kunnen worden) zijn vrij moeilijk te begrijpen. Er moet namelijk gerekend worden met elke kleurcomponent in combinatie met alpha waarden. Deze algoritmes zijn dan ook niet zelf geschreven. Het is echter wel belangrijk dat het effect van een bepaalde blendmode begrepen wordt.

De eerste en in praktijk meest gebruikte vorm is de normal blendmode. Deze zorgt voor het plaatsen van twee afbeeldingen boven elkaar. De onderste afbeelding is zoals eerder gezegd enkel zichtbaar waar de bovenste transparantie bevat. Het feit dat zowel de bovenste als onderste afbeelding gedeeltelijk transparant kunnen zijn zorgt voor de moeilijkheid van het algoritme.

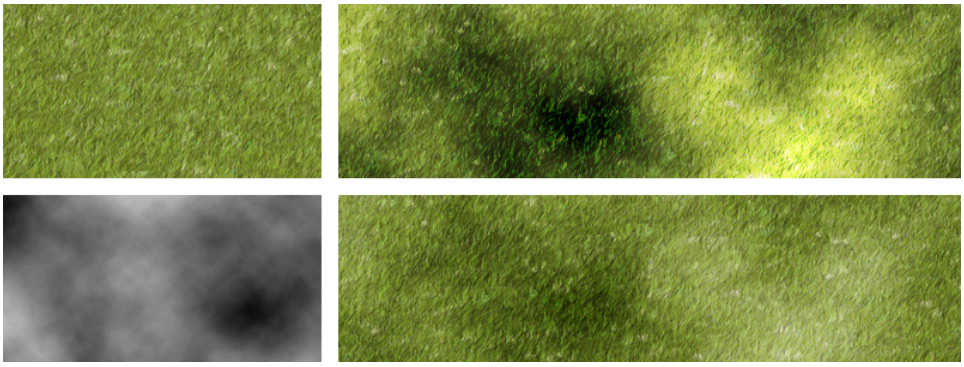


Figuur 38: Normal blendmode

De andere twee gelijken qua werking op elkaar. Het zijn de overlay en soft light blendmodes. Daar waar de kleuren op de bovenste afbeelding lichter zijn dan 50% grijs (#7F7F7F), wordt de onderste afbeelding lichter. Hoe lichter de kleur in de bovenste afbeelding, hoe meer de onderste afbeelding lichter zal worden. Daar waar de kleuren op de bovenste afbeelding donkerder zijn dan 50% grijs, wordt de onderste afbeelding donkerder. Als de bovenste afbeelding dus helemaal 50% grijs is zal de onderste niet beïnvloed worden.

Het grote verschil tussen overlay en soft light is de invloed van de bovenste afbeelding. Bij soft light zijn de gevolgen op de onderste afbeelding eerder

nuancerend. Zelfs wanneer de bovenste laag puur zwart is, zal de onderste laag slechts een lichte schaduw ondervinden. Bij overlay is het effect veel drastischer. Naargelang het gewenste resultaat wordt er in het project van beide blendmodes gebruik gemaakt. Afbeelding 39 toont het verschil. De afbeelding linksonder wordt over de afbeelding linksboven gelegd. Het resultaat van de overlay blend is rechtsboven te zien, het resultaat van het soft light algoritme rechts beneden.



Figuur 39: Het verschil tussen soft light en overlay blendmodes

Tenslotte werd er nog een vrij eenvoudige blendmode zelf geschreven. Deze maakt de kleuren op de onderste afbeelding donkerder wanneer de kleur op de bovenste donkerder is dan #C8C8C8 (vrij licht grijs). De eenvoud van dit algoritme impliceert enkele voorwaarden. De bovenste afbeelding mag enkel grijswaarden bevatten, en met transparantie wordt geen rekening gehouden.

9.2.4 Blur filter

De blur filter verandert elke pixel van een bitmap door het mengen van de omringende pixels met de kleur van de huidige pixel. Het effect hiervan is dus dat de afbeelding onscherp, wazig wordt. Hoe groter de invloed van omringende pixels, hoe groter het effect van de blur filter. Figuur 40 stelt de invloed voor van de omringende pixels.

De curve die gebruikt wordt voor het berekenen van deze invloed is een gaussiaanse curve. De formule voor een dergelijke curve in 2 dimensies is:

$$E^{-1/blur*x^2}$$

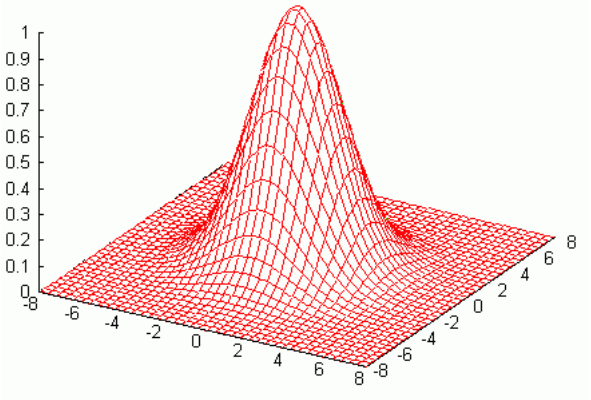
E is een constante die in dit project gelijk gesteld wordt aan 1.5. De variabele x is de afstand tussen het punt waar de invloed van wordt berekend tot het punt waar de invloed op van toepassing zal zijn. De variabele *blur* duidt op de grootte van de blur filter. In het voorbeeld op figuur 40 is deze

grootte 8 pixels. Wanneer we de formule uitbreiden naar het 3 dimensionale assenstelsel bekomt men de volgende formule:

$$E^{-1/blur*x^2} * E^{-1/blur*y^2}$$

De implementatie van deze formule is vrij eenvoudig, keerzijde van de medaille is de performantie van deze code. Stel dat er net als in het voorbeeld voor elke pixel op de afbeeldingen een gebied van 16 op 16 pixels moet berekend worden. Dit betekent per pixel 256 $(16 * 16)$ keer de bovenstaande bewerking maken. Voor een grote afbeelding loopt het aantal keer oproepen van de bewerking snel op in de miljoenen.

Heel dit proces kan worden versneld door het apart berekenen van horizontale en verticale blur. De kwaliteit van de blur filter gaat hierdoor een klein beetje naar beneden, maar er kan gesteld worden dat dit even goede resultaten oplevert. Door het gebruik van deze methode gaat het aantal bewerkingen significant naar beneden, dit zijn er nu slechts 32 per pixel (ervan uitgaande dat de grootte van de blur hetzelfde gebleven is).



Figuur 40: Voorstelling van invloed andere pixels

9.2.5 Anti-aliasing

Anti-aliasing is het proces dat zogenaamde kartelvorming tegengaat wanneer er een schuine lijn op een beeldscherm (of afbeelding) wordt getekend. Om aliasing tegen te gaan zijn er verschillende technieken. Bij het maken van de achtergrondafbeelding wordt er gebruik gemaakt van anti-aliasing door verkleining van een afbeelding. Als een afbeelding bijvoorbeeld twee keer kleiner wordt, worden vier pixels er één. Deze ene pixel bevat dan de gemiddelde kleurwaarde van de vier oude. Deze manier van anti-aliasing wordt supersampling genoemd. Afbeelding 41 laat het verschil zien tussen een gewone figuur, en één waarop anti-aliasing is toegepast.

Figuur 41: Anti-aliasing in actie

9.3 De verschillende stappen

9.3.1 Rasterizatie

Zoals gezegd was de eerste stap in het genereren van achtergronden het rasterizen van het circuit. Dit proces werd al eerder uitvoerig beschreven in hoofdstuk 4.3.2 op pagina 18. Er zijn twee grote verschillen tussen de rasterizatie voor pathfinding en achtergrondgeneratie.

Een eerste verschil is de schaal van het raster. Bij pathfinding was performantie belangrijker dan nauwkeurigheid. Voor het genereren van een raster voor achtergrondgeneratie wordt er niet zo naar deze performantie gekeken. Een achtergrond moet namelijk vooral mooi om naar te kijken zijn, en bij gevolg dus nauwkeurig.

De schaal van het raster wordt hier gelijk gesteld aan 2:1. Het raster is dus 2 keer zo groot als de afbeelding die uiteindelijk zal gecreëerd worden. Dit wordt gedaan om een natuurlijke vorm van anti-aliasing te veroorzaken.

Een ander verschil is het aantal terrein types op het raster. Standaard bevat rasterizatie er slechts twee, namelijk asfalt en gras. Aangezien het oog ook wat wilt, was het belangrijk nieuwe zones te definiëren die het geheel wat gevarieerder en aantrekkelijk zouden maken.

Een eerste groep zones zijn de wegmarkeringen. Eerst en vooral de finishlijn, maar ook de startpositie indicatoren en de lijn die het checkpoint markeert. De lijnen worden aan het raster toegevoegd aan de hand van het Bresenham lijn algoritme (ook te vinden in het hoofdstuk over rasterizatie). De finish en checkpoint lijnen zijn het eenvoudigst toe te voegen. De twee punten aan de kant van de weg waar de lijnen elk tussen lopen zijn immers al berekend.

Iets moeilijker liggen de indicatoren van de startposities. Dit zijn halve rechthoeken die de bestuurders duidelijk maken waar ze hun auto moeten

parkeren aan de start van een race. Hiervoor wordt weer de wiskunde gebruikt uit hoofdstuk 13.2, waar een punt berekend wordt aan de hand van een startpositie, een hoek en een afstand. De punten die de lange dwarse lijn vormen, liggen op een hoek loodrecht op de startrichting en op 8 pixels afstand van de positie van het startpunt. Wanneer deze twee punten verbonden worden krijgt men dus een lijn van 16 pixels waarachter de auto's zullen plaatsnemen. De kleinere hulplijntjes worden elk getrokken vanaf deze twee punten naar een punt dat in de omgekeerde hoek van de startrichting 4 pixels vandaar wordt verschoven.

De lijnen aan de zijkanten van het circuit worden tijdens het inkleuren van de verschillende zones bepaald. Hieronder is er daarover meer informatie te vinden.

Hierna wordt er meer variatie gestoken in de omgeving naast het asfalt, met name door de toevoeging van grindbakken en boomrijke gebieden.

Grindbakken komen traditioneel vooral voor aan bochten. Hoe scherper de bocht, hoe groter de omliggende zone met grind moet worden.

Een eerste vereiste om deze zones te bepalen, is het berekenen waar de bochten nu eigenlijk voorkomen. Hierbij worden alle punten van het circuit afgegaan. Een eerste belangrijk aspect van het herkennen van bochten is natuurlijk de hoek met het voorgaande en volgende punt op het circuit. Er moet echter ook rekening gehouden worden met de afstand die tussen die drie punten moet worden afgelegd. Als punten ver uit elkaar liggen wordt een bocht logischerwijs minder scherp. De formule om de breedte van de grindbak te bereken is de volgende:

$$asfaltBreedte + C * \frac{afstand}{hoek}$$

$AsfaltBreedte$ is de breedte van het circuit op dat punt, C is een constante die ervoor zorgt dat de bekomen waarde aangepast wordt aan een aanvaarde schaal. $Afstand$ is de som van de afstand tussen het vorige punt en het huidige, en afstand tussen het volgende punt en het huidige. Zelfs met deze formule blijven er soms 'pieken' ontstaan bij bepaalde bochten, waardoor de grindbak er zeer vreemd kan uitzien. Om dit tot een minimum te beperken wordt de breedte van het grind op een punt het gemiddelde van zijn eigen grindbreedte, en deze van het vorig en het volgend punt.

Na het berekenen van de breedte van de grindzone rond elk punt wordt er een nieuw raster gemaakt, maar de breedte van het circuit wordt eerst gelijkgesteld aan de breedte van het grind. Wanneer dit raster gecreëerd is, zal het weer enkel asfalt en graszones bevatten. Het moge duidelijk zijn dat het asfalt eigenlijk het grind is. De laatste stap is het mixen van informatie

met het eerste (en uiteindelijke) raster en het nieuwe. Daar waar het nieuwe raster asfalt is en het oude gras, wordt het oude raster grind.

Het toevoegen van de zone met bomen is eenvoudiger. De breedte van elk punt, dat dus eigenlijk de breedte van de grindzone bevat, moet gewoon vermenigvuldigd worden met een constante, bijvoorbeeld 2. Weer wordt er met deze nieuwe breedte waarden een raster gecreëerd. Waar het nieuwe en het oude raster gras is, wordt het oude raster een boomrijk gebied.

9.3.2 Generatie van wolken

Het effect van deze stap wordt vooral duidelijk wanneer hij niet wordt uitgevoerd. Wanneer de hierboven gedefinieerde zones worden ingekleurd, gaat dit voor vrij saaie resultaten zorgen. Gras heeft overal de zelfde tinten groen en het asfalt is buitenaards gelijkaardig over het hele circuit. Die dingen worden al heel wat interessanter als we hierover de schaduw van imaginaire wolken plakken. Bepaalde zones gras gaan donkerder zijn dan andere, en het asfalt gaat plots ook realistischer lijken.

Het genereren van deze wolken steunt op twee technieken: noise en interpolatie. Om een mooi wolkeffect te bekomen is het nodig verschillende lagen noise over elkaar te plakken. Er wordt begonnen met ruwe noise, die zo de ruige basisvorm van de wolken vastlegd. Denk hierbij aan blokken van bijvoorbeeld 100 op 100 pixels boven en onder elkaar, waar elk blok een willekeurige kleur heeft. Hierna wordt er een laag gecreëerd die fijnere noise genereert, met name de helft zo klein. Het 'gewicht' van deze laag zal echter lager liggen dan de eerste layer, waardoor de gecreëerde kleuren minder prominent aanwezig zullen zijn in het uiteindelijke resultaat. Dit proces wordt dan herhaald tot de grootte van de blokken klein genoeg is. Afhankelijk van de verdeling van gewicht kan dit variëren, maar in dit project ging het in het algemeen over blokken van 1 tot 2 pixels. Het uiteindelijke resultaat van alle opgetelde lagen is noise die veel minder willekeurig lijkt, omdat er een soort van patronen lijkt ontstaan te zijn. De wolken zijn geboren.

Interpolatie wordt in deze stappen gebruikt wanneer de blokken pixels groter of gelijk zijn aan 2 op 2 pixels.

De afbeelding die gegenereerd wordt heeft een grootte gelijk aan een raster dat op schaal 1:1 gecreëerd zou worden, dus de grootte die oorspronkelijk in TrackEditor bepaald was, en die de uiteindelijke achtergrond zal hebben.

Figuur 42: Vorming van wolken

9.3.3 Inkleuring van het circuit

Na het definiëren van de verschillende zones en het maken van de afbeelding met wolken is alle voorbereiding voltooid. Het eigenlijke inkleuren van de achtergrond kan beginnen.

Voor elke pixel van de afbeelding met achtergrond worden er eigenlijk vier berekend. Zoals eerder gezegd is het raster met de verschillende zones dubbel zo groot als de uiteindelijke afbeelding, en wordt zo de supersampling anti-aliasing verwezenlijkt. Nadat de kleur van de vier pixels is berekend wordt het gemiddelde van die kleur berekend. Dit gebeurt door het gemiddelde te berekenen van de rode, groene en blauwe componenten van elke kleur.

De inkleuring van zowel de zones asfalt, wegmarkering, gras en grind wordt gedaan door middel van het eenvoudige noise algoritme beschreven in hoofdstuk 9.2.1. Asfalt krijgt voor elk kleur component dezelfde waarde tussen 80 en 100 (#505050 tot #646464). Wegmarkering doet hetzelfde maar dan tussen 160 en 240 (#A0A0A0 tot #F0F0F0). Bij gras ligt de groene component tussen 60 en 140, de rode en groene component zijn gelijk en liggen tussen 0 en 10 (#003C00 tot #0A8C0A). Grind tenslotte bevat een rode component van 181 tot 221, een groene van 143 tot 183 en een blauwe van 98 tot 138 (#B58F62 tot #DDB78A).

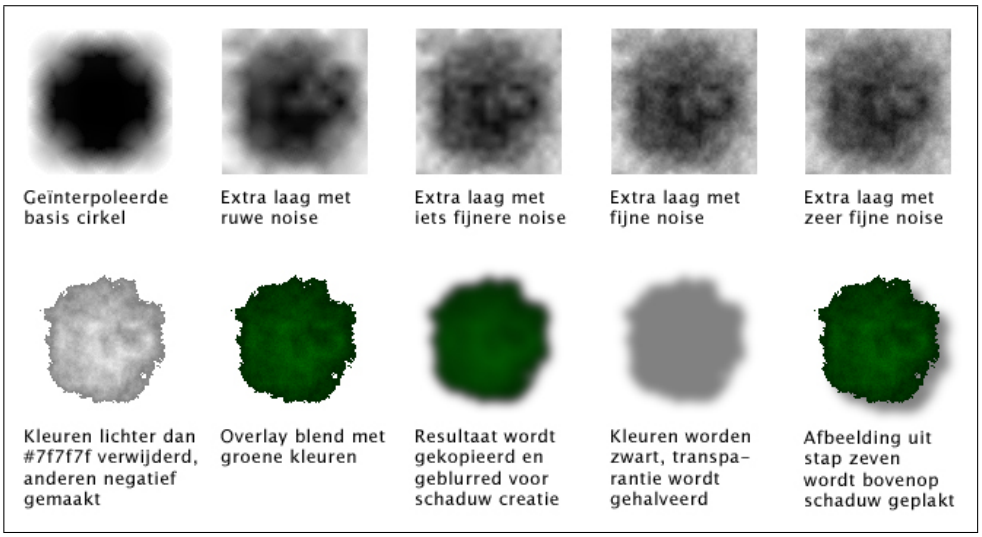
Wanneer de kleur van de (sub)pixel bepaald is, wordt hij nog gemixt met de afbeelding die de wolken bevat. Dit wordt gedaan via de soft light blendmode.

De asfaltzone bevat nog een andere eigenschap. Indien er namelijk een ander type zone, verschillend van wegmarkeringen, binnen een bepaalde afstand ligt, wordt er een lichtgrijze kleur geplaatst in plaats van de normale donkergrijze kleur van asfalt. Dit zorgt ervoor dat er aan de kant van het circuit witte lijnen ontstaan. Asfalt maakt ook als enige gebruik van de zelfgeschreven blendmode beschreven in hoofdstuk 9.2.3.

De boomrijke zone wordt hier hetzelfde behandelt als gras, de bomen zelf worden namelijk pas in de volgende stap toegevoegd.

9.3.4 Plaatsing en generatie van bomen

Een toepassing van noise die al meer tot de verbeelding zal spreken is de generatie van bomen.



Figuur 43: Vorming van een boom

Zoals te zien op figuur 43 wordt er weer gestart met een ruwe basisvorm die later meer vorm krijgt. Verschillend met de wolken is dat de eerste, meest ruwe laag vast ligt op gebied van kleuren. Het gaat namelijk om een geïnterpoleerde, uitvergrote cirkel. Deze basisvorm is essentieel in het vormingsproces. Hij bepaalt namelijk ongeveer de uiteindelijke vorm van de boom.

Over deze laag worden dan weer de noise lagen gelegd, bekend van het proces dat de wolken vormt. Nadat dit is afgerond, worden alle kleuren onder 50% grijs verwijderd, en is de uiteindelijk vorm van de boom bepaald. Op de plaats waar er nog kleuren zijn worden nu weer willekeurige groenwaarden geplaatst, vergelijkbaar met de productie van gras. Om de boom een gevoel van diepte mee te geven wordt de gecreëerde noise afbeelding meteen weer geblend met de groenwaarde door middel van het bekende overlay algoritme. Merk op dat hiervoor de noise afbeelding eerst negatief gemaakt moet worden om de boom donkerder te maken aan de zijkant. Laatste stap bij de creatie van bomen is de schaduw. Omdat er geen diepte factor is werd er een schaduwfilter ontwikkeld vergelijkbaar met 2D beeldbewerkingsprogramma's als Photoshop. Hiervoor wordt er eerst een geblurde afbeelding gemaakt van de boom, en dan omgezet in een grijstint met een bepaalde transparantiewaarde.

Na de creatie van de boom en zijn schaduw worden beide afbeeldingen bo-

venop de reeds gecreëerde achtergrond geplaatst door middel van normal blending. (De schaduw schuift enkele pixels naar rechtsonder, zodat het lijkt dat de zon linksboven staat.)

Nu dat het duidelijk is hoe een boom gecreëerd wordt rest nog het antwoord op de vraag waar ze geplaatst worden.

Dit gebeurt aan de hand van een imaginair raster dat op de achtergrond wordt gelegd. Een vak van dit raster heeft ongeveer dezelfde grootte als een boom. Per vak wordt er een willekeurige pixel bepaald, wanneer uit het raster met de verschillende zones blijkt dat er op deze plaats een boomrijke zone is wordt er op die plaats een boom gezet.

Figuur 44: Van raster tot volledige achtergrond

10 INI Bestand

10.1 Algemene werking

Om gemakkelijk opties aan het project toe te voegen, is er gekozen om met een initialisatie (INI) bestand te werken. Zo een bestand bevat traditioneel verschillende instellingen voor een computerprogramma en bestaat uit verschillende delen.

Eerst en vooral zijn er secties, deze worden gevormd door het gebruik van vierkante haakjes, bijvoorbeeld "[Sectie1]". Secties worden voornamelijk gebruikt om lange lijsten van instellingen te groeperen, en zo dus orde in chaos te scheppen.

De eigenlijke instellingen worden opgeslagen in verschillende parameters. Een parameter wordt geschreven in de vorm "parameternaam = waarde". Elke parameter moet op een andere regel geschreven worden.

Tenslotte zijn er nog regels die commentaar bevatten. Deze regels worden voorafgegaan door een puntkomma en worden genegeerd bij het inlezen van een bestand.

Een eenvoudig voorbeeld van een initialisatie bestand zou alsvolgt kunnen gaan:

```
[Resolutie]
;definieert de grootte van het venster
breedte = 1024
hoogte = 768
```

10.2 CIniFile klasse

CIniFile is een vrij populaire klasse in C++ die het gebruik van INI bestanden zeer eenvoudig maakt. Bestanden kunnen worden uitgelezen, aangepast en weer opgeslagen worden. Het aanpassen van de bestanden gaat van het aanpassen van parameterwaarden tot het aanmaken van nieuwe secties.

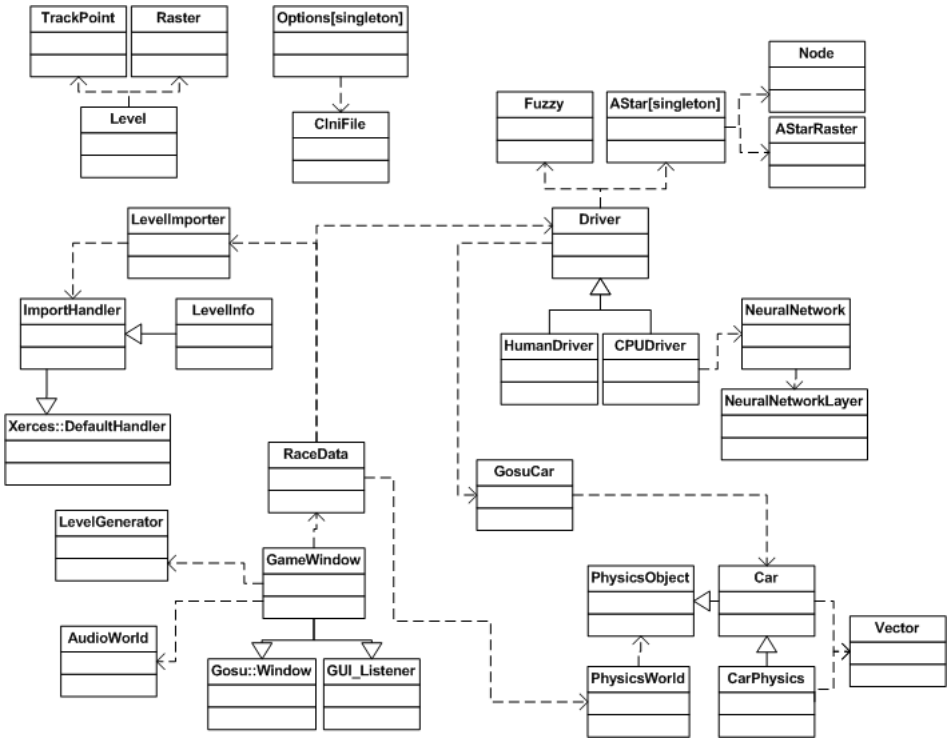
De klasse voegt wel enkele beperkingen toe aan het INI formaat. Zo mogen er geen spaties staan rond het '='-teken. Ook wordt alle commentaar binnen een sectie onder de sectietitel geplaatst. Commentaar per regel is dus geen optie.

In het project maakt de singleton klasse *Options* gebruik van een CIniFile object. De Options klasse zelf voegt enkele extra functionaliteiten toe. Zo

worden de verschillende instellingen, door het feit dat dit een singleton klasse is, over alle klassen beschikbaar. Ook zorgt deze klasse in elk geval voor de minimaal vereiste data. Dit is nodig wanneer een gebruiker bijvoorbeeld per ongeluk een parameter verwijderd heeft. Ook in het geval dat het hele bestand verwijderd wordt, zullen de standaardwaarden ingesteld worden, en zal bij het afsluiten van het programma een nieuw bestand te vinden zijn.

11 Programmastructuur

In Figuur 45 is het klassendiagram van de simulatie te zien. Alle connecties met *Level*, *TrackPoint* en *Options* zijn weggelaten om een chaotisch diagram te vermijden, aangezien een groot deel van de klassen van deze drie gebruik maken.



Figuur 45: UML diagram

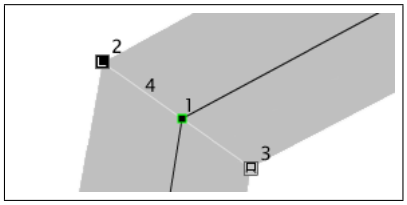
12 TrackEditor

12.1 Algemene werking

Het idee achter TrackEditor was het schrijven van een eenvoudig te gebruiken racecircuit editor. In zijn minimale vorm zou het dus mogelijk moeten zijn om er een bepaalde racebaan oppervlakte mee te kunnen definiëren, en een bepaalde oppervlakte van niet-racebaan (visueel voor te stellen door bijvoorbeeld gras). Om de interne flow van het circuit vrij vloeiend te houden werd er besloten om de gebruiker enkel de punten op het midden van de baan te laten bepalen. Hierdoor kan er dan berekend worden waar de zijkanten van de baan liggen, en hebben zowel de linker als de rechterkant van het circuit evenveel punten op gelijkwaardige plaatsen. De dwarse lijnen over het circuit zouden dan later een hulp kunnen zijn bij het design van bijvoorbeeld de AI. Later is echter gekozen om de rol van deze dwarse lijnen te minimaliseren, enkele kleine toepassingen buiten beschouwing gelaten. Een bepaald gebruik van deze breedtelijnen komt nog voor bij het maken van de finish- en checkpointmarkeringen tijdens de achtergrondgeneratie van een circuit (zie pagina 60).

De Editor is (grotendeels) geschreven vóór de rest van het project. Hij moest vlug functioneel genoeg zijn voor later gebruik. Omdat het visuele gedeelte van deze applicatie vrij interactief moet zijn met de input van de gebruiker is er gekozen om deze applicatie voor het grootste deel in Flash te maken. Hierin zijn dingen als het tekenen van lijnen of verslepen van objecten vrij makkelijk te programmeren. Het gebruik van Flash heeft echter één groot nadeel. Er kunnen namelijk niet rechtstreeks bestanden mee weggeschreven worden. Hiervoor werd er een C#.NET form applicatie rond gebouwd.

De circuitbestanden zelf zijn vrij eenvoudige XML bestanden. Verder in dit hoofdstuk is er meer informatie te vinden over de inhoud van deze bestanden.



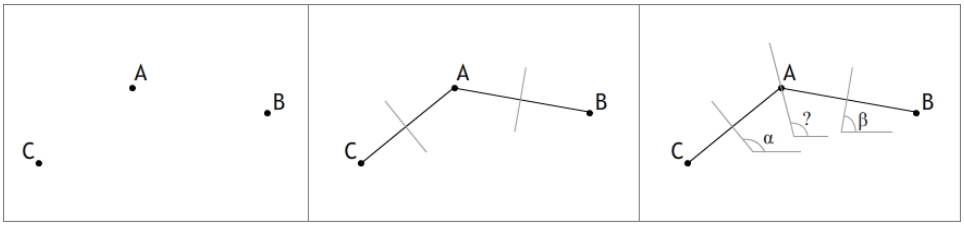
Figuur 46: Verschillende onderdelen van een circuit

Een circuit wordt opgebouwd met verschillende opeenvolgende punten. In een standaard circuit is dit punt het midden van de baan op die plaats. Deze punten zullen vanaf hier centrale punten genoemd worden. Op figuur 46 is dit punt 1. Op de afbeelding is ook het linker (2) en het rechter punt

(3) te zien. Deze punten worden intern allemaal voorgesteld door middel van de CircuitPoint klasse. De lijn die over de breedte van het circuit loopt wordt de breedtelijn genoemd (4) en wordt op zijn beurt voorgesteld door de Line klasse. Naast de positie van het punt, moet er per punt dus ook de breedte van de baan worden opgeslagen. Omdat het in praktijk handig is om het centrale punt niet altijd in het midden van het circuit te plaatsen is er naast de positie en de breedte van een punt nog een derde variabele, namelijk de offset. Indien de offset 0 is, ligt het linkerpunt van de baan op dezelfde afstand van het centrale punt als het rechterpunt, met name op de helft van de breedte. Als deze offset negatief is, wordt de rechterzijde van de baan breder, in het andere geval de linkerzijde.

Nu rest nog het antwoord op de vraag hoe de hoek wordt bepaald van een bepaalde breedtelijn. Het antwoord is vrij voor de hand liggend en maakt gebruik van de punten voor en achter het huidige centrale punt.

Op figuur 47 is te zien hoe de hoek van de breedtelijn op punt A wordt berekend. Eerst worden de hoeken van de loodlijnen op de lijnstukken naar A gemeten. De hoek die de breedtelijn zal maken is het gemiddelde van α en β.



Figuur 47: Berekening hoek van breedtelijn

12.2 Werking Flash gedeelte

De klasse waarmee alles begint en eindigt is de Functionality klasse. Deze klasse zorgt voor het de verbinding tussen de verschillende visuele componenten, achterliggende klassen en de omringende C# applicatie.

De Functionality klasse wordt bij de start van de applicatie aangemaakt. Deze stelt meteen verschillende Flash parameters in zoals schaalmodus en uitlijning. Hierna creëert hij via andere klassen het menu en linkt hij de verschillende knoppen aan hun functionaliteit. De basis van deze functionaliteit wordt ook voorzien in deze klasse, al gaat het dan meestal enkel om een aanroep naar een functie op een bepaalde klasse, of een aanroep naar de omringende applicatie.

Achterliggend verzorgt de Circuit klasse het grootste deel van het werk. Deze klasse beheert de verschillende circuitpunten en bevat alle functionaliteit nodig om een circuit te bewerken, laden en te exporteren naar XML. Circuit maakt gebruik van de klassen CircuitPoint en Line. Deze bevatten informatie over de breedtelijnen, centrale, linker en rechter punten en stellen deze visueel voor.

Hiernaast zijn er ook nog klassen die zorgen voor de voorstelling van het menu. MenuStructure bevat de hele menu structuur en is in feite niet meer dan een verzameling MenuStructureItems. Zo een item bevat een naam, de tekst voor het visuele menu item, een tekstuele beschrijving, een groep van submenu's die ook allen MenuStructureItems zijn en een referentie naar een functionaliteit uit de klasse Functionality. Tenslotte bevat het item ook nog een optionele naam voor gebruik in de toolbar.

De visuele kant zit ongeveer hetzelfde in elkaar. Hier zorgt de klasse VistaMenu voor het beheer van het menu. De naam van deze klasse is te danken aan de visuele stijl die aan het menu is gegeven, deze is namelijk gebaseerd op de algemene stijl van programma's draaiend op het gelijknamige besturingssysteem. De klasse biedt functionaliteiten aan om MenuItems toe te voegen.

Een MenuItem is het geheel van het globale menu item en zijn opties. Dus bijvoorbeeld het item 'Window' met zijn subitems 'Tools', 'Properties' en 'Statusbar' is één MenuItem. Het is dus ook niet mogelijk om items onder subitems te maken.

Het omzetten van de menu structuur naar het visuele menu gebeurt zoals eerder gezegd bij het aanmaken van een nieuw Functionality object. Tegelijk met de creatie van het menu maakt deze klasse ook de toolbar aan. Dit doet hij aan de hand van de toolbar-naam van een MenuStructureItem. Wanneer deze gedefinieerd is, wordt er aan de movieclip van de toolbar een nieuw icoon toegevoegd (het icoon wordt geladen uit de bibliotheek van het project).

Het bewerken van een circuit gebeurt door middel van verschillende modi. Er is een aparte modus voor het tekenen van een circuit, bewerken van een getekend circuit, verplaatsen van een circuit en het toevoegen van startposities. Naargelang de modus zal de gebruiker andere opties hebben en zal mogelijk het eigenschappen paneel veranderen.

De gebruiker verandert van modus door het gebruik van verschillende tools. Deze zijn te selecteren in het toolmenu, via de toolbar of toetsenbord shortcuts.

12.2.1 Het eigenschappen paneel

Het eigenschappen paneel (of properties tab) is een belangrijk element bij het gebruik van TrackEditor. Het paneel bevat namelijk contextuele informatie en opties. Er is een verschillend paneel voor volgende situaties: standaard, bij de selectie van een punt, bij het maken van een willekeurig circuit en bij het toekennen van startposities.

Het standaard paneel wordt weergegeven wanneer er geen andere situatie van toepassing is. Hij bevat globale informatie en de mogelijkheid om de naam van het circuit te veranderen.

Wanneer de gebruiker in de wijzigingsmodus zit, wordt de tab veranderd van zodra de gebruiker een centraal punt selecteert. Hierop is informatie te vinden over de positie, breedte van de baan en offset van de breedte. Ook zijn er opties om het punt het startpunt te maken, te deleten, of een nieuw punt toe te voegen.

Als de gebruiker een willekeurig circuit aanmaakt, zal het bijbehorende eigenschappen paneel getoond worden. Hierop staan verschillende schuifbalken die de vorm van het circuit zullen beïnvloeden.

Bij het toevoegen van startposities ten slotte, zal de gebruiker enerzijds de keuze krijgen posities toe te voegen, of een positie te selecteren en hier eigenschappen van te wijzigen.

Het eigenschappen paneel wordt door andere klassen aangesproken door middel van het Functionality object. Deze bevat namelijk de setProperties methode. Hieraan wordt meegegeven om welke situatie het gaat en een object. Dit object kan verschillen van type en zal naargelang de situatie anders gebruikt worden. Wanneer er bijvoorbeeld een centraal punt geselecteerd is zal het object het punt in kwestie zijn, en aan de hand daarvan zal de informatie op het paneel ingevuld worden.

12.2.2 Tekenen van een circuit

Wanneer de gebruiker zich in de teken modus bevindt kan hij een circuit (verder) tekenen. Wanneer er op de achtergrond wordt geklikt, wordt de addPoint functie van Circuit aangeroepen. Deze functie voegt dus een punt toe aan zijn lijst met als positie, de positie waar de muis zich bevond, en als breedte, indien mogelijk de breedte van de lijn op het vorige punt en een offset van 0. Indien er minstens drie punten aanwezig zijn, zal via de eerder beschreven methode de berekening van de hoeken plaatsvinden. Per punt

zullen er drie visuele CircuitPoints worden gemaakt. Het centrale punt zelf, en bijkomend de punten aan de linker- en rechterkant van de baan.

Het laatst getekende punt wordt altijd verbonden met het eerste, zodat het circuit steeds gesloten is.

12.2.3 Wijzigen van een circuit

In de modificatie modus zullen alle CircuitPoints extra functionaliteit meekrijgen. De centrale punten zullen verplaatsbaar worden gemaakt, waardoor ze dus een andere locatie krijgen. Wanneer dit gebeurt moet ook de hoek van de breedtelijn herberekend worden, samen met deze van de voorafgaande en volgende punten.

Ook de linker en rechter punten kunnen verplaatst worden. Normaal gezien zouden deze enkel lineair van of naar het centrale punt mogen verschoven worden. Om hiervoor te zorgen moet er zelfs voor Flash standaarden vrij hard gefoefeld worden. Dit is vermeden door de gebruiker wel volledige vrijheid te geven in de verplaatsing van het punt. Wanneer het punt echter wordt losgelaten zal de afstand tussen het verschoven linker- of rechter punt en het centrale berekend worden. Deze afstand wordt dan lineair (in de richting bepaald door de breedtelijn) toegepast op het verschoven punt. Ten gevolge van zo'n verschuiving moeten de breedte en offset opnieuw worden bepaald.

12.2.4 Verplaatsen van een circuit

Omdat een circuit logischerwijs groter mag zijn dan de maximale resolutie van de gebruiker, is het nodig dat het circuit verschoven kan worden.

Een eerste vorm van verplaatsen is alles verplaatsen. Hierbij wordt aan zowel de movieClip die de achtergrond bepaald, als die die het getekende circuit bevat, dezelfde functionaliteit gegeven. Wanneer één van beiden verschoven wordt, zal het andere mee verschuiven. Dit is de meest gebruikte verschuivingsmodus, aangezien er zo andere gebieden van het circuit kunnen worden bekeken, zonder iets aan de structuur van het circuit te veranderen.

De twee andere vormen van verplaatsing veranderen de positie van het circuit ten opzichte van de achtergrond. Men kan zowel de achtergrond als het circuit apart verplaatsen. Deze verplaatsingen worden pas bij het maken van een XML bestand verrekent in de coördinaten van de centrale punten.

12.2.5 Omdraaien van een circuit

Het omdraaien van een circuit is een vrij eenvoudige functionaliteit. Zowel de lijst van lijnen als de lijst van centrale punten wordt omgedraaid. Per punt moet ook de offset negatief gemaakt worden, om de vorm van het circuit te behouden.

12.2.6 Creëren van een willekeurig circuit

TrackEditor biedt de optie een een willekeurig circuit te laten genereren. Via een aantal parameters kan de gebruiker regels opleggen aan de wijze waarop het circuit wordt gegenereerd.

Standaard wordt er begonnen met een leeg raster. Dit raster is gewoon een 2 dimensionale array (vergelijkbaar met een dambord). In dit raster bevat elk vakje een getal. Standaard is dit getal nul, wat staat voor het feit dat er geen circuit terug te vinden is op die plek.

Bij het creëren van een nieuw circuit wordt er eerst en vooral een startpositie bepaald. Dit gebeurt willekeurig, enige voorwaarde is dat het vakje zich minstens twee vakjes van de zijkant bevindt. Dit is nodig om te voorkomen dat er geen circuits gemaakt kunnen worden. Op het startvakje wordt nu een één geplaatst in plaats van de nul. De waarde van de getallen in de vakjes zal vanaf nu telkens met één omhoog gaan. Hierna wordt er een huidige richting vastgesteld. Ook dit gebeurt willekeurig en kan tot acht verschillende resultaten leiden. Namelijk boven, onder, links, rechts, linksboven, rechtsboven, rechtsonder en linksonder.

Vanaf nu wordt er elke keer gezocht naar een volgend punt tot de start weer bereikt wordt. Aan de hand van de huidige richting wordt bepaald welke richting zal worden ingeslagen (de eerste keer zal dit dus aan de hand van de willekeurige richting zijn). Naargelang de meegegeven parameters zal de kans op een bepaalde nieuwe richting verminderen (wanneer de huidige richting naar boven is, is de kans groter om verder naar boven te gaan dan naar linksonder). Ook het feit dat er nog plaats is in het raster speelt hierbij een rol.

Wanneer de richting is gekozen wordt dit de huidige richting en komt het nummer van het vakje in de plaats van de nul die wees op de afwezigheid van circuit.

In het ideale geval blijft de vorige stap zich herhalen tot de start dus weer bereikt wordt. De kans is echter reëel dat er op een bepaald moment geen nieuwe richting gevonden wordt. Dit kan bijvoorbeeld optreden wanneer er

enkel reeds toegekende vakjes rondom het huidige einde van het circuit zijn. Zolang dit voorkomt moet er een vakje terug worden gegaan. De waarde van het getal in het vakje dat geen verdere uitweg kon bieden wordt op -1 gezet zodat er niet opnieuw in dezelfde val wordt gelopen. Er kan namelijk enkel een nieuw vakje gekozen worden met de waarde nul of één, wat in het laatste geval resulteert in het completeren van het circuit.

Het merendeel van de parameters die worden meegegeven zijn van invloed op de hierboven beschreven stap van de generatie. De parameter stijfheid (stiffness) bepaalt de kans op verandering van richting. De kans op scherpe bochten wordt bepaald door de jaggedness parameter. Hoogte en breedte tenslotte bepalen het aantal vakjes in het raster.

Nadat het circuit gemaakt is volgt de volgende stap. Namelijk de vertaling van de data in het raster naar het tekenen van een visueel circuit.

Voor de gemakkelijkheid wordt er vanuit gegaan dat de coördinaten van het raster dezelfde schaal hebben als die van het visuele circuit (in het echt moet er natuurlijk vermenigvuldigd worden met een bepaalde constante). Per vakje van het raster worden er twee centrale punten toegevoegd. Het eerste punt ligt tussen het midden van het huidige vakje, en het midden van het vorige vakje. Aan de hand van de hoek die wordt gemaakt tussen de baan naar dit vakje en vanuit dit vakje wordt het tweede punt berekend. Het tweede punt dient te voorkomen dat het circuit te hoekig wordt, en zal dus zo goed als mogelijk een mooie bocht willen teweeg brengen.

De constante die gebruikt wordt voor het omzetten van coördinaten uit het raster is afhankelijk van een andere parameter, namelijk de dichtheid (density). Hoe hoger de dichtheid hoe kleiner de constante en hoe dichter de verschillende centrale punten bij elkaar zullen liggen. De laatste parameter is marge (margin) en bepaalt de afstand van de zijkant van het circuit tot de zijkant van de achtergrond.

Ten slotte wordt er gezocht achter een goede startlijn. Meestal ligt het willekeurig gekozen startpunt midden in een bochtencombinatie. Om dit te verhelpen wordt er gekeken waar het langste rechte stuk is, en aan het einde ervan wordt dan het startpunt gelegd. Het langste stuk baan wordt hier bepaald door het aantal keer dat er bij het kiezen van een nieuwe richting, dezelfde richting is gekozen. Het heeft dus niet meteen een verband met de werkelijke lengte. (Deze is namelijk langer wanneer het rechte stuk schuin loopt.)

Na het bepalen van de startlijn worden er standaard nog vier startposities toegevoegd. Er wordt hieraan begonnen met twee gegevens, namelijk de positie van het startpunt en de hoek die het stuk baan voor de finish maakt. Het eerste startpunt ligt links op de baan. Er moet dus een bepaalde afstand

verschoven worden volgend de gegeven hoek plus negentig graden (als de richting boven is, en dus 90 graden, zal deze 180 graden en dus links worden). Via de methode uitgelegd op pagina 84 wordt de startpositie via de verkregen hoek over een kleine afstand verplaatst. Hierna gebeurt nog een soortgelijke verplaatsing, namelijk de verplaatsing naar achteren. Hiervoor wordt de hoek gewoon de omgekeerde hoek van het laatste stuk baan.

Deze bewerking wordt dan enkele keren in licht aangepaste vorm gebruikt voor de drie andere startposities.

12.2.7 Opslaan en inladen van een circuit

Zoals eerder vermeld, is Flash niet in staat om bestanden weg te schrijven. Technisch gezien gaat het hier dus niet over het opslaan van een bestand, maar wel om de creatie van de inhoud. Deze inhoud wordt dan met de hulp van de omringende .NET applicatie naar een bestand weggeschreven. Meer informatie over de communicatie tussen Flash en .NET applicaties is te vinden in het volgende hoofdstuk.

In hoofdstuk 3.1 wordt al uitgebreid beschreven hoe de inhoud van het circuit er moet uitzien. Samenvattend kan gesteld worden dat er drie delen zijn. Een deel met lineaire informatie, zoals de naam van het circuit en de achtergrondafbeelding. Daarna volgt een lijst van startposities en een lijst van circuitpunten.

De inhoud creëren van het bestand is een vrij eenvoudig proces. Het eerste gedeelte wordt gewoon lijn voor lijn weggeschreven. Hier en daar wordt informatie uit variabelen gehaald, de rest is allemaal hardcoded. De overige twee delen zijn makkelijk aan te maken door middel van een kleine lus. Per startpositie, en later per circuitpunt, wordt er een stuk XML code aan de voorlopige bestandsinhoud toegevoegd.

Het inladen van XML gaat via de ingebouwde XML klasse in ActionScript 2.0. Deze klasse gebruikt het Document Object Model (DOM) om XML bestanden te parsen. Zoals uitgelegd in hoodstuk 2.5 is het zo vrij eenvoudig om direct naar bepaalde delen van een bestand te springen. Het inladen van lineaire data wordt zo een vrij rechttoe rechtaan proces. Ook het opzoeken van de twee lijsten is vrij eenvoudig. Eerst wordt de node met bijvoorbeeld alle circuitpunten gezocht, en worden alle childnodes in een array geplaatst. Deze array wordt volledig afgelopen. Wanneer dan alle informatie over één punt gevonden is, wordt via de standaard functie om een punt toe te voegen, dit punt toegevoegd. Hetzelfde geldt voor de startposities.

12.3 Integratie in C#.NET applicatie

Zoals eerder gezegd is het onmogelijk om met Flash bestanden weg te schrijven. Voor dit probleem is de oplossing het gebruik van een C#.NET form applicatie. Een form applicatie is niet meer dan een typisch Windows venster gebruikt door windows programma's. Een ander voordeel van de integratie is dat het omringende Flashplayer venster niet verschijnt, zoals wanneer de Flashmovie geëxporteerd wordt naar een .exe bestand.

12.3.1 Communicatie tussen Flash en C#

Het is vrij simpel om een Flash filmpje te gebruiken in een .NET applicatie. Het volstaat namelijk om een bepaald component[21] op de form designer te slepen en daaraan de bestandsnaam mee te geven van het gewenste Flash filmpje. Een ander paar mouwen is de eigenlijke communicatie tussen het Flash object en de .NET applicatie.

Flash aanspreken via C#.NET gaat op verschillende manieren. Een eerste mogelijkheid om de SetVariable methode van het Flashobject op te roepen. Deze functie heeft twee parameters nodig, namelijk de variabelenaam en zijn waarde. Een andere mogelijkheid is het gebruik van de CallFunction methode. Het gebruik van deze methode is een beetje ingewikkelder, maar sluit meer aan bij de gewenste functionaliteit dan SetVariable. De methode vereist maar één parameter, namelijk een XML-request-string. Dit is niet meer dan een stuk tekst gevormd volgens de XML standaard die Flash informatie geeft over de gewenste functie oproep. Een voorbeeld hiervan is:

```
<invoke name="load circuit" returntype="XML">
   <arguments>
      <string>circuit.rst</string>
   </arguments>
</invoke>
```

Wanneer het Flashobject bovenstaande informatie ontvangt haalt het daaruit genoeg informatie om een bepaalde functie op te roepen. De functie die zal opgeroepen worden is in dit geval de functie waar "load circuit" naar refereert (meer hierover zo dadelijk). Indien deze functie een object teruggeeft zal ook dit in een vergelijkbare XML-string gegoten worden. In het project wordt er echter nooit een object teruggegeven dus hier zal niet verder op in gegaan worden. Naast de functie die zal worden opgeroepen worden ook de

[21] Shockwave Flash Object - Dit is terug te vinden in de lijst van beschikbare COM componenten wanneer de Flash Player ActiveX control geïnstalleerd is op de computer

parameters voor die functie gedefinieerd. Uit het voorbeeld is één argument te halen, namelijk een string met de waarde “circuit.rst”.

In Flash zelf moeten er eerst enkele maatregelen getroffen worden voor dat er zich een eigenlijke aanroep van een functie voordoet. Flash bevat namelijk een klasse ExternalInterface. Deze klasse laat Flash toe te communiceren met zijn omringende omgeving, dit kan een webpagina zijn, of zoals in dit project de .NET applicatie.

```
ExternalInterface.addCallback("load circuit", this, loadCircuit);
```

Bovenstaande regel code maakt een referentie (“load circuit”) naar de loadCircuit methode op het this object. Deze methode zal dus degene zijn die wordt uitgevoerd wanneer de .NET applicatie de CallFunction methode oproept met het eerder gegeven XML voorbeeld.

Om via Flash de omringende .NET applicatie aan te spreken wordt er gebruik gemaakt van de fscommand functionaliteit. Dit is te vergelijken met het gebruik van ExternalInterface, het communiceert namelijk ook met de omliggende applicatie. De fscommand functie biedt ruimte voor twee parameters van het type string. De eerste wordt, in dit project, gebruikt voor het aangeven van de gevraagde functionaliteit, de tweede voor informatie nodig om de functionaliteit uit te voeren. In Flash gebeurt dan bijvoorbeeld het volgende:

```
fscommand("save", circuit.toXML());
```

Deze lijn geeft dus aan de C#.NET applicatie aan dat een bepaalde tekst moet worden opgeslagen, namelijk de door Flash gegenereerde XML omschrijving van een circuit.

In de .NET applicatie kan er gemakkelijk een functie gemaakt worden die luistert naar fscommand opdrachten vanuit Flash. Deze zit namelijk standaard bij een Flash object. Een eenvoudig voorbeeld van C# code gaat als volgt:

```
private void FlashObj_FSCommand(object sender, FSCommandEvent e){
   String arg1 = e.command.ToString();
   String arg2 = e.args.ToString();
   if (arg1 == "save")
      SaveToFile(arg2);
}
```

De twee argumenten die origineel werden meegegeven aan fsCommand zitten nu beiden in het FSCommandEvent object. De eerste twee regels in de methode halen de argumenten er weer uit.

12.3.2 Werking

De TrackEditor .NET applicatie bestaat uit 2 forms. Eén is verantwoordelijk voor het tonen van het eigenlijke programma. Het andere is het about venster, met daarop statische informatie over het product. Aangezien het informatie venster weinig tot geen onderliggende code bevat wordt hier enkel het eerste venster besproken.

Om te beginnen wordt logischerwijs eerst de Flash applicatie in het Flash object geladen. Hierna worden de meegegeven argumenten aan de main functie nagekeken. Wanneer het eerste argument bepaald is, wordt dit namelijk aanzien als een bestandsnaam, en zal het bestand geopend worden. Hiervoor wordt dan meteen een opdracht naar Flash gestuurd, die dan het XML bestand zal uitlezen en het circuit zal tekenen.

De applicatie heeft twee belangrijke variabelen, een string *filename* en een booleaanse waarde *changes*. Filename houdt de bestandsnaam bij van het huidige circuit. Changes bepaalt of er sinds de laatste keer dat het bestand opgeslagen werd, veranderingen zijn aangebracht.

De belangrijkste functie van de applicatie is deze die luistert naar fsCommand van het Flash object. Zoals eerder gezegd bevat deze twee door Flash bepaalde argumenten. Het eerste hiervan is een bepaalde oproep, die de uit te voeren taak omschrijft.

Een overzicht van de verschillende oproepen:

- **New**: Flash geeft aan dat er een nieuw circuit wordt gemaakt. Filename wordt een lege string en changes wordt false.
- **Open**: Flash geeft aan dat de gebruiker een eerder gemaakt circuit wilt openen. Aan de hand van de OpenFileDialog klasse wordt er een venster getoond waar de gebruiker kan zoeken naar XML of RST bestanden. Als en slechts als de gebruiker een bestand selecteert wordt de Flash applicatie hiervan op de hoogte gebracht. De bestandsnaam van het circuit bestand is in dit geval een argument. Filename wordt in dit geval aan de geopende bestandsnaam gelijkgesteld, changes wordt false.
- **Close**: Flash geeft aan dat het huidige circuit wordt gesloten. Filename wordt een lege string en changes wordt false.
- **Save**: Flash geeft aan dat de gebruiker het circuit wilt opslaan. Indien filename reeds een bestandsnaam bevat wordt het circuit bestand overschreven met de nieuwe, meegegeven, circuit data. Changes wordt false. In het andere geval wordt de 'save as' actie uitgevoerd.

- **Save as**: Flash geeft aan dat de gebruiker het circuit wilt opslaan onder een nieuwe bestandsnaam. Aan de hand van de SaveFileDialog klasse wordt er een venster weergegeven waarmee de gebruiker het pad en de bestandsnaam kan kiezen van het bestand. Filename wordt de gekozen bestandsnaam en changes wordt op false gezet.

- **Import background**: Flash geeft aan dat de gebruiker een achtergrondafbeelding voor het circuit wilt laden. Net als bij open wordt er weer een OpenFileDialog venster getoond. Nu kunnen er enkel afbeeldingen gekozen worden. Als en slechts als de gebruiker een bestand selecteert wordt de Flash applicatie hiervan op de hoogte gebracht. De bestandsnaam van de geopende afbeelding is dan een argument. Changes wordt in dit geval op true gezet.

- **Random circuit**: Flash geeft aan dat er een willekeurig circuit gemaakt zal worden. Indien changes true is zal via de MessageBox klasse een waarschuwing aan de gebruiker getoond worden. Hiermee wordt de gebruiker gewaarschuwd dat het maken van een willekeurig circuit een ouder, niet opgeslagen circuit, volledig zal verwijderen.

- **About**: Flash geeft aan dat de gebruiker meer informatie over het programma wilt zien. De tweede form van de .NET applicatie wordt nu getoond.

- **Changes**: Flash geeft aan dat de gebruiker een wijziging heeft aangebracht aan het circuit. Changes wordt op true gezet.

- **Exit**: Flash geeft aan dat de gebruiker het programma wilt sluiten. De .NET applicatie zal zichzelf beëindigen.

- **Maximize**: Flash geeft aan dat de gebruiker het venster wilt maximaliseren of uit de gemaximaliseerde status wilt halen. Via de FormWindowState klasse wordt de gevraagde status van het venster ingesteld.

- **Minimize**: Flash geeft aan dat de gebruiker het venster wilt minimaliseren. Via de FormWindowState klasse wordt de applicatie geminimaliseerd.

- **Menu size**: Een laatste feature van deze applicatie is het verschuiven van de applicatie over het scherm wanneer het venster een normale grootte heeft (en dus niet gemaximaliseerd is). Hiervoor moet ofwel de typische blauwe windows programma balk te zien zijn. Aangezien dit niet gewenst is kan er altijd via bijvoorbeeld een paneel op de form dit gesimuleerd worden. Dit paneel moet dan vóór het Flash object staan. Via een bepaalde implementatie van het mouseMove event op dit paneel is het verschuiven dan te verwezenlijken.

Het paneel zelf staat op de plaats waar in het Flash object de menubalk te zien is, en heeft als achtergrond dezelfde afbeelding als de menubalk. Hierdoor wordt het paneel perfect verstopt, en kan de gebruiker het venster dus op die plaats verschuiven.

De eigenlijke informatie die met deze oproep wordt meegegeven is de exacte positie en grootte die het paneel mag hebben.

Bij alle oproepen die ervoor kunnen zorgen dat het huidige circuit verwijderd wordt, wordt er nog een extra test uitgevoerd. Als changes true is wordt er eerst aan de gebruiker gevraagt of hij de veranderingen aan het circuit wilt opslaan. Indien dit het geval is wordt de save oproep uitgevoerd.

Nadat een oproep voltooid is wordt er naar de Flash applicatie een oproep gestuurd die zegt dat de gewilde functionaliteit uitgevoerd is en de uitvoering van het filmpje verder kan gaan.

13 Gebruikte wiskunde

13.1 Afstand tussen twee punten

13.1.1 Afleiding van de formule

Voor afstandsberekening wordt een formule gebruikt, afgeleid van de stelling van Pythagoras. Deze stelling luidt $a^2 = b^2 + c^2$ en geldt voor alle rechthoekige driehoeken. Het kwadraat van de lengte van de schuine zijde (a), is gelijk aan de som van de kwadraten van de rechthoekszijden (de twee zijden die de rechte hoek vormen (b en c)). Stel dat de schuine zijde bepaald wordt door het punt P_1 met coördinaten (x_1, y_1) en het punt P_2 met coördinaten (x_2, y_2). Uit deze informatie in combinatie met de stelling van Pythagoras kan de lengte van de schuine zijde afgeleid worden.

$$a^2 = b^2 + c^2$$

$$\Downarrow$$

$$a = \sqrt{b^2 + c^2}$$

$$\Downarrow$$

Omdat het gaat om een rechthoekige driehoek kan de lengte van de verticale zijde voorgesteld worden door $x_2 - x_1$ en de horizontale lengte door $y_2 - y_1$. Het feit dat er zo negatieve lengtes kunnen ontstaan wordt verholpen door het omringende kwadraat.

$$\Downarrow$$

$$a = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

De afstand tussen twee punten P_1 en P_2 is dus gelijk aan de afstand van de schuine zijde uit een rechthoekige driehoek, in bovenstaande vergelijking a.

13.1.2 Praktisch gebruik

Wanneer er in praktijk twee afstanden vergeleken moeten worden, wordt meestal het gebruik van de vierkantswortel uit de vergelijking achterwege gelaten. Hierdoor moeten dus de kwadraten van de twee afstanden vergeleken worden. Zelfs wanneer de afstand waarmee vergeleken moet worden tot de tweede macht moet worden verheven, gaat dit sneller dan het nemen van een vierkantswortel. Een vierkantswortel bepalen heeft veel meer berekeningstijd nodig dan het vermenigvuldigen van 2 getallen. Dit is vrij logisch aangezien er geen exacte formule is om een vierkantswortel te bepalen. Deze moet benaderd worden, wat vele stappen inneemt. Vermenigvuldigen daarentegen, is één enkele operatie.

13.2 Verschuiving punt over een afstand en een hoek

Wanneer men een punt wil verschuiven over een bepaalde afstand en een bepaalde hoek, kan dit berekend worden met trigonometrie. Het komt erop neer dat het nieuwe punt P(x, y) wordt bekomen door het originele punt O(a, b) te projecteren op een cirkel met een straal van de gewenste afstand en over de gewenste hoek.

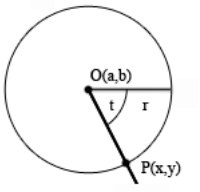
Dit wordt voorgesteld met deze formules:

$$x = a + r * \cos(t)$$

$$y = b + r * \sin(t)$$

Waarbij r de straal is en t de hoek.

Figuur 48 verduidelijkt de verschuiving.



Figuur 48: Verschuiven punt met trigonometrie

13.3 Afgeleide van de sigmoid functie

Een standaard sigmoid functie wordt beschreven met volgende formule: $f(x) = \frac{1}{(1+e^{\frac{-x}{c}})}$, waarbij c gebruikt kan worden om de functie over de horizontale as uit te rekken of samen te duwen.

Bij het berekenen van de afwijking van een neuron bij een neuraal netwerk is de afgeleide van deze functie nodig. Deze wordt bekomen met volgende stappen.

$$\frac{ds(x)}{dx} = \frac{1}{(1+e^{\frac{-x}{c}})}$$

$$\Downarrow$$

De afgeleide van een breuk wordt opgelost met deze formule: $\left(\frac{f}{g}\right)' = \frac{f'g-fg'}{g^2}$.

$$\frac{ds(x)}{dx} = \frac{1'(1+e^{\frac{-x}{c}}) - 1(1+e^{\frac{-x}{c}})'}{\left(1+e^{\frac{-x}{c}}\right)^2}$$

$$\Downarrow$$

De afgeleide van een exponentiële functie is gelijk aan die functie: $\frac{d(e^x)}{dx} = e^x$. De afgeleide van een constante is 0: $1' = 0$.

$$\frac{ds(x)}{dx} = \frac{0(1+e^{\frac{-x}{c}}) - (0+e^{\frac{-x}{c}})}{\left(1+e^{\frac{-x}{c}}\right)^2}$$

$$\Downarrow$$

$$\frac{ds(x)}{dx} = \frac{-e^{\frac{-x}{c}}}{\left(1+e^{\frac{-x}{c}}\right)^2}$$

$$\Downarrow$$

$$\frac{ds(x)}{dx} = \left(\frac{1}{\left(1+e^{\frac{-x}{c}}\right)}\right)\left(\frac{1}{\left(1+e^{\frac{-x}{c}}\right)}\right)(-e^{\frac{-x}{c}})$$

$$\Downarrow$$

$$\frac{ds(x)}{dx} = \left(\frac{1}{\left(1+e^{\frac{-x}{c}}\right)}\right)\left(\frac{-e^{\frac{-x}{c}}}{\left(1+e^{\frac{-x}{c}}\right)}\right)$$

$$\Downarrow$$

$$\frac{ds(x)}{dx} = s(x)(1-s(x))$$

De afgeleide van de sigmoid functie is die functie maal 1 min die functie.

14 Retrospectie

14.1 Artificiële intelligentie

De combinatie van A* en neurale netwerken is vernieuwend en ontwikkeld om eender welk circuit te kunnen berijden. Toch zijn er enkele nadelen:

- AI faalt op 8-vormige circuits omdat A* geen zin van richting heeft en dus bij de kruising gewoon de baan afsteekt
- Wanneer de auto veel van het pad afwijkt, zal het niet herberekend worden vanuit zijn positie, maar moet hij zelf terug naar het pad rijden
- Herberekende paden zijn niet perse betere paden

14.1.1 Verbeteringen

Het berekenen van de bereikbare nodes bij A* steunt op enkele trigonometrie berekeningen die het gedrag van een auto op simpele wijze proberen te simuleren. Om dit realistischer te maken en zo het pad ook te verbeteren zou moeten gebruik gemaakt worden van de physicsengine om verplaatsingen te berekenen. Dit zou als enorm voordeel hebben dat aanpassingen in de physics engine geen weerslag hebben op de AI. Het grote probleem hierbij en ook de reden waarom het niet op deze manier is geïmplementeerd is de verminderde performantie van de pathfinding. De simpele berekeningen die duizenden keren worden gemaakt worden vervangen door ingewikkelde berekeningen die meer processor cyclussen nodig hebben, waardoor het geheel exponentiëel vertraagt.

14.1.2 Alternatieve pathfinding

In de huidige implementatie bestaat er geen mogelijkheid om over tijd het pad op een circuit te verbeteren. Het wordt weliswaar constant herberekend, wat regelmatig verbeteringen oplevert, maar ook mindere alternatieven worden zonder probleem geaccepteerd. Een oplossing hiervoor kan een Ant-algoritme zijn. Dit zijn relatief nieuwe toepassingen binnen het AI domein en baseren zich op het gedrag van mieren. Simpelweg komt dit neer op volgende stappen:

1. At random nodes aflopen op zoek naar het doel
2. Onderweg 'feromonen' achterlaten

3. Na verloop van tijd zijn de beste paden die waar het meest feromonen liggen

Een andere manier om paden te evalueren is bij slecht rijgedrag de nodes rond het huidige pad penaliseren op basis van de orientatie binnen het pad. Op deze manier is er minder kans dat de volgende keer deze rijrichting in die buurt zal gebruikt worden en zal dat stuk circuit onder een andere hoek aangesneden worden.

15 Besluit

Het eindresultaat van dit eindwerk is, verrassend genoeg, groter en complexer uitgevallen dan hetgeen oorspronkelijk beschreven was in de doelen van dit project. Een physics engine werd geselecteerd na veel gewik en geweeg, om daarna uitgebreid te worden met collision detection en terreinafhankelijkheid. Het grafische gedeelte bestaat uit een volledig zelf ontwikkeld GUI framework voor de Gosu library. Extra features als een circuit editor en procedurele circuit- en achtergrondgeneratie hebben het geheel enorm verrijkt en vereenvoudigd voor de eindgebruiker.

Toch moet ook besloten worden dat het hoofddoel, namelijk competitieve artificiële intelligentie, zeker niet volledig bereikt is. Onderling rijdt de AI weliswaar een competitieve race over paden die de ideale lijn benaderen, maar tegenover een persoon met enige notie van autorijden vervalt de AI tot een amateuristisch niveau. Dit is onder andere te wijten aan het objectief en de ongewone samenstelling van de AI engine (in het geval van een race simulatie). Terwijl andere engines voor races geholpen worden door extra informatie of gewoon niet zo adaptief en realistisch proberen te zijn, is het de bedoeling van de ontwikkelde engine dat hij op eender welk circuit kan rijden en zelfs dat circuit kan leren kennen, hetgeen lukt tot op zekere hoogte.

16 Literatuurlijst

1. *C++ - wikipedia.*
   http://en.wikipedia.org/wiki/C++.
2. David M. Bourg en Glenn Seemann. *AI for Game Developers.* O'Reilly Media, 2004.
3. *Dragging a frameless window in C#.NET.*
   http://bnetdev.net/showpost.php?p=9859postcount=1
4. Elliotte Rusty Harold. *Processing XML with Java*, hoofdstuk 9, pagina 489. Addison-Wesley, 2002.
5. *How a Blur Filter Works.*
   http://home.scarlet.be/zoetrope/blur.htm
6. Hugo Elias. *Perlin Noise.*
   http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
7. Josh Chunick. *Blend Mode Algorithms.*
   http://www.mediamacros.com/item/item-1006687616/
8. Julian Raschke. *Gosu Gamedevelopment Library.*
   http://code.google.com/p/gosu/
9. Marcus Monstrous. *Monstrous physics engine.*
   http://web.archive.org/web/20040803175849/home.planet.nl/~monstrous/tutcar.html
10. Matt Kincaid. *2D Car Physics engine.*
    http://www.gamedev.net/reference/programming/features/2dcarphys/
11. *Optimizing alpha blending of two 32bit images.*
    http://objectmix.com/graphics/374350-optimizing-alpha-blending-two-32bit-images.html
12. Oren Becker. *2D Rotated Rectangles Collision Detection.*
    http://www.ragestorm.net/tutorial?id=22
13. Russell Smith. *Open Dynamics Engine.*
    http://www.ode.org/
14. Shane Hill. *CIniFile.*
    http://www.codeguru.com/cpp/misc/misc/article.php/c233/
15. *Sigmoid functie.*
    http://nl.wikipedia.org/wiki/Afbeelding:Logistic-curve.png
16. Tim Walters. *Embedding Flash in a Windows Forms .NET app.*
    http://weblogs.asp.net/twalters/archive/2003/12/29/46265.aspx