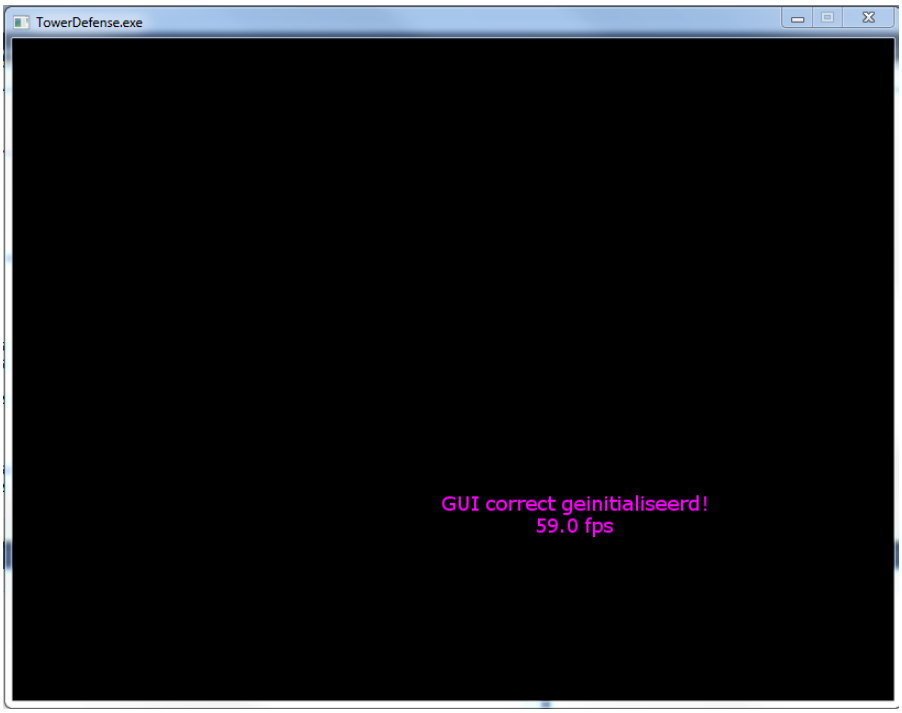
Project SOI: implementatie walkthrough

Om jullie wat op weg te helpen, geven we in dit document aan hoe de Tower Defense opgave stap voor stap kan uitgewerkt worden. Wat volgt is bedoeld als een hulpmiddel en hoeft zeker niet strikt gevolgd te worden.

1 Gui testen

Aangezien we met een externe library werken voor de grafische weergave controleer je best zo snel mogelijk of alles correct werkt. Je kunt de gui controleren door een test game loop op te starten door de RUN_TEST macro constante op 1 te zetten. Bij uitvoeren zou je dan het volgende scherm moeten zien.



2 Datastructuren

De gevraagde datastructuren HashSet en PriorityQueue zijn een noodzakelijke voorwaarde voor de implementatie van pathfinding. Daar dit een cruciaal onderdeel van het spel is, raden we jullie aan hier zo snel mogelijk mee te beginnen.

Het is aangewezen om te beginnen met HashSet daar deze eenvoudiger kan geïmplementeerd worden (enkel aaneengeschakelde lijsten, terwijl er in de PriorityQueue gebruik gemaakt wordt van dubbel aaneengeschakelde lijsten).

3 Pathfinding

Eens de datastructuren geïmplementeerd en grondig getest zijn, dan kan gestart worden met de implementatie van pathfinding a.h.v. het A* algoritme. Zorg dat je het algoritme begrijpt voordat je begint aan de eerste regel code.

TIP: denk goed na over hoe je de Nodes van het speelveld in het geheugen zal bijhouden (en let er op dat de bezochte Nodes die uiteindelijk geen deel zullen uitmaken van het Path, ook correct opgeruimd worden).

4 Spawnen van vijand

Pathfinding dient dan uiteraard uitgebreid getest te worden. Een goede manier is om gewoon een vijand te laten spawnen en deze naar het castle te laten lopen. We hebben twee debug-functionaliteiten voorzien die jullie hierbij kunnen helpen: de macro constante SHOW_PATHFINDING op 1 zetten zorgt ervoor dat de paden die berekend zijn voor de vijanden op het scherm getekend worden, de macro constante SHOW_GRID op 1 zetten zorgt ervoor dat de indeling van de spelwereld in tiles visueel voorgesteld wordt. De eerstgenoemde functionaliteit is reeds geïmplementeerd, de laatst genoemde moeten jullie zelf implementeren. (wordt uitgelegd in de opgave bij de functies die de wereld renderen)

De bewegingscode voor de vijanden dient aangevuld te worden in de functie update_movement in game.c. De Nodes in het toegekende Path kunnen eigenlijk gezien worden als waypoints en de bewegingscode voor vijanden bestaat er dan in om deze van waypoint naar waypoint te laten lopen, rekening houdend met hun snelheid.

5 Plaatsen van torens

Eens pathfinding werkt en vijanden kunnen gespawned worden en zich voortbewegen richting Castle, dan kunnen we beginnen aan het plaatsen van torens aangezien er een afhankelijk bestaat tussen beide. Een toren kan namelijk niet geplaatst worden als dit ervoor zou zorgen dat er geen geldig pad meer is van Spawn naar Castle. Aan de andere kant moeten bestaande paden vernieuwd worden bij het plaatsen van een toren, aangezien de optimale route op dat moment gewijzigd kan zijn.

We stellen voor om in het begin enkel te focussen op machine gun towers. Hiertoe dient een stuk interactie met de gui geprogrammeerd te worden (zie daarvoor de opgave). Belangrijk voor deze fase is dus dat je niet vergeet de paden die momenteel gebruikt worden door vijanden te herbereken a.h.v. de functie refresh_path in path.h/c.

6 Targeting/Projectielen

Nu er torens kunnen geplaatst worden en er vijanden over het scherm bewegen is de eerstvolgende logische stap het schietgedrag van de torens te implementeren. Focus je hier initieel ook enkel op machine gun tower met projectielen van het type bullet.

7 Uitbreiden toren/vijand types

Eens het NORMAL type vijand en torens functioneren, dan kan men dit vrij snel uitbreiden. De basis structuur van elke additionele implementatie blijft namelijk hetzelfde.

8 Waves logica

Daarna is het interessant om wat structuur in het spawnen van de vijanden te brengen door het systeem van waves te implementeren. Ofwel volg je hiervoor het basis voorstel vanuit de opgave, ofwel ga je op zoek naar een eigen systeem. Dit ondergedeelte zal de grootste invloed hebben op de uiteindelijke gameplay van het spel.

9 Spells

Het enige wat dan nog overblijft is de implementatie van de spells.

10 Afwerking

In de afwerkingfase worden vooral schoonheidsfoutjes opgelost en kan men op zoek gaan naar kleine tweaks die het spel leuker kunnen maken. Let op: schoonheidsfoutjes zijn geen memory leaks of grote bugs. Deze zouden in elke stap die je genomen hebt al moeten opgelost zijn, zo kan je de problemen beter isoleren tot een bepaald stuk van de code en is het gemakkelijker debuggen.

Begin dus zeker niet aan deze fase zolang de rest nog niet op punt staat. Deze stap is eerder voor jullie zelf en zal nog weinig invloed hebben op de uiteindelijke punten.