

Verslag: Fysische simulatie en Game AI

Inhoud

[1. Inleiding 3](#_Toc490771743)

[2. Fysieke Simulatie 4](#_Toc490771744)

[2.1 Begin 4](#_Toc490771745)

[2.2 Voorstudie 4](#_Toc490771746)

[2.1 Display 4](#_Toc490771747)

[2.3 Render-Loop 5](#_Toc490771748)

[2.4 Game-Loop 6](#_Toc490771749)

[2.5 Timer 6](#_Toc490771750)

[3. Game AI 7](#_Toc490771751)

[3.1 Voorstudie 7](#_Toc490771752)

[3.2 Implementatie 7](#_Toc490771753)

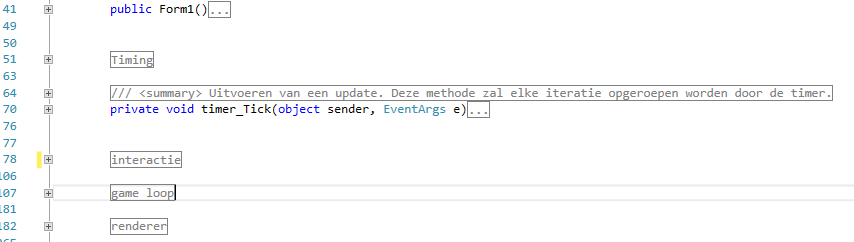
[4. Bronnen 9](#_Toc490771754)

# Inleiding

Voor Game Programming Periode 2 wordt er een pad-vind simulatie gemaakt. Deze simulatie wordt gemaakt in 2 delen. In het eerste deel maakt men de RRT die het pad vindt. Het tweede deel bestaat uit de fysische simulatie. Dit houdt in: de bewegingen van de robot, de versnelling en vertraging van deze robot en de eventuele botsingen.

# Fysieke Simulatie

## Begin

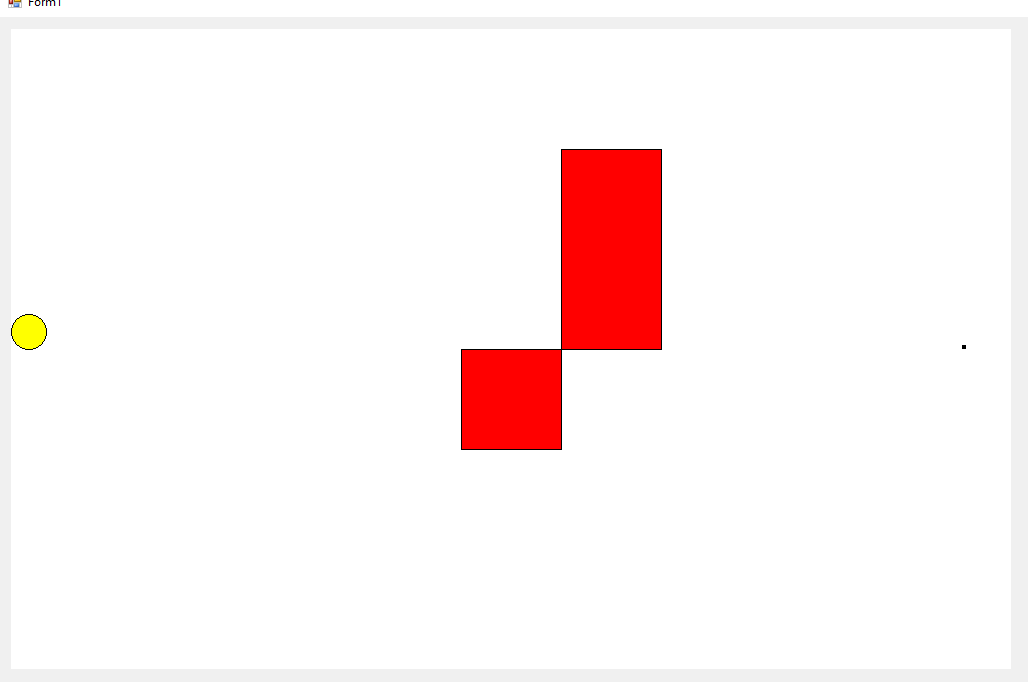
Om een werkende simulatie te maken moet men eerst bepaalde componenten voorzien. Ten eerste Wordt er een GUI gemaakt in een Windows-Form. Hierop maken we een display waarop de simulatie zal worden afgespeeld. Dit display steken we in een Render-Loop die steeds de display opnieuw tekent maar elke keer als er eerst nieuwe RRT-nodes worden getekend en vervolgens als die robot zich verplaatst. Deze nieuwe nodes en coördinaten krijgen we in het volgende deel: de Game-loop. In deze loop berekent men de verplaatsingen van de robot, de berekeningen van de nodes en hierin vindt men ook de collision-detection die ervoor zorgt dat de robot en de muren niet overlappen. Als laatste wordt nog een timer toegevoegd die de loops ondersteund.

## Voorstudie

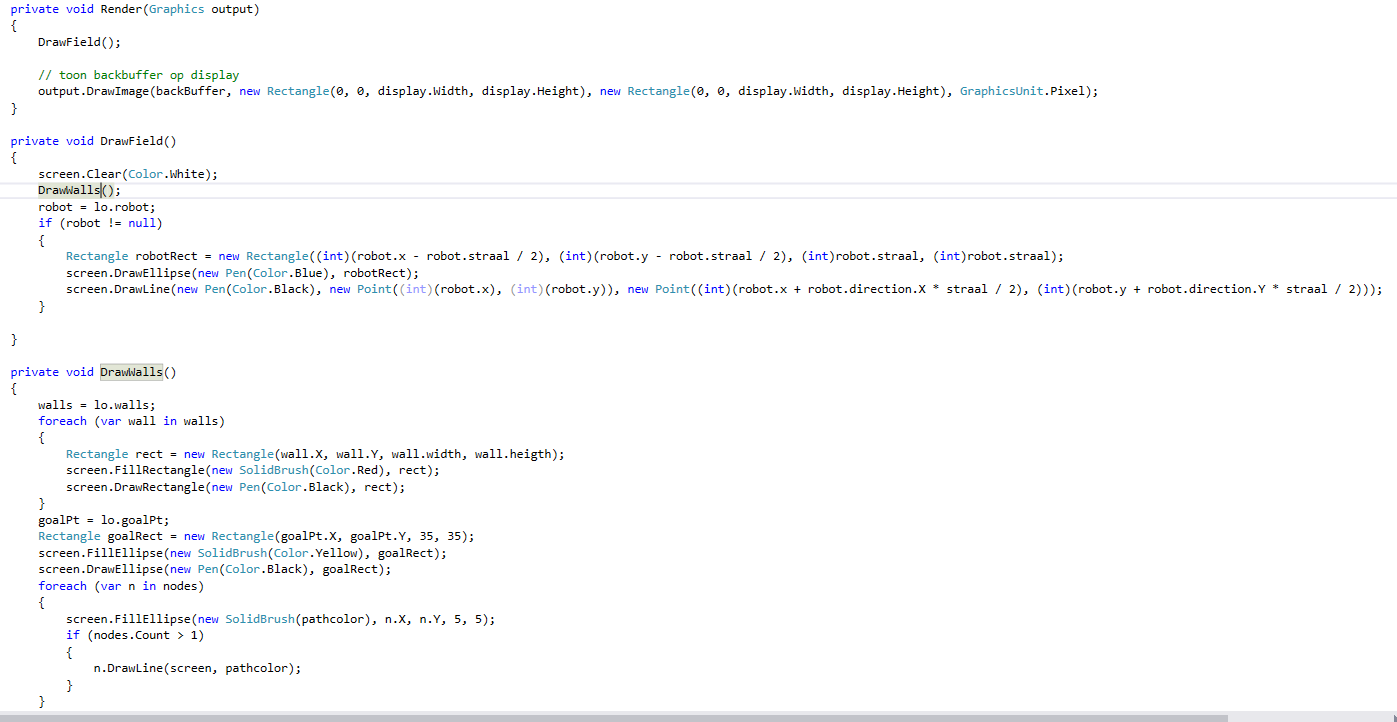
Bij het tekenen van een display maakt men vooral gebruik van de onderdelen in de System.Drawing namespace. Hierin zit geen echte moeilijkheid aangezien alle berekening voor wat waar getekend moet worden uit de Render-loop wordt gehaald en in de Game-Loop wordt berekend. De grootste moeilijkheid zit in de Game-Loop. Hier is een groot deel van het opzoekwerk in gekropen, enerzijds naar de bewegingen van de objecten en anderzijds naar de Collision-detection[1].

## 2.1 Display

In de display wordt gestart met een witte achtergrond aan te maken. Vervolgens wordt er op deze achtergrond de muren getekend wiens data uit de App.config wordt gehaald. Hiervoor wordt een simpele FillRectangle methode gebruikt.



Dit display wordt aangemaakt in de Render-Loop en wordt dus steeds opnieuw aangemaakt. Dit wordt gedaan omdat de posities van de robot en de nodes veranderen. Deze robot en nodes worden aangemaakt in een rectangle met variabele coordinaten die in de Game-Loop worden berekend.

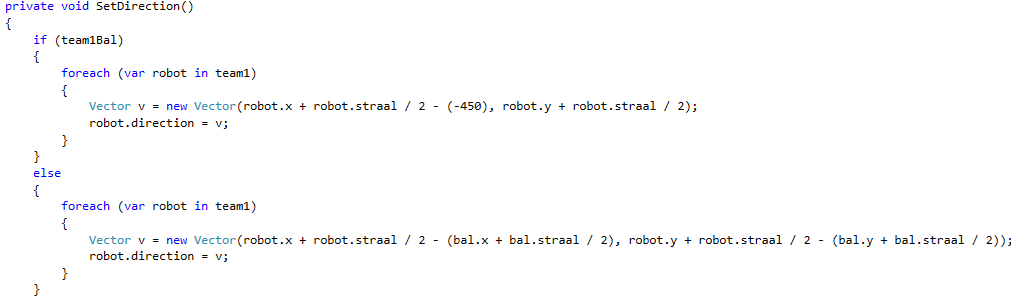


## Render-Loop

De Render-Loop wordt zoals eerder uitgelegd gebruikt om de Display te ondersteunen. Verder wordt dit ook nog gebruikt om de andere delen van de GUI aan te passen. In dit geval is dit enkel nog de Score en de tijd.

## Game-Loop

In het begin van de Game-Loop wordt er een initgame methode opgeroepen. Deze methode maakt de teams aan, zet robots op een random plaats in hun deel van het veld en zet de bal op het centrale punt. Deze methode wordt ook telkens na een doelpunt opnieuw opgeroepen. Een laatste methode die nog wordt opgeroepen is de setDirection-methode. Deze methode is zeer belangrijk doorheen de Game-Loop. De methode zegt dat als de robot niet in het bezit is van de bal hij naar de bal moet gaan. Indien de robot wel in het bezit is van de bal gaat hij naar her doel van de tegenstander.



Indien er een doelpunt wordt gemaakt wordt de score aangepast en start er een nieuwe game.

Een ander belangrijk deel is de Collision-detection: hoe de robot zich gedraagt als ze tegen een muur botst. In deze methode wordt steeds opnieuw gekeken of de robots niet overlappen met elkaar. Indien dit zo is wordt er van snelheid gewisseld.

## Timer

De timer wordt gebruikt om de Loops in gang te houden.

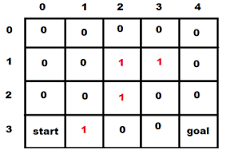
# Game AI

## Voorstudie

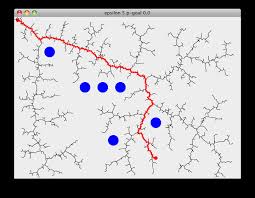
Game AI is een veelgebruikte applicatie in niet alleen Game maar ook in andere applicaties. Dit wordt typisch gebruikt voor zowel random als tactische beslissingen in de applicatie. Hierdoor neemt het een groot deel van de werklast van de programmeur of de gebruiker af.

In het geval van dit project wordt er gewerkt naar een Path-planning algoritme. Dit is een algoritme dat naar verschillende routes zoekt tussen 2 punten. Aan de hand van het gevonden pat kan de AI hierna beslissingen maken.

Er zijn 2 vormen van Path-planning algoritmen: een map-grid based algoritme waarbij men vaste vakken in een grid heeft waar men naartoe gaat om naar het eindpunt te gaan.

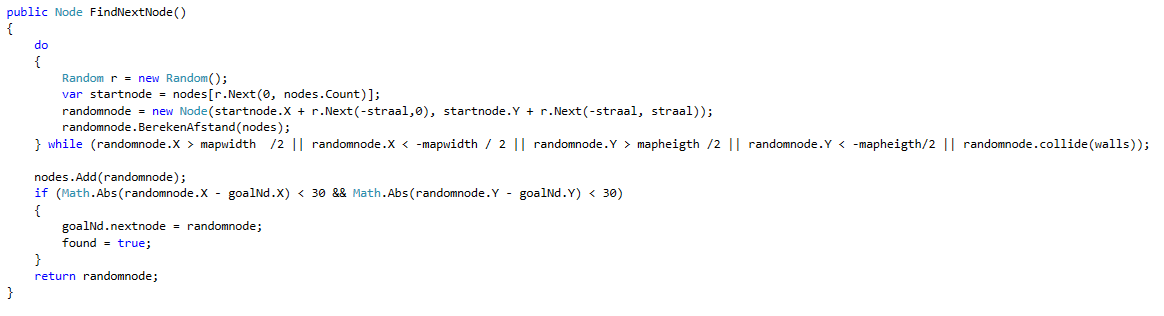


Hiernaast is er ook nog een Sampling gebaseerde path planning. Dit laatste wordt gebruikt in dit project onder de vorm van Rapidly exploring Random Trees (RRT). Hierbij word de zoektocht van het begin naar het eindpunt niet beperkt door vakken in een grid. In RRT wordt er steeds gesampled naar niet ge-exploiteerde delen van de map, de “tree” kan zo blijven verdergingen tot het eindpunt gevonden wordt.



## Implementatie

In het programma zelf wordt een FindNextNode methode gebruikt die vanaf een random node een nieuwe node zoekt om zich daarmee vervolgens te verbinden, indien die node aan de juiste regels houdt. Zoals niet in de muren gaan of de muur overlappen.



Nadat een nieuwe node is gevonden word die direct getekend, indien deze node binnen een bepaalde afstand van de ‘goal’ is wordt de hierop volgende node de goal. Hierna wordt de FindNextNode methode niet meer opgeroepen en wordt het pad getekend.

# Bronnen

[1] <https://stackoverflow.com/questions/345838/ball-to-ball-collision-detection-and-handling>

[1] https://www.dotnetperls.com/maze