Technisch Uitwerkingsplan Throwtris

Bob Jeltes – GDV2

# Introductie

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.

# Design patterns

Hier een verzameling van de design patterns die ik zou willen verwerken in mijn code:

## Object pool

Er kunnen veel objecten op het scherm komen te staan, en de speler zal veel verschillende objecten het veld in gooien om een goede score te behalen. Daarom lijkt het mij een goed idee om de Tetrino’s met object pools te managen zodat er efficiënt omgegaan wordt met de resources. Gepaard met goed onderzoek zorgt dit er ook voor dat er nooit meer objecten in het spel komen dan theoretisch nodig kan zijn.

## Singleton

Er zijn minstens twee objecten die te allen tijde enkelvoudig aanwezig zullen zijn:

* GameManager
* TetrinoThrower

Hierop zou ik dan ook het Singleton design pattern willen toepassen. Het gebruik van meerdere GameManagers of meerdere TetrinoThrowers zou het spel erg kunnen veranderen, of zelfs onbruikbaar kunnen maken in het geval van twee GameManagers.  
Daar tegenover staat dat het gebruik van één enkele GameManager het hele programma overzichtelijker kan maken.

## Uitprobeerwerk met design patterns

Object pools zijn nog niet gelukt. Ik zou er meer tijd voor nodig moeten hebben om dit uit te zoeken, omdat de Tetrino’s op een bepaalde manier gevormd worden die ervoor zorgt dat de brokstukken opnieuw children moeten worden op het moment dat ze teruggeplaatst worden onder de parent.

Ik zou dit kunnen doen door van de kleuren van de blokken een variabele te maken, en ze een nieuwe kleur toewijs op het moment dat ik ze opnieuw toewijs aan een nieuwe parent. (Long block heeft een lichtblauwe kleur, Square block een paarse, enzovoort)

## 

# Engine

Ik heb voor de Unity gekozen vanwege de volgende redenen:

1. Mijn ervaring met Unity is groter dan dat met andere engines, dus kan ik sneller een goede fundatie leggen om nieuwe design patterns op uit te proberen;
2. Ik kan 2D en 3D makkelijk combineren als ik dat wil, wat bijvoorbeeld met GameMaker minder makkelijk gaat;
   1. Ik ga een 3D-omgeving gebruiken met gebruik van een 2D isometrisch aanzicht om een kleine twist te geven aan de originele look van Tetris
3. Unity werkt op basis van C#, waarop onze code conventies het meest gebaseerd zijn;
4. Physics-based games zijn makkelijk te prototypen in Unity, en er wordt veel controle gegeven over de physics;

# Hoe hangt het spel in elkaar?

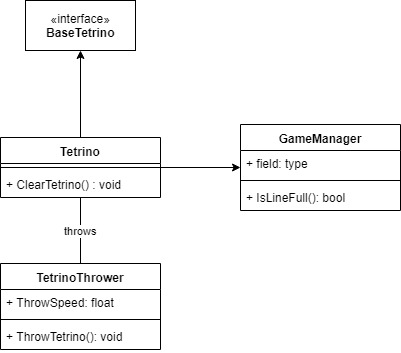
Het spel loopt voortdurend door, en maakt gebruik van de physics van Unity. Hierdoor is het spel al fundamenteel anders dan de originele Tetris, waarin blokken stukje voor stukje naar beneden zakken. Daarnaast heeft de speler ook geen controle over hoe de blokken vallen; hoe de blokken vallen is compleet afhankelijk van hoe de speler de blokken lanceert, hoe de blokken terecht komen en hoe ze uit elkaar vallen.

# Wat is de prioriteit?

Hier is een lijst van prioriteiten, van meest naar minst belangrijk

1. De code moet overzichtelijk zijn;
2. Het spel moet efficiënt met de resources omgaan – goede prestaties;
3. Het spel is een werkend prototype waarmee duidelijk wordt gemaakt wat het spel ongeveer gaat worden
4. De main mechanic moet leuk zijn en moet makkelijk op te pakken kunnen zijn.

# UML & Activity diagram



Figuur : Oude UML

# 

Figuur : Activity Diagram