Technisch verslag THGA

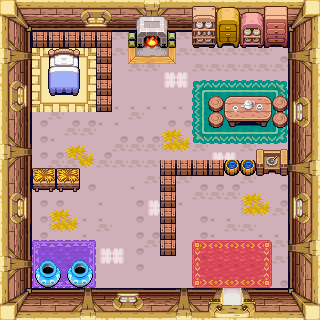
The Wonders of Mazalt

*Diego Nijboer*

*Arsalan Anwari*

*Coen Schoof*

*Leon Zhang*

**

## 

## **Inhoud**

[**Inhoud**](#_m18hjw9oxewt) **2**

[**Inleiding:**](#_m64io78d4o8l) **3**

[**Methode:**](#_108tq0vbt2lt) **4**

[Instelling gebaseerde klasse parameters](#_x8mffron1ylo) 4

[Verplaatsing van een speler met collision en ingangen](#_7lg6cq1l0hkp) 4

[Het afvuren van projectielen met hit detection](#_l50i3r8pq6xk) 5

[Game state manager](#_ppjvdmvgyhr) 5

[Map generator](#_bdtqbp30ty2k) 5

[Enemy AI:](#_7nf7863rbh07) 6

[Music player (voor geluidseffecten en achtergrondmuziek)](#_vx4md2ahnfon) 6

[Camera view voor de speler](#_13aau2yp0x3w) 6

[**Toelichting**](#_ds3f9wqwp1cc) **7**

[Instelling gebaseerde klasse parameters](#_sb0irm14us78) 7

[Verplaatsing van een speler met collision en ingangen](#_nwvm061die4k) 7

[Afvuren van projectielen met hit detection](#_4aw9r1tr7vjy) 7

[Game state manager](#_3adgovtgtnpj) 7

[Map generator](#_a8g6dx3nr3mn) 7

[Enemy AI](#_3ho3ozq3zopt) 8

[Music player (voor geluidseffecten en achtergrondmuziek)](#_6s4benitadk5) 8

[Camera view voor de speler](#_60mh0n1qau4a) 8

[**Conclusie**](#_axqz3uqkvaz2) **9**

## 

## 

## 

## 

## **Inleiding:**

De themaopdracht Gaming heeft als leerdoel om te werken met C++ en SFMLterwijl je de scrum methode toepast. Dit hebben wij onder andere gedaan door korte sprints op te zetten waarin we de taken hebben verdeeld. Op deze manier kunnen wij tussendoor evalueren of we tevreden zijn met de vooruitgang en wat we eventueel aan moeten passen om het doel te realiseren.

Wij hebben gekozen om een 2D topdown RPG game te maken gebaseerd op een tile based systeem. De game is geïmplementeerd aan de hand van een 2D game engine genaamd “DWARF”. Dit is een library met SFML die wij zelf hebben geschreven en die extern naast de game code gebruikt kan worden.

Een groot voordeel van de library is de mogelijkheid om een .csv bestand als een factory pattern om te zetten naar game data die gebruikt kan worden in een implementatie. De library is verder volledig voorzien van:

* instelling gebaseerde klasse parameters
* verplaatsing van een speler met collision en ingangen
* afvuren van projectielen met hit detection
* game state manager
* map generator
* enemy AI
* music player (voor geluidseffecten en achtergrondmuziek)
* camera view voor de speler

## 

## 

## 

## **Methode:**

### Instelling gebaseerde klasse parameters

De library geeft de gebruiker de mogelijk om een instellingen lijst mee te geven aan verschillende objecten die deze lijst dan weer omzetten naar parameters. Wij gebruiken een C-style struct genaamd “options” als lijst. Vervolgens hebben wij een hiërarchie gemaakt met namespaces zodat een object onderscheid kan maken tussen de verschillende structs met dezelfde naam.

|  |
| --- |
| **namespace** settings **{**  **namespace** music\_player **{**  struct options **{**  const std**::**string song**;**  float volume **=** 100**;**  float pitch **=** 1.0**;**  bool loop **=** **true;**  **};**  **};**  **};** |

### Verplaatsing van een speler met collision en ingangen

Een map kan toegewezen worden aan een speler. Deze map is echter uniek en kan niet veranderen. Als dezelfde speler dus moet voorkomen in meerdere maps, dan moet een nieuwe speler worden toegewezen met vergelijkbare instellen (maar met een verschillende map). Elke map bevat een 2D struct array waar het volgende in bekend is:

* tile id:, level nr, positie, middenpunt en texture pad.

Tijdens het genereren van de map worden bepaalde waardes worden toegewezen aan het “tile id” en het “level nr”. De speler begint met een (aangegeven) startpositie op de map en er wordt bij elke beweging gekeken of de volgende positie een geldig “level nr” bevat waar de speler op mag lopen (0 is geldig, 1 niet). Als dit volgende positie een special “level nr” is (bijv een doorgang) kan dit worden gecontroleerd met een functie.

De positie van de speler wordt constant aangepast op basis van de input van de gebruiker. aan de hand van deze positie kan er bepaald worden welke element uit de 2d struct array gebruikt moet worden en wordt het bijbehorende pixel coordinaat meegegeven.

### 

### Het afvuren van projectielen met hit detection

Als de gebruiker op spatie drukt, wordt er in een loop 10 keer een sprite aangepast en:

* gechecked of de sprite collision heeft met andere objecten op de map ( sprite komt op een positie waar het “level nr” gelijk is aan 1)
  + zo ja, dan stopt de loop en speelt er een hit sound effect
* gechecked of de positie van de sprite en de enemy gelijk zijn
  + zo ja, dan stopt de loop en verliest de enemy een leven

Als de loop helemaal voltooid is wordt er een andere sound effect afgespeelt.

### Game state manager

Met de game state manager is het mogelijk om meerdere fasen van het spel toe te kennen. Wij hebben dit gebruikt om bijvoorbeeld van het hoofdmenu naar het introductieverhaal te wisselen. Het gebruik hiervan is eenvoudig:

|  |
| --- |
| state**{**  7**,** 7**,**   std**::**string**(**"main map Game state"**),**  std**::**vector**<**\_event\_**>{**  \_event\_**{**  4**,** "Entranced entered"**,**  **[&**player0**]()->**bool **{** **return** player0**.**is\_entrace**();** **},**  **[&**player1**]()->**void **{** player1**.**move\_down**();** player1**.**move\_down**();** **}**  **}**  **},**  **[&**player0**]()->**void **{** player0**.**start**();** **}**  **},** |

We maken dus eerst een state aan, en het eerste wat we aangeven is in welke state we zijn en wat de volgende is. In die states kun je dan events aanroepen.

### Map generator

Wat de map generator eigenlijk inhoudt, is dat hij op basis van het csv bestand een grid over het plaatje van de map heen plaatst. Dit grid bevat ID’s die je functies kan toekennen maar ook collision waardes bevat. Een voorbeeld van een functie die wij gebruiken is het verplaatsen van de speler naar de binnenkant van een gebouw, maar dit is toepasbaar voor vele features. Vervolgens heeft het grid dan ook coordinaten waar je objecten overheen kunt plaatsen.

### Enemy AI:

De opzet van de AI is vrij simpel. In een extern object dat gebruikt wordt door de speler wordt periodiek gekeken of de speler binnen een bepaalde range van de enemy zit. Zodra de enemy de speler heeft ontdekt begint zijn aanvals procedure. Dit procedure bestaat uit een simpel algoritme dat aan de hand van de positie en de richting van de speler bepaalde in welke van de 4 zones hij moet spawnen. deze zones zijn: (links boven, rechts boven, links onder, rechts onder). Zodra de enemy in deze zone zit wordt er gekeken of de speler dichter qua x of y coordinaat zit. Aan de hand van deze laatste beslissing wordt de richting van het afvuren bepaald

### Music player (voor geluidseffecten en achtergrondmuziek)

De music player gebruikt ingebouwde functionaliteiten van SFML Music en Sounds. De music player houdt echter ook bij welke toestand hij zich verkeerd met een boolean waarde die wordt aangepast bij verschillende functie aanroepen

### Camera view voor de speler

SFML heeft een ingebouwde functie om de camera een object te laten volgen, wij hebben het toepassen daarvan alleen iets simpeler gemaakt door de size, positie en view al aan te geven. Ook kan het middenpunt steeds worden aangepast via een functie.

## 

## **Toelichting**

### Instelling gebaseerde klasse parameters

Met onze persoonlijke ervaringen, waren we het alle gezamelijk eens dat het vrij vervelend kan zijn om een overzicht te krijgen in klassen parameters die geërfde zijn van een ander object. Daarom leek het ons handig om een simpele struct met instellingen mee te geven die een ander bestand aangemaakt kunnen worden.

### Verplaatsing van een speler met collision en ingangen

Ons eerste idee was om verschillende objecten met textures in een JSON formaat op te slaan in een bestand en dit bestand te verwerken met een factory pattern. Deze methode leek ons echter heel omslachtig en minder herbruikbaar dan onze definitieve aanpak (tile based). Het voordeel van deze aanpak is dat een csv bestand dat gegenereerd wordt door meeste sprite editors (zoals Tiled) heel makkelijk om te zetten is naar een 2d array. Het enige probleem dat we hiermee hadden was dat we moeilijk konden onderscheiden welk coordinaat een object bevat met een speciale actie. Hiervoor leek een 2d struct array met meerdere waardes ons een beter idee.

### Afvuren van projectielen met hit detection

Het dynamisch teken van een projectiel en wachten op user input via twee individuele threads leuk ons de meest efficiënte manier om deze functionaliteit te implementeren. We hadden echter het probleem dat SFML OpenGL based is wat er voor zorgt dat er altijd maar een thread naar het scherm kan schrijven. We hadden een poging gedaan om dit op te lossen met een mutex en flag, maar dit was in de meeste gevallen te onvoorspelbaar. Daarom hebben we ervoor gekozen om deze twee taken na elkaar te laten verlopen

### Game state manager

Arsalan had in de tweede sprint een (beta)versie geschreven voor een game state manager met een STL pattern. We kwamen er echter later achter dat deze manier moeilijker te implementeren was voor taken die events hebben die de naar een andere state kunnen gaan. We hebben dus uiteindelijk besloten om een functionele versie te maken waar we direct na een event een bepaalde actie kunnen uitvoeren en terug kunnen naar een andere state.

### Map generator

In onze eerste versie toonde we meerdere kleinere sprites van de map in de hoop dat we in de toekomst de implementatie konden aanvullen zodat er alleen bepaalde delen geladen werden op basis van de positie van de speler. Hiermee dachten we runtime efficiënter te werken. We kwamen er echter snel achter dat het aanmaken van een aantal sprites de runtime verslechterd en hebben besloten om de gehele map in zijn geheel te tonen.

### Enemy AI

Bij het maken van de enemy AI hadden we het doel om het simpel maar leuk te houden. De enemy moest leuk zijn om tegen te vechten maar niet complex om te implementeren. Ons eerste idee was om met een graaf berekening de AI het kortste pad te laten berekenen naar de speler. We merkte echter later dat dit te veel geheugen kosten en kwamen tot het 4 zone idee. Het 4 zone idee is zeer makkelijk te implementeren omdat alleen een paar coördinaten gecontroleerd te moeten worden en is veel geheugen vriendelijker.

### Music player (voor geluidseffecten en achtergrondmuziek)

De music library van SFML was voor onze ogen al goed, alleen hadden we het probleem dat de library geen optie had om de status van de music player te checken (althans niet in een formaat dat wij konden gebruiken). Daarom besloten om een simpele controller te schrijven die in een variabel de status bijhoudt.

### Camera view voor de speler

De camera view was net zoals de music library goed genoeg voor onze eisen. We wouden het echter alleen simpeler voor ons zelf maken door er een controller van te maken.

### 

## 

## **Conclusie:**

De implementatie van de game state manager en het genereren van de game map is onze ogen zeer efficiënt en herbruikbaar. Wij vinden het echter jammer dat we hebben gekozen om er een template klasse van te maken wat de implementatie van de speler wat complexer maakte.

Het is namelijk niet mogelijk volgens onze methode om meerdere werelden toe te wijzen aan een speler waardoor er onnodig een nieuwe speler aangemaakt moet worden per wereld. We kozen voor een template klasse om de map te kunnen opslaan in een std::array ipv een std::vector. Arsalan heeft niet veel goede ervaringen met vectoren en wou liever met arrays werken.

Als wij toch besluiten om met vectoren te werken, dan zou de implementatie van de game map wat minder efficiënt worden, maar wel een stuk meer herbruikbaar.

Wij hebben gezamenlijk besloten om de volgende keer minder aandacht te besteden aan de efficiënte van de code en meer aan de effectiviteit en herbruikbaarheid van de code.

## 