Studentennummer: s0171059
Naam: Dagrain
Voornaam: Miguel
Jaar: 2018

Roadfighter Design

De gemaakte designkeuzes:

- Het opdelen van de code in 3 delen de **app** (alles wat specifiek is aan deze app), de **logic** (alles wat met de inherente logica van de werelde enz. te maken heeft), de **gui** (alles wat met het voorstellen van objecten te maken heeft).

- Grotendeels overnemen van de voorgestelde klasse hiërarchie.

- De factory klasse uit de game logic laten als zowel de klasse (Interface) dat interactie heeft met de wereld (World).

- De logica in een library steken, maar ook de representatie.

- Elke map die de code van een library bevat opsplitsen in include (include files) en src (source files). Om installatie makkelijker te maken, nu ik weet dat dit hier overbodig is maar ik zie het als een goede gewoonte.

- Er is een kleine aftakking van de Road klasse om een speciaal afgeleide hiervan Finish aan te maken.

Voor enkele waarvan de redenering niet zo voor de hand liggend is zal ik even wat extra uitleg geven. Zoals bv. de eerste en de derde designkeuze.

Het opdelen van de code in drie onderdelen en niet gewoon de main tussen de files van de gui of logic gooien. Dit is gedaan omdat de main en enkele andere files niet inherent zijn aan de logic library er is namelijk wel interactie met de wereld. Maar er moeten ook verwijzingen gemaakt worden naar de voorstelling van het spel. Bijgevolg zou men deze files dan gewoon tussen de code van de representatie kunnen zetten. Slechts deze redenering botst met het feit dat de interactie van deze bestanden dan toch nog iets verder gaat dan enkel de representatie waarvoor de graphisch onderdeel van de code bedoeld is.

Bijkomend, en dit argument is met betrekking tot de vierde designkeuze, ik wilde niet enkel de logic in een library maar ook de voorstelling, elke file die dan specifiek is aan deze applicatie en gebruik maakt van de libraries gaat dus in een aparte map specifiek voor deze applicatie.

Nu als men er zou op gelet hebben kan men zien dat ik zeg de factory klasse uit de logic te laten maar niet de observer. Het grote verschil hiertussen zie ik is dat de observer specifiek objecten observeert puur op basis van de logica. De factory daarentegen, zul je merken als je mijn code er op na leest, maakt objecten aan bedoeld voor representatie (SFML objecten), anders zouden we ze niet kunnen tekenen.

Factory heb ik ook niet tussen de grafische representatie gezet, omdat het een design pattern was vereist voor deze applicatie, niet betekende dat men in een andere applicatie gewoon de constructors zou willen aanroepen en dan met de library onnodige code importeert.

Ook even een korte toelichting voor de aftakking van het laatste wegstuk de Finish. Als we hier geen aparte klasse voor aanmaken dan kan de Player de finish niet herkennen. (Collision detection met afleiden van de klasse waarmee we botsen).

Als laatste nog de argumentatie waarom ook de representatie in een library werd gestopt. Het meest voor de hand liggende is al de gemakkelijke herbruikbaarheid van de code en het doorgeven. Het andere is dat het ook het onderscheid tussen de voorstelling van objecten en deze applicatie duidelijker maakt.

Indien ik hiervoor niet geopteerd zou hebben, zou het verschil minder duidelijk zijn. Daarnaast zou het de Cmakelist waar de applicatie wordt gemaakt zeer slordig hebben gemaakt.

Ik genoeg verduidelijking was bij de gemaakte designkeuzes. En de rest vanzelfsprekend kan beschouwd worden.