Report Arkanoid

Zhong-Xi Lu

January 18, 2017

1 Classes

Voor de classes, heb ik me zoveel mogelijk gebaseerd op de diagram die in de opdracht staat. Ik heb een class **Entity** gemaakt dat een basis vormt voor alle entities in de game. Deze bevat `pure virtual` methodes `update` en `draw`, m.a.w. elke entity in de game heeft deze methodes. Naast dat, heeft deze class ook andere members, zoals `position` en `size` met de nodige getters and setters. (nodig voor collision detection) Als laatst is er ook een `collidesWith` methode, die een andere entity als argument neemt en controleert of er een collision is tussen deze twee entities.

In de game logic, zitten paar 'math' classes, zoals **Random** (singleton) en **Vector2D**; de random class wordt gebruikt bij de collision tussen een player en een ball en de vector class doorheen het spel voor de positie, snelheid, ... Verder bevat deze de directe subclasses van **Entity**: **Ball**, **Player**, **Wall** en **World**.

Daarentegen in de game gui, zitten alle sfml entities die direct een subclassen zijn van de entities uit de game logic. (Bijvoorbeeld: BallSFML is een subclass van Ball) Deze sfml entities bevatten natuurlijk de sfml elementen, zoals texture, positie op het scherm, ... Ook heeft de gui, een **Transformation** (singleton) class, die pixels van het scherm vertaalt naar coordinaten in het spel.

De speciale blokken (in de game) zijn een subclass van **Block** en hebben een extra public methode die dan de *special effect* moet vormen. Bijvoorbeeld **InvisBlock** heeft een methode `effectBall` die de bal onzichtbaar maakt. Ook deze speciale blokken hebben een aparte sfml class.

Verder is er nog de abstract factory **EntityFactory** die een interface weergeeft van wat er allemaal gemaakt kan worden. Hiervan afgeleid is er de **SFMLFactory**, die zoals de naam al zegt, sfml elementen creëert voor het spel.

Heel het spel is 'gewrapt' in een class **Arkanoid**, die zowel logic als de gui elementen bevat met de sfml factory daarbij. Deze bevat één public member functie, nl. `run` die het spel zal starten. Deze class bevat zoals eerder vernoemd de SFML elementen, waaronder de game window, background, ... Intern gaat hij alle game entities maken en laadt telkens een nieuwe level in als dat nodig is. Natuurlijk is **World** een member hiervan, waardoor we de world makkelijk kunnen aanroepen om de game in goede banen te leiden.

2 Verduidelijkingen

De **World** class heeft i.p.v. één grote lijst, één lijst/veld voor elke type, d.i. **Ball**, **Player**, **Wall**, **Block**, op die manier is het gemakkelijker om entities met elkaar te vergelijken en weet men wat voor type het object is. Bijvoorbeeld bij collision detection, vertoont de bal een ander behaviour met de player dan met een wall. Ook kan men zo direct de player en de bal aanroepen. Het nadeel is wel dat men alle lijsten/velden moet doorgeven, als men een iets moet doen met all entities, zoals updaten of drawen.

In de `checkCollisions` van **World** wordt er gebruik gemaakt van `dynamic cast`'s om zo te kunnen bepalen wat voor speciaal blok het is en daarop gebaseerd de juiste actie nemen. En zoals gezegd, checkt deze methode apart de collision met de player, de walls en de blocks, aangezien we geen grote lijst hebben met alle entities.

De class **Ball** heeft een methode `bounceIfPossible` die als argument oftewel een lijst van entities (deze specifieke methode is getemplatiseerd, zodat men eender welke enitity mee kan geven) meeneemt oftewel de player (andere signatuur) en zal hiervan afhankelijk de juiste beweging maken. **Ball** heeft `invisDuration` en `speedUp` als member, deze zijn nodig om de *special effects* te realiseren.

Ik heb ook geen aparte class gemaakt voor de background, dit kon makkelijk opgelost worden door de sfml library.

3 Known bugs

Het is soms mogelijk dat de bal vast komt te zitten (om één of andere reden) in de wall en zo 'glijdt' tegen de muur, als dit gebeurt, druk dan "esc" om de positie van de player en ball te resetten.

4 Opmerkingen

De meeste dingen spreken wel voor zich en zijn te vinden in de *Doxygen* documentatie of als comment in de code zelf. Ik heb geprobeerd om de niet voor de hand liggende zaken hierboven uit te leggen.

Voor meer info over de **gameplay**, zie de *homepage* van de gegenereerde *Doxygen* html bestanden.