Kernmodule Game Development 1

Ruben Bimmel
GDV2

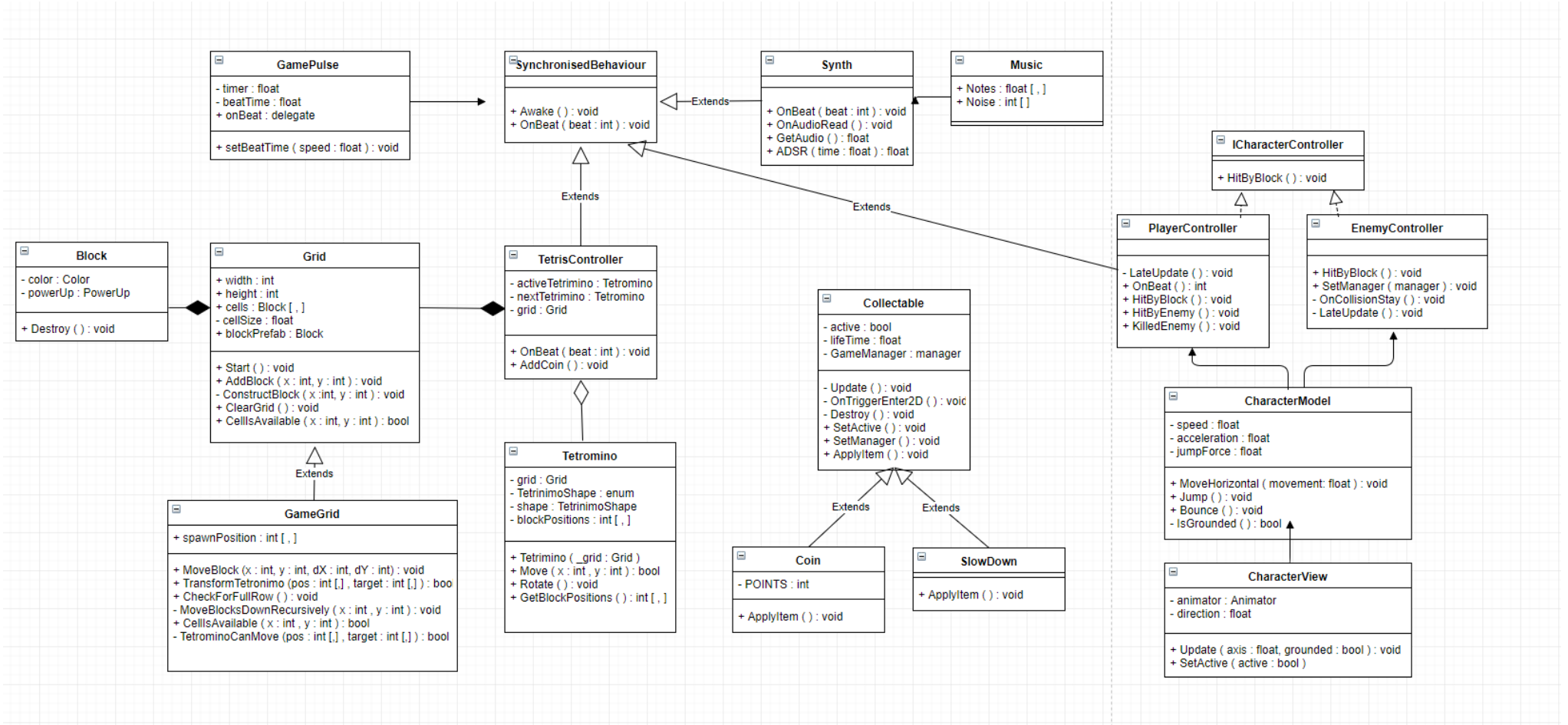
Kernmodule GDEV 1
Aaron Oostdijk
Valentijn Muijrers

26-10-2017
Versie 1

Inhoud

Project overzicht

Class diagram



Toelichting Class Diagram

Het class diagram bevat een overzicht van de belangrijkste classes en de manier waarop die met elkaar verbonden zijn. Om het overzicht te bewaren missen de GameManager en UI classes. De GameManager class bevat voornamelijk set en get functies en de classes Grid, Tetriscontroller, Collectable, PlayerController en EnemyController refereren allemaal naar deze class. De GameManager is de enige class die verbonden is met de UI Canvas class. De ander UI classes worden alleen in de andere menus gebruikt en refereren alleen naar de GameManager.

Grid en GameGrid

De Grid class is een van de belangrijkste classes in het spel. De class bevat een 2 dimensionale array waarin de blokken geplaatst kunnen worden. De classes zijn zo geschreven dat alleen Grid classes blokken (in hun eigen grid) kunnen verplaatsen. Op deze manier kan er geen verschillen ontstaan tussen de locaties van blokken in de array en de locaties van de gameObjects.

GameGrid is een class die extend van de gewone Grid class. Deze class heeft voornamelijk meer functies dan de gewone class. De belangrijkste zijn het verplaatsen van blokken en het verwijderen van volle rijen. In het spel zitten 2 Grid objecten. Het grote speelveld is een GameGrid. Het veld waar het volgende blok getoond wordt is een gewoon Grid.

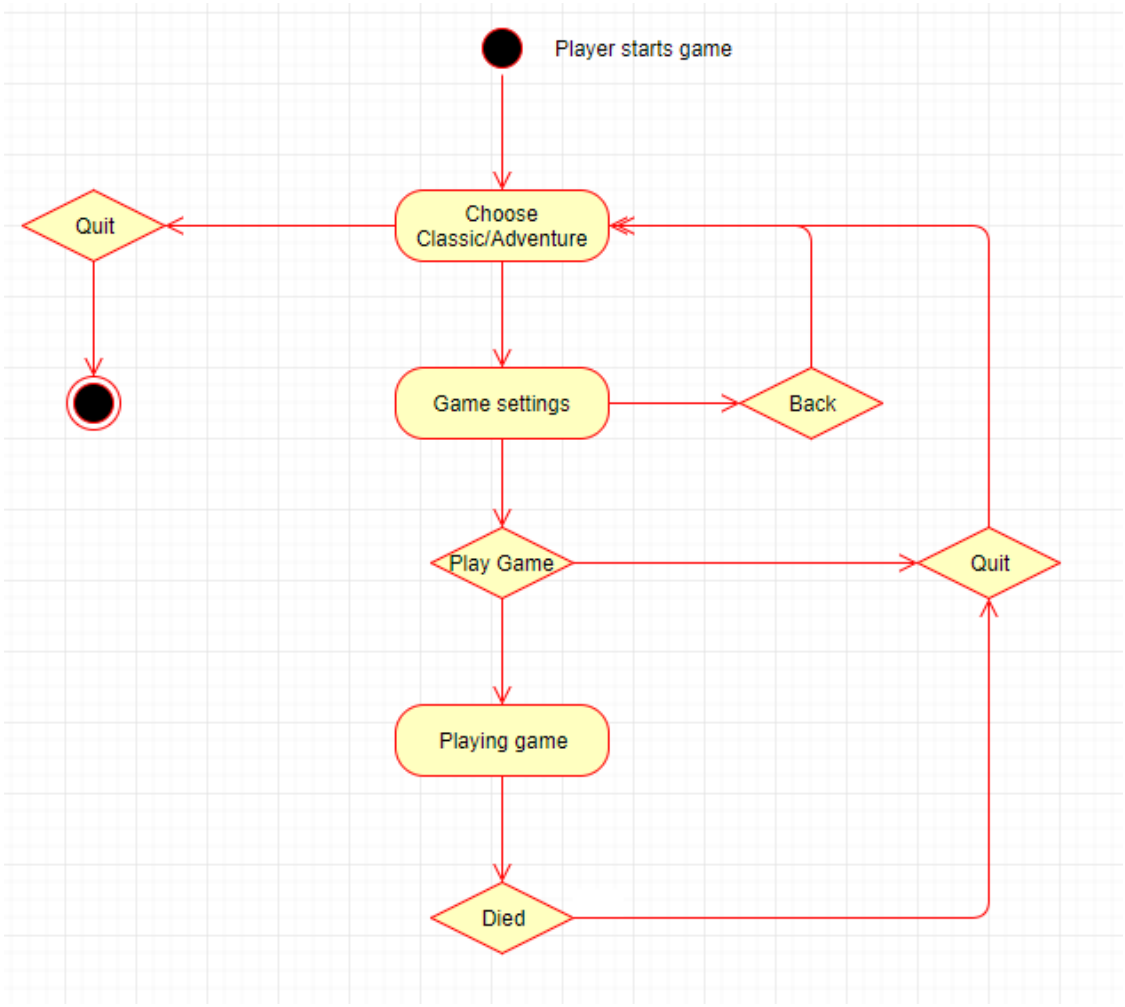
GamePulse en Synchronised behaviour

Tetris heeft een eigenlijk een hele lage framerate. Om dit na te maken in unity heb ik de gamepulse class gemaakt. Deze heeft één event waarin hij een pulse stuurt. Deze pulse is een int die oploopt.

Synchronised behaviours zijn monobehaviours die luisteren naar de gamepulse. Ze hebben een OnBeat functie die aangesproken wordt. Door een simpele modulus operatie kan de frequentie van de pulse aangepast worden. Hierdoor lopen alle classes altijd synchroon.

Game menu

Het menu is erg simpel en geïnspireerd door de NES versie van tetris.



Design Patterns

Character : MVC

De character classes maken gebruik van een model - view - controller structuur. Zowel non-playable als playable characters gebruiken deze classes. De view class regelt de animator, de model class regelt beweging en de controller class regelt de input/AI. Eerst had ik hiervoor 2 classes gemaakt (een voor alle enemy behaviour en een voor alle player behaviour). Het grote voordeel van dit pattern was voor mij dat ik de Model- en View class allebei kon hergebruiken. Daarnaast ben ik erg blij met hoe overzichtelijk deze classes zijn gebleven door ze op deze manier op te splitsen.

Later in het project ben ik meer functionaliteit toe gaan voegen aan de controller classes die eigenlijk meer in de andere classes thuis horen. Zo regelt de player controller class ook de beweging van de speler wanneer deze op het grid zit. Dit breekt dus wel met het pattern. Ik heb dit gedaan omdat de code erg verbonden is met die van de PlayerController en ook erg uniek is voor de player.

TetrisController : State machine

Deze class beheert alle tetris mechanics. De state machine bestaat uit de volgende states:

- start
- newBlock
- blockIsMoving
- blockHasLanded
- gameOver

De class is ook een Synchronised Behaviour. Deze combinatie vond ik heel interessant omdat hierdoor elke nieuwe state wordt uitgevoerd tijdens de volgende GamePulse. Het wordt daardoor heel makkelijk om bijvoorbeeld een kleine delay te maken tussen het moment dat een blok geland is en het volgende blok spawned. Hierdoor voelt het spel ook gelijk meer aan als de retro tetris.

GameManager : Observer

De game manager verzamelt gegevens over de voortgang van het spel van allerlei classes. Deze gegevens hebben invloed op de belangrijkste game dynamics: score, speed en level. De game manager stuurt deze informatie ook als enige door naar de UI.

Verandering van het ontwerp tijdens het project

Het ontwerp is best veel veranderd door het project heen. Dit komt onder andere doordat ik aan het begin van het project nog niet wist hoe goed de tetris en platformer mechanics zouden werken in één game. Mijn originele idee was om meer platformer mechanics toe te voegen. Het idee was dat je echt door een wereld zou kunnen gaan lopen, interactie kon hebben met blokken en een doel had waar je naar toe moest zien te komen.

Van tetris naar platformer

Ik ben begonnen met het maken van tetris. Daarna heb ik een heel simpel character gemaakt en die in het veld gezet. Ik kwam er al gelijk achter dat er weinig platformer mechanics nodig zijn om het spel interessant te maken. Het level veranderd namelijk al continu door de vallende blokken.

Het spel was toen nog wel heel makkelijk omdat je het character makkelijk op een veilige plek kon laten staan. Daarom heb ik er voor gekozen om voornamelijk mechanics toe te voegen die je uitdagen om het character te verplaatsen. Dit zijn dus de enemies die heen en weer lopen en de coins die op willekeurige posities in het grid spawnen.

Game management

De grootste verandering aan het ontwerp die ik pas laat heb doorgevoerd is de manier waarop de gamemanager werkt. Toen ik begon aan het project was de gamemanager ook de class die alle tetris mechanics regelde. Ik heb dit later gesplitst omdat deze class erg groot werd. De TetrisController class is daardoor geworden wat eerst de GameManager was. De GameManager class regelt nu alleen de game flow en de UI.

Overige design patterns

Tijdens het project heb ik ook nog een aantal andere design patterns uitgeprobeerd. De meeste hiervan heb ik niet gebruikt omdat ze veel sterker zijn dan nodig waren voor dit project. Zo had ik een plan om een object pool te gebruiken voor blokken en enemies. Ik heb dit achterwegen gelaten omdat er maar weinig blokken gespawnd worden en het zijn heel simpele objecten.

Ik heb ook overwogen om een decorator te gebruiken voor de AI van de enemies. Ik heb dit niet meer gedaan omdat ik nog geen goede manier had gevonden om een decorator te maken met monobehaviours. Daarnaast is de code voor de AI erg kort. In plaats van een decorator maakt de class nu gebruik van de bools IsKillableByBlock, IsKillableByPlayer en CanJump.

Wat ik anders wil doen in volgende projecten

Toen ik begon met het project was het redelijk logisch dat de game manager ook de tetris mechanics controleerde. Dit komt omdat toen het spel alleen maar bestond uit Tetris. Toen ik de twist er in begon te maken raakte deze gameManager class gelijk erg vol. Doordat ik pas laat in het project de Tetris mechanics afgesplitst heb van de game manager is het verschil tussen deze twee een beetje vaag geworden. De GameManager class is wel een redelijk overzichtelijke class nu maar de TetrisController doet ook veel dingen waar van je verwacht dat de GameManager dit zou doen. Daarnaast zorgt de TetrisController ook nog voor het spawnen van een aantal platformer objecten. De volgende keer ga ik daarom op het begin van het project beter nadenken over welke functies dit soort manager classes moeten hebben.

Tijdens dit project heb ik er voor gekozen om eerst de tetris mechanics te maken en later steeds meer platformer mechanics toe te gaan voegen. Doordat ik de weken voor de laatste week van het project wat minder tijd heb gehad om hier aan te werken heb ik in de laatste paar dagen nog veel nieuwe mechanics toe moeten voegen. Dit was niet waar ik op gehoopt had. Ik heb hierdoor erg weinig tijd gehad om alles goed te polishen. Voor volgende projecten ga ik daarom proberen om een deadline aan te houden waarna ik niet meer nieuwe functionaliteit ga toevoegen.

Als ik meer tijd had gehad om dingen te polishen had ik ook graag meer tijd willen steken in de animaties van alle objecten. Zo had ik ook graag animaties willen maken voor blokken die kappot gaan en had ik particle effecten willen toevoegen wanneer een character plat gedrukt wordt. Ik denk dat deze dingen erg belangrijk zijn omdat er nu heel veel dingen gebeuren in het spel die de speler snel mist omdat er weinig visuele feedback is.