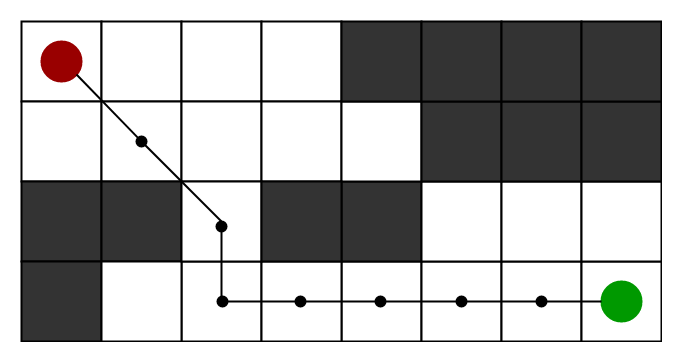
Koen Elderhorst

algoritmiek

Koen Elderhorst, Joeri de Wagt

Groep 26



Inhoudsopgave

[Inleiding 2](#_Toc31401746)

[Probleemstelling 2](#_Toc31401747)

[Aanpassingen 2](#_Toc31401748)

[Resultaten 3](#_Toc31401749)

[Conclusie 5](#_Toc31401750)

[Bronnen 6](#_Toc31401751)

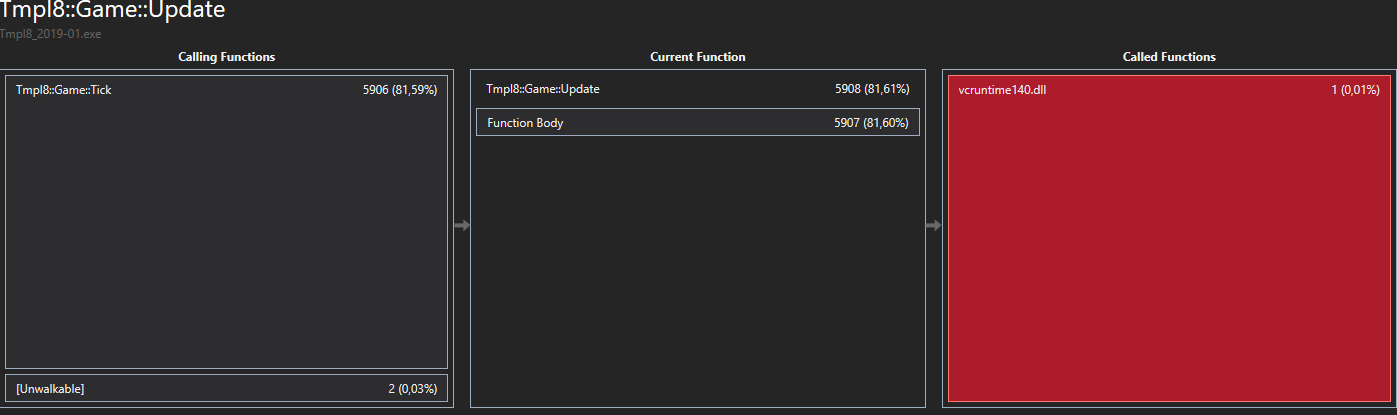
# 

# Inleiding

Voor het project deze periode hebben wij de opdracht gekregen om een oorlogssimulatie te optimaliseren, zodat hij zo snel mogelijk is. Dit hebben we gedaan door algoritmes te vervangen of aan te passen, en door gebruik te maken van *threads*. In dit verslag staan de veranderingen aan de algoritmes gedocumenteerd. Dit is gedaan door middel van *screenshots* en uitleg van de veranderingen.

# Probleemstelling

Het probleem met de begin situatie was dat er veel gebruik werd gemaakt van *nested for-loops*, dit betekent dat er *for-loops* in *for-loops* zaten. Zo checkt elke tank de afstand tussen zichzelf en alle andere tanks, om dan nog te checken er geladen is. Er staan op meerdere plekken in de oorspronkelijke code zulke *nested for-loops* constructies. Zoals te zien is in de screenshot hieronder, nemen de update en tick functies een erg hoog cpu percentage op zich.



Figuur 1, CPU gebruik van de Update functie en Tick functie

# Aanpassingen

De aanpassingen die door ons zijn gemaakt zijn gedaan om de n^2 functies te versnellen, zoals het algoritme om de dichtstbijzijnde vijand te vinden. Dit hebben we gedaan door een *Grid* toe te voegen. Op deze manier kunnen we per tank alleen zoeken in de *cell* waarin de tank zich begeeft. Hierdoor wordt de lijst tanks waar elke tank doorheen moet zoeken drastisch verkleind.

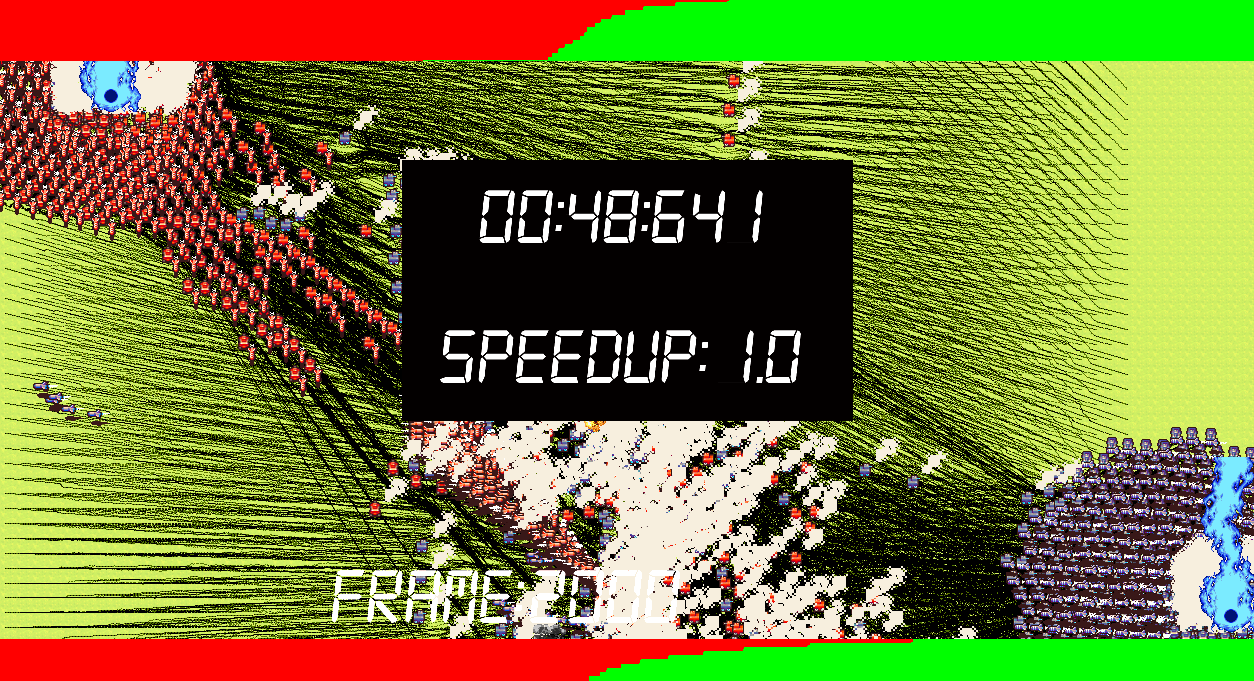
Deze aanpassing is ook vertaald voor de update methode van de *particle beams* en de *rockets*. De *rockets* hebben al een positie attribuut en kunnen daarom dezelfde functies gebruik die tanks gebruiken om de huidige *cell* te bepalen. De *particle beams* staan echter niet op één positie, maar bezitten een oppervlakte. Om deze reden is er voor de *particle beams* een aparte functie geschreven om de huidige *cells* te bepalen.

# Resultaten

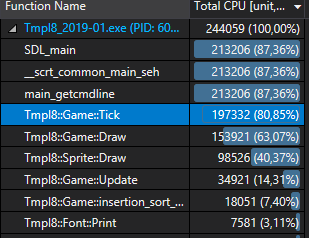


Figuur 2 beginsituatie project code

Zoals op de screenshot hierboven is te zien is de begin stand van de applicatie, dus zonder enige aanpassing, een tijd van 1:24. Door de applicatie van een *grid* te voorzien en de tanks door middel van dit *grid* een kleinere lijst kunnen laten doorzoeken is een tijd van 0:48 behaald.



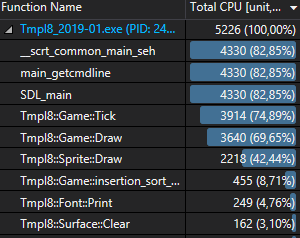
Figuur 3 Speedup door middel van de grid



Figuur 4, CPU gebruik voor toepassen van Grid

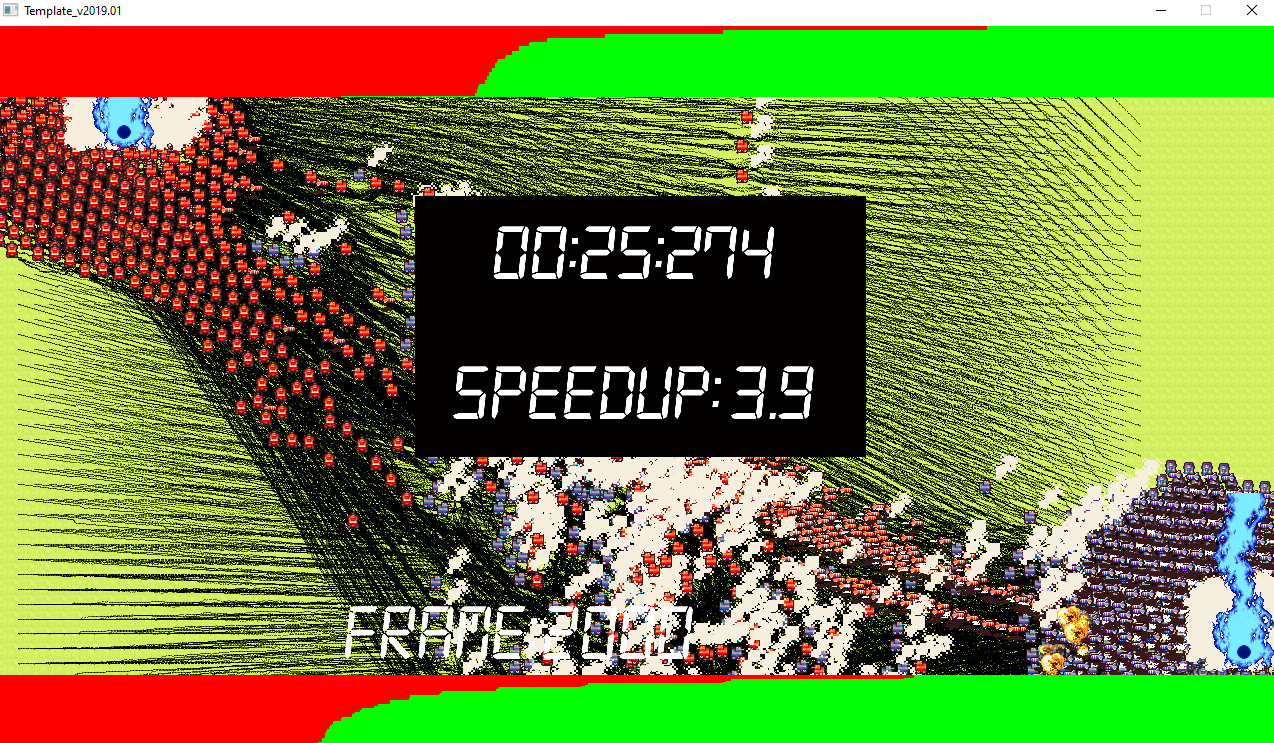
Zoals al eerder vermeld is, hebben de functies een big-O van n^2. Door implementatie van de grid is deze big-O niet veranderd. Elke tank checkt nog steeds de afstand tussen elke andere tank uit de lijst. Het verschil zit hem in de grootte van de lijst, waar de grootte eerder standaard 2558 was, is dat nu rond de 150 per *cell*. Het verschil dat dit in cpu gebruik oplevert is te zien in de afbeeldingen hierboven en onder.

Figuur 5, speed up na het implementeren van een grid.



Figuur 6, CPU gebruik na implementeren van grid.

Dit verschil van 36 seconden kan nog vergroot worden door dit idee ook te implementeren op de *particle beams* en de *rockets*. Door ook bij deze algoritmes gebruik te maken van de *grid* kan een grotere speedup verkregen worden.



Figuur 7 speedup na gebruik grid bij zowel rockets en particle beams als tanks

# Conclusie

Uit de verkregen resultaten is te concluderen dat door middel van een *grid* toe te passen, de hoeveelheid data die berekend moet worden aanzienlijk kleiner kan worden. Dit is mogelijk omdat er goed wordt bepaald welke data gebruikt moet worden in de berekeningen en daarom op een zeer efficiënte manier overbodige data links kan laten liggen. Dit zorgt ervoor dat de rekenkracht voor nuttige berekeningen gebruikt kan worden. Dit draagt bij aan een versnelling van de applicatie.

# Bronnen

Arnold, B. (2015, 4 februari). *Advanced C++/Graphics Tutorial 41: Spatial Partition Pt. 1*. Geraadpleegd op 17 januari 2020, van <https://www.youtube.com/watch?v=WG3Sl3m4rNs>

Arnold, B. (2015, 4 februari). *Advanced C++/Graphics Tutorial 41: Spatial Partition Pt. 1*. Geraadpleegd op 17 januari 2020, van <https://www.youtube.com/watch?v=RN1GRX2ByLM>