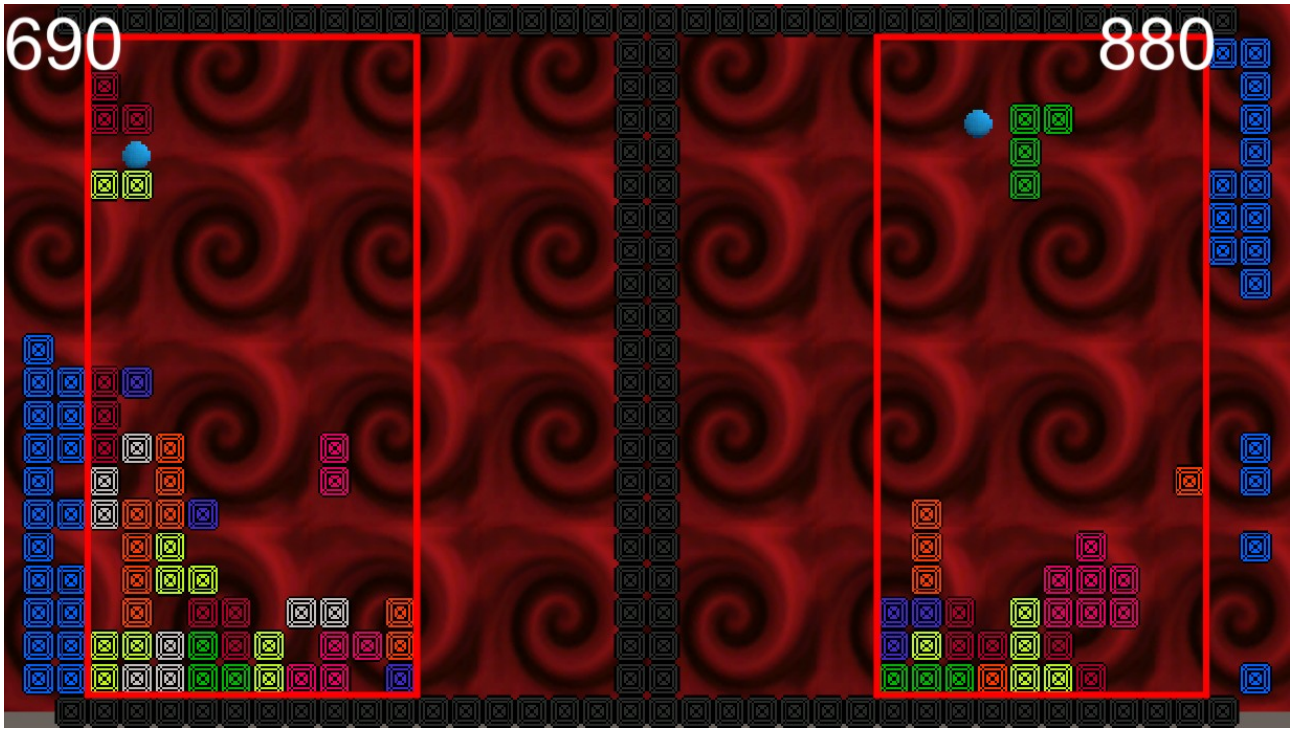
Kernmodule 1 Game Development

Tjaard van Verseveld – 3028886



Juni 2020

Concept

Voor deze opdracht heb ik besloten om een spel te maken wat een combinatie is van Tetris, Pong en Breakout. Twee spelers spelen tegen elkaar en moeten zowel voorkomen dat ze hun speelveld tot aan de bovenrand vullen, als dat de bal aan de zijkant het scherm verlaat. De minos, ofwel de bouwstenen van de tetrominos (vallende blokken) kunnen in tegenstelling tot Tetris niet alleen kapot gemaakt worden door een rij volledig te vullen, maar ook net als de blokken bij breakout door een bal hier tegenaan te laten stuiteren. De minos hoeven hiervoor niet eens op de grond te liggen en kunnen ook vernietigd worden als ze nog vallende zijn. Dit kenmerk zorgt ervoor dat er veel variatie in blokvormen ontstaat wat zowel voordelig als nadelig kan uitpakken voor de spelers. Het spel eindigt als beide speler hun speelveld volledig gevuld hebben. De speler met de hoogste score is dan de winnaar.

Codestructuur

Design Patterns

Ik heb de volgende design patterns binnen dit project gebruikt.

- **Singleton**
  Classes die slechts eenmalig voorkomen, waardoor het makkelijker is om ze te benaderen.
- **Object pool**
  Zorgt dat het spel efficiënter werkt door objecten te hergebruiken in plaats van ze telkens opnieuw aan te maken.
- **Interfaces**
  Specifieke functies die door enkele scripts benut worden.

Uitwerking

Hieronder een overzicht van de classes die gebruik maken van patterns. Het voorvoegsel V3 dat bij de meeste classes staat, slaat op het feit dat deze tweemaal herschreven zijn, aangezien zij eerst geen gebruik maakten van design patterns.

- **V3AudioManager**
  Singletonclass die verantwoordelijk is voor al het geluid in het spel. Voor het afspelen van geluiden word gebruik gemaakt van een pool waarin verschillende audiokanalen beschikbaar zijn.
- **V3Ball**
  Normale class die verantwoordelijk is voor het bewegen van een bal. Erft de startpositie-eigenschap van een interface waarvan gebruik gemaakt word als een bal het scherm verlaat.
- **V3BlockManager**
  Singletonclass die verantwoordelijk is voor het vormen van tetrominos door minos uit een poolobject te halen. Stopt minos tevens terug in de pool als ze ‘vernietigd’ worden.
- **V3Controller**
  Singletonclass die algemene spellogica en functionaliteit verzorgt.
- **V3Mino**

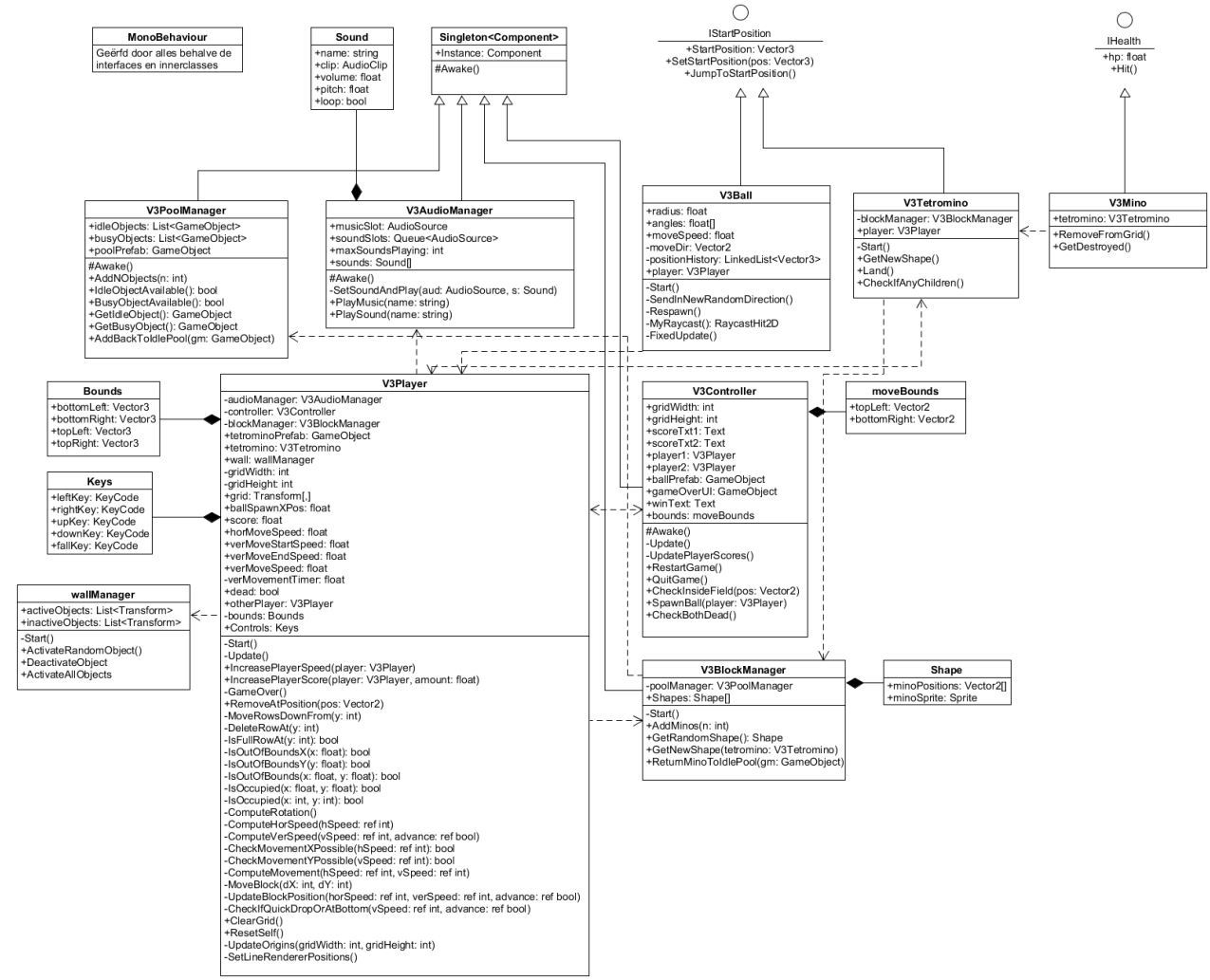
Normale class die ervoor zorgt dat het spelerveld waarin een mino zich bevindt bijgewerkt wordt als hij vernietigd wordt. De class maakt alleen gebruik van een health-interface om bij te houden hoe vaak de mino geraakt kan worden voordat hij kapot gaat.

- **V3PoolManager**
  Singletonclass die voor alle pools zou moeten zorgen, maar dit nu alleen voor de minos van de BlockManager doet.
- **V3Tetromino**
  Zorgt voor het vallende blok bij een speler. Haalt zijn elementen via de BlockManager uit de minopool. Net als de bal erft deze class de startpositie-interface, en maakt hier gebruik van als hij op de grond is gevallen en weer bovenaan het speelveld van een speler als een nieuw blok verschijnt.
- **WallManager**
  Aanvankelijk was het idee dat deze class voor alle muren van de spelers zou zorgen, waardoor dit een singleton zou worden. Echter is dit nu niet het geval en bezit hij deze eigenschap dan ook niet. Wel werkt deze class met een pool om de stenen voor de muren in te houden.

Uitdaging

Over het algemeen kostte het mij moeite om patterns in de code toe te passen. Dit komt voornamelijk, omdat ik hier simpelweg tot op heden weinig gebruik van heb gemaakt en de lessen van deze module indertijd heb gemist, omdat ik destijds nog de richting game design volgde. Daarnaast vereist het ook dat je als programmeur al vanaf het begin een goed idee hebt hoe je iets op zal gaan zetten, en niet zoals ik vaak neig dit pas ontdekken na veelvuldig methodes te bedenken en iteratief stukken code te schrijven en herschrijven.

UML Diagram



Activity diagram