Ward Gauderis
Gevorderd Programmeren
2de bachelor Informatica
UAntwerpen

Verslag Project Gevorderd Programmeren 2019 – 2020

In dit verslag overloop ik feature per feature de structuur van 'Space Qubes' en verklaar ik de redeneringen achter mijn aanpak.

Model-view-controller pattern
Vermits het project doordrongen is van dit patroon, is dit het eerste punt. Er zijn online meerdere definities van het MVC-patroon te vinden die verschillen op allerhande vlakken. Ik heb me strikt gehouden aan één definitie die me het meest logisch klonk. Dit hielp me onderscheiden welke data en operaties zich in welke tak van MVC moesten bevinden.
Het model mag niets afweten van view of controller en bevat alle data die essentieel zijn voor de toestand van het spel (als het spel moet worden opgeslagen, moet enkel het model worden opgeslagen). De view heeft enkel weet van het model en past zich aan het model aan. Er kunnen verschillende views zijn per model. De library die de view gebruikt, is ook verantwoordelijk voor de user input en dus geeft de view deze input door aan de controller die erop handelt. De controller zelf kent het model en de view, want deze moet naargelang de binnenkomende input uit de view het model eventueel aanpassen.
Ik heb er dan ook voor gezorgd dat de SFML library nooit voorkomt in de controller of het model. Daarom heb ik mijn eigen Vector, Event en Key classes gecreëerd. Als het spel dan zou moeten werken met een ander I/O library, dan moeten enkel de view classes aangepast worden.

Polymorphism
In elke tak van het MVC-patroon inherit elke entity van de abstracte Entity class (voor die tak). De meeste entities moeten kunnen colliden en hebben een fysieke representatie nodig en inheriten daarom van de PhysicalEntity class.

Game class
De enige class die niet is opgesplitst in MVC (buiten de utilities) is de overkoepelende SpaceInvaders class. Deze creëert de MVC-werelden en update ze in de eventloop. Er is geprobeerd om zoveel mogelijk code en logica af te schuiven naar de andere classes zodat deze algemene class overzichtelijk blijft. Zo wordt bijvoorbeeld de volledige timing van de eventloop afgeschoven naar de Stopwatch.

Observer pattern
Elk model Entity in het MVC-patroon inherit van de Subject class en verwittigt zijn observers na een update in het model. Elke view class inherit van de Observer class en voorziet een functie die wordt uitgevoerd na een update in het model.

Singleton pattern
De utility classes Transformation, Stopwatch en Assets werden geïmplementeerd met het singleton patroon. Transformation converteert pixel coördinaten naar het coördinatenstelsel dat in het model wordt gebruikt. Stopwatch bepaalt wanneer model, controller en view moeten worden geüpdatet zodat het model en de controller updaten tegen een vaste frequentie en voorrang krijgen op de updates van de view indien het spel te zwaar wordt voor de hardware. Assets is ook een singleton zodat de verschillende textures, fonts en muziek niet verschillende keren in het geheugen zouden worden geladen.

Json levels
Het spel blijft lopen totdat de game over is voor de speler of totdat er geen waves meer komen (de speler wint). Elke wave van vijanden is gespecifieerd in een json file. Zolang er correct benaamde json files te vinden zijn in de data folder, zullen de waves blijven komen. In een wave file kunnen er verschillende eigenschappen van een wave beschreven worden (of niet, ze hebben allemaal defaults). Hiervoor verwijs ik naar de readme.

Exception handling
De meeste exceptions kunnen optreden bij de I/O. Daarom heeft de model class Wave (verantwoordelijk voor het inlezen en creëren van de waves van vijanden) verschillende checks op foutieve waardes, evenals sommige setters van models die ingelezen kunnen worden. Het inlezen van de assets gebeurt via de singleton Assets class die ook throwt indien bestanden niet geopend kunnen worden.

Factory method pattern
In de model class Wave, is er een factory method, genaamd parseEnemy, aanwezig die als input een json tree object neemt en een correct Enemy model (of een child ervan) teruggeeft.

Namespaces
De code van het spel bevindt zich volledig in de namespace SI, die ook is opgedeeld in de namespaces model, view, en controller. Elke Entity is (indien nodig) gepresenteerd in elke namespace. Daarnaast bestaat er ook de namespace utils met daarin classes die ook buiten het spel bruikbaar zijn.

Smart pointers
De eigenschappen van smart pointers kwamen erg van pas in het MVC-patroon en observer patroon. De verschillende connecties tussen model, view en controller kunnen eenvoudig uitgedrukt worden door middel van weak pointers, evenals de connecties tussen observer en subject. Ze werden pas echt handig bij het toevoegen of verwijderen van MVC-Entities uit de MVC-werelden van binnenuit. Wanneer een reeds bestaande model Entity een nieuw model wil toevoegen aan de wereld, maakt deze het model aan en op het einde van elke game tick vraagt de game op of er nieuwe entities moeten worden toegevoegd. Een smart pointer naar het nieuwe model wordt dan teruggegeven zodat de game kan beslissen of hier een bepaalde view en controller moeten worden voor toegevoegd.
Wanneer een model uit de wereld mag worden verwijderd geeft het dat ook aan, aan de model wereld. De bijbehorende views en controllers zullen dan automatisch worden verwijderd omdat zij merken via weak pointers en het observer patroon dat hun model weg is. Zo kunnen zij aan hun respectievelijke werelden aangeven dat zij ook verwijderd mogen worden.
Op deze manier moet het model niets afweten van view en controller en de view niets van de controller.

Collision detection
In Space Qubes was er geen nood aan ingewikkelde collision detection. Alle collision detection gebeurt via axis-aligned bounding boxes. Elke Entity die kan colliden, inherit van PhysicalEntity die de nodige functies en data voorziet.
De collision met de PhysicalEntity Shield is wel opmerkelijk doordat deze gebruik maakt van space partitioning. Opdat het schild geleidelijk aan zou kunnen afbreken, moet het bestaan uit zeer veel deeltjes. Elke game tick collision testen tussen elke PhysicalEntity en elk deeltje van het schild zou veel te zwaar zijn. Daarom werd de functie die collision test met het schild virtueel overschreven om de test op te delen in drie delen (eerst op de schaal van het volledige schild, dan op de schaal van een schildsegment en tenslotte op de schaal van één schilddeeltje).

Doxygen
In elke header file zijn er comments aanwezig in de stijl van Doxygen die verklaren wat de functies en classes betekenen. Er is ook een Doxygen-config aanwezig.

Easily extendable

Door de sterke opsplitsing in model-view-controller, is Space Qubes erg flexibel en makkelijk uitbreidbaar. Indien een nieuwe Entity wenst toegevoegd te worden, moeten immers de nieuwe MVC-classes enkel inheriten van de juiste base classes om te kunnen werken en door de inheritance is het vaak zelfs niet nodig om alle drie de classes aan te maken. Om dit aan te tonen zijn twee extra vijanden toegevoegd met speciale eigenschappen: 'ghost' en 'witch'. Die laatste vereist enkel het toevoegen van een nieuwe model class, de eerste heeft unieke visuele eigenschappen en heeft dus ook een nieuwe view nodig waarin enkele virtuele functies worden overschreven. Een ander goed voorbeeld is de view Display class die geen tegenhanger heeft in de model of controller namespace omdat het een view is dat wordt aangesloten op de player en wave models.