Kernmodule Game Development blok 1

Einddocumentatie Bob Jeltes, GDV2

Het concept

De retrogame waar ik voor heb gekozen is Tetris. Mijn keuze is hiervoor gevallen na het bedenken van de twist, omdat deze leuker was dan wat ik kon verzinnen voor andere games.

De twist

In plaats van dat de speler de Tetrino's in het speelveld laat zakken, gooit de speler de Tetrino's vanaf de zijkant in het veld.

Daarnaast vallen de Tetrino's in de afzonderlijke blokjes uit elkaar, om te compenseren voor het feit dat de blokken vrij kunnen draaien om de z-as. Wanneer een rij 10 blokken bevat, wordt de rij gecleart.

Toelichting codestructuur

Ik heb zoveel mogelijk de verantwoordelijkheden van de afzonderlijke classes proberen te verdelen om de scripts kort te houden, en heb circular dependencies expres geprobeerd te vermijden omdat dit verwarrende code op kan leveren.

Ongebruikte Design Patterns (die ik wel had willen gebruiken)

Ik heb nog helaas minder gebruik gemaakt van design patterns dan ik had gewild.

De observer had ik willen gebruiken voor de linecheckers in combinatie met een geluidseffect of iets dergelijks. Daar ga ik me later nog in verdiepen omdat ik wel inzie hoe het gebruik van events handig kan zijn bij het beheren van vele objects.

De object pool had ik willen gebruiken voor de blokken die in het spel zijn. Dit had ik willen gebruiken om de gedeactiveerde blokken weer terug te geven aan een nieuwe parent, maar daarvoor moet ik de filestructuur van mijn hele code herorganiseren en de limieten hiervan uitdenken zodat ik niet met onterecht gedeactiveerde blokken kom te zitten.

Abstract classes zijn voor mij nog steeds vaag. Dit is een leerpunt, omdat ik ook nog niet lijk te begrijpen in welke situaties het handig zou kunnen zijn. Er begint soms wel iets te dagen, maar de mogelijkheid voor implementatie in mijn game zag ik niet direct in.

Wel gebruikte design patterns

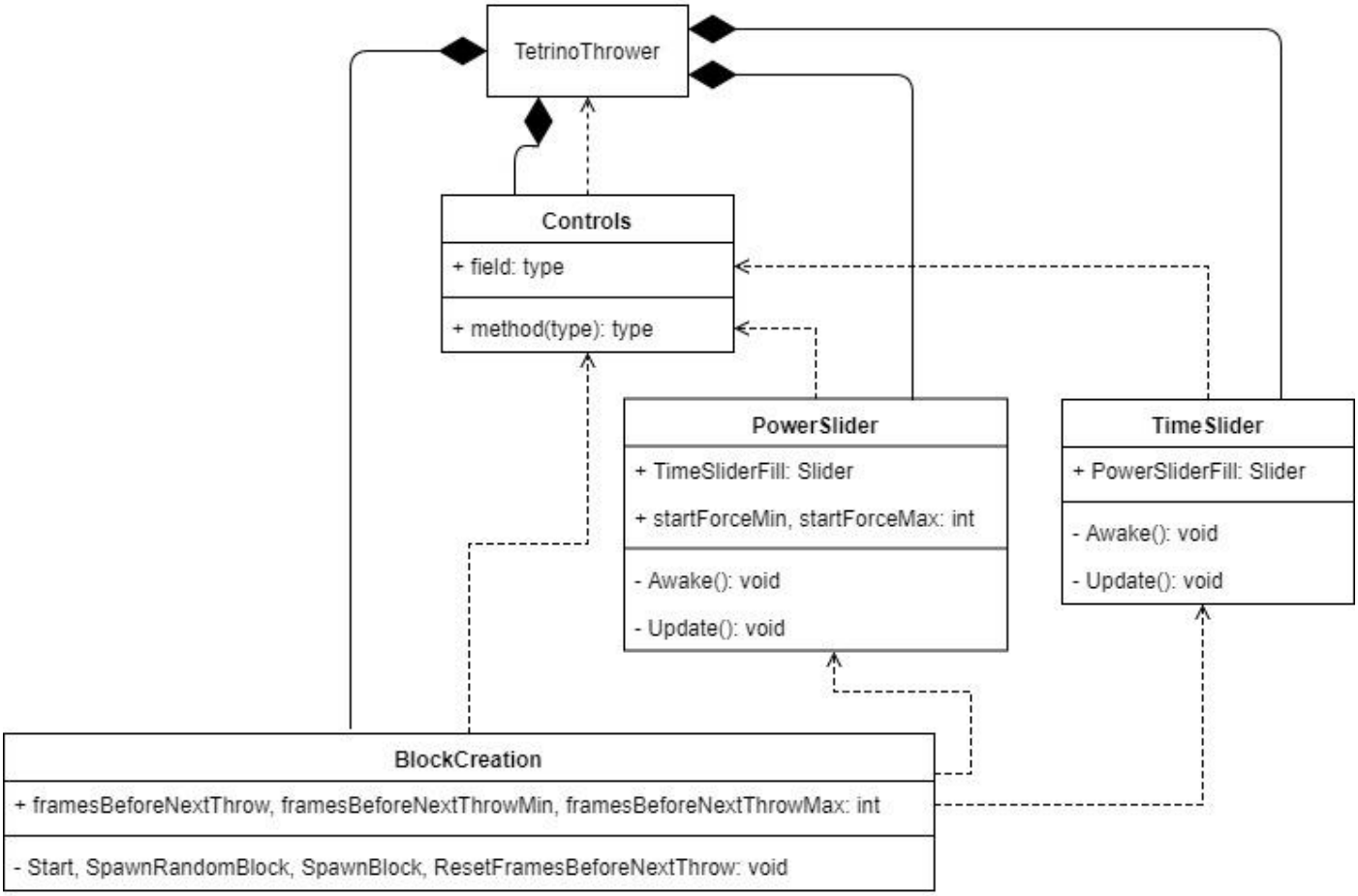
De Singleton heb ik op de absolute basis geïmplementeerd voor de GameManager. Ik had de implementatie het liefst nog verder willen uitbreiden maar ik kon niet direct iets vinden dat ik nog meer kon doen met deze design pattern. Het is wel fijn dat ik het een keer geprobeerd heb om te zien wat het nou werkelijk doet.

De builder heb ik enigszins per ongeluk gebruikt, want ik was er niet van bewust dat de manier waarop ik het instantieren van mijn objecten een design pattern was. Ik heb de creatie van de blokken afgegeven aan een aparte class, en die methodes aangeroepen vanuit de Controls class. Dit heeft erg geholpen met de overzichtelijkheid van de code in dat de verantwoordelijkheid van verschillende classes een stuk duidelijker is. Daarnaast scheelt het ook in de lengte van de classes.

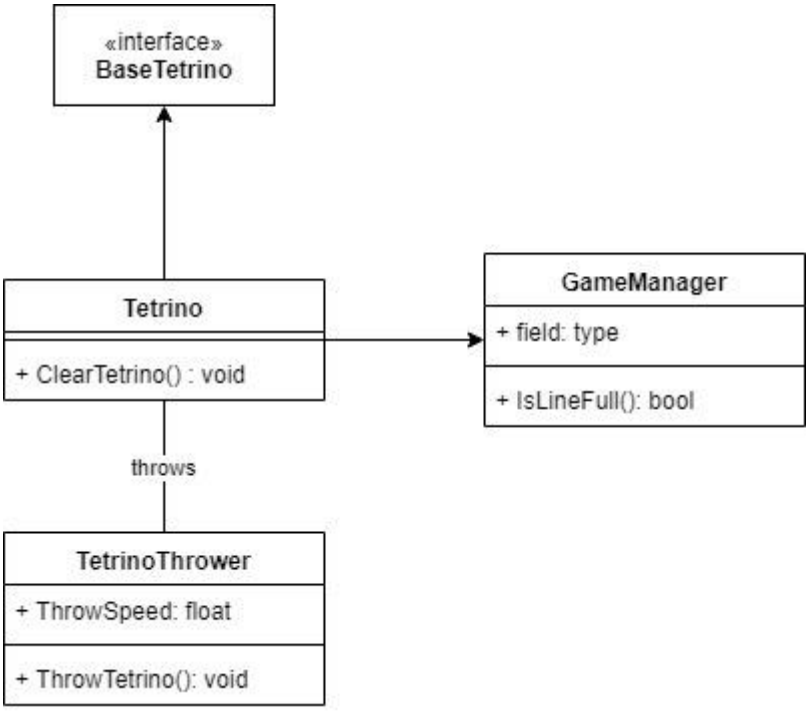
Overig

Properties, met getters en setters, vind ik nog lastig om te begrijpen wanneer ik deze het beste in kan zetten. Visual Studio vertelt mij hier en daar wel dat ik ze het best kan gebruiken, maar dan komt er in een keer 7 regels bij wat de code minder leesbaar maakt. Ik ga hier nog verder over in gesprek met betrekking tot mijn eigen code, om te zien waar ik Properties het best had kunnen toepassen.

UML-diagram van de TetrinoThrower (nieuw)



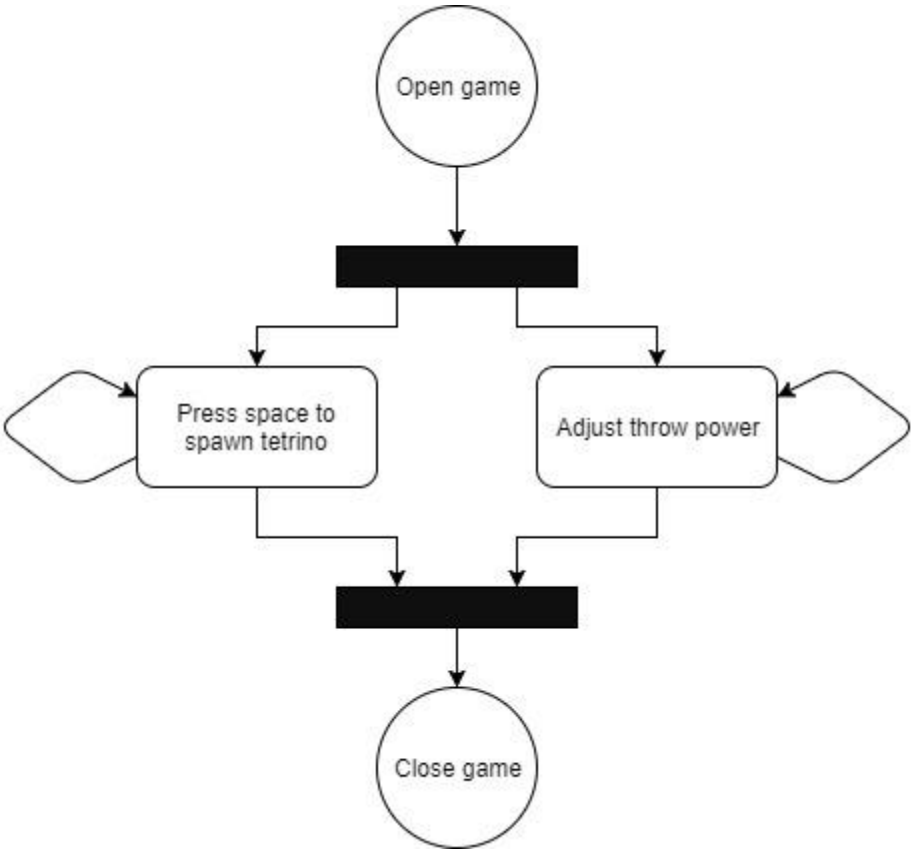
UML-diagram van de hele game (oud)



Toelichting

Er is al veel meer werk gaan zitten in de TetrinoThrower alleen dan ik dacht aan de hele game te doen. Dit blijkt ook uit het bijbehorende UML-diagram.

Activity diagram (nieuw)



Activity diagram (oud)