Start Document Design Patterns

Stendinator the game – VERSIE 0.1

Henk bembom, sydney minnaar en Jordy Neef

2022

Contents

[1. Het Idee 2](#_Toc97297590)

[2. Werkmethode 3](#_Toc97297591)

[3. Design Patterns 4](#_Toc97297592)

[3.1 Strategy Pattern 4](#_Toc97297593)

[3.2 Factory method pattern 4](#_Toc97297594)

[3.3 Builder Pattern 4](#_Toc97297595)

[3.4 Observer Pattern 4](#_Toc97297596)

[3.5 Decorator 4](#_Toc97297597)

[4 Klassendiagrammen 5](#_Toc97297598)

# 1. Het Idee

Voor de eindopdracht van Design Patterns moet er een softwareproduct worden opgeleverd. Aan de eindopdracht zijn eisen gesteld. Het softwareproduct moet minimaal vier *design patterns* bevatten. Design patterns zijn software architectonisch van aard om een specifiek probleem in de code op te lossen.

De projectgroep dat gaat werken aan de eindopdracht bestaat uit drie personen. Het idee is om een game te ontwikkelen waarbij vijf design patterns worden toegepast. De game heet *Stendinator.* In de game kan de speler een robot samenstellen om vervolgens tegen vijanden te vechten. De vijand en de speler moeten om de beurt een actie uitvoeren. De speler kan niet winnen in het spel, omdat het spel oneindig doorgaat.

De speler begint met het samenstellen van de robot. De robot bestaat uit verschillende ledematen. De ledematen kunnen verschillende unieke eigenschappen hebben. Zo kan een ledemaat offensief of defensief van aard zijn.

Als de speler tevreden is met zijn robot, dan begint het spel. De speler wordt naar verschillende planeten gestuurd om tegen vijanden te vechten. Planeten kunnen een positief- of negatief effect hebben op de prestatie van de robot om de gevechten spannend te maken. De speler vecht tegen twee vijandige facties: De Aliens en de Humans. De vijanden worden random gegenereerd, waardoor geen enkel gevecht hetzelfde zal zijn. Hoe meer vooruitgang de speler boekt, hoe sterker de vijanden worden.

Wanneer de speler een vijand verslaat, dan krijg hij resources om ledematen te upgraden. Afhankelijk van het type ledemaat dat wordt geüpgradede kunnen bepaalde statistieken van de robot aangepast worden. De speler kan pas ledematen upgraden nadat alle vijanden op een planeet zijn verslagen.

## Stappen die worden doorlopen bij het spel

De volgende stappen is de procedure die de speler in de console bij langs gaat.

1. Bouw de Cyborg
   1. Componenten uitkiezen om te worden toegevoegd
2. Versla een vijand
3. Upgrade de Cyborg
   1. Selecteer nieuwe componenten met betere statistieken:  
      Er worden een willekeurig aantal componenten gedropt en de componenten zijn ook willekeurig, de statistieken worden vermenigvuldigd met het planeetniveau)
4. Versla een planeet (Verhoog het planeetniveau)
5. Begin opnieuw bij stap 2.

# 2. Werkmethode

De projectgroep bestaat uit drie personen: Henk Bembom, Sydney Minnaar en Jordy Neef. Om het idee te realiseren zal om te beginnen een klassendiagram worden gemaakt. Tijdens het ontwerpen zal direct rekening worden gehouden dat bepaalde design patterns geïmplementeerd worden.

Design patterns zijn best practices die softwareontwikkelaars kunnen toepassen om softwareproblemen op te lossen, zoals het overzichtelijker maken van code. Het zijn daarom zeer handige methoden om toe te passen bij het ontwerpen en realiseren van Stendinator.

Verder zal de projectgroep de SOLID principes toepassen om ten einde te komen van Clean Code. Dit houdt in dat code flexibel en duidelijk is, zodat andere softwareontwikkelaars eenvoudig verder kunnen voortbouwen op de bestaande code.

Het eindresultaat gaat getest worden volgens de best practices op het testgebied. Om effectief en efficiënt te testen zal gebruikt worden gemaakt van twee tools: NSubstitute en NUnit. Door deze tools te combineren ontstaat er een krachtig middel om de code te testen.

Om code tussen de projectleden uit te wisselen zal gebruikt worden gemaakt van Github. Hierbij zal gebruik worden gemaakt van Branches, zodat projectleden binnen veilig in hun eigen omgeving kunnen werken zonder code stuk te maken in de *master* branch.

De projectleden zullen werken in de nieuwste versie van Visual Studio (2022). Hiervoor zal het team gebruik maken van het Universal Windows Platform (UWP). UWP is een geschikt platform om een game mee te ontwikkelen. Daarnaast wil het team graag beter bekend raken met UWP.

Tot slot zal het team de door NHL Stenden gedocumenteerde C# programmeerconventie hanteren om de kwaliteit van de code hoog te houden.

# 3. Design Patterns

In dit hoofdstuk worden de toegepaste ontwerppatronen onderbouwd.

## 3.1 Strategy Pattern

Voor het bepalen van het type creature wordt de strategy toegepast, aangezien de verschillende soorten creatures vergelijkbaar met elkaar zijn maar verschillend gedrag hebben. Zo heeft elke creature zijn eigen attacks, en de planeten van elk soort creature net een andere (de-)buff voor de speler heeft.

## 3.2 Factory method pattern

De factory method maakt het mogelijk om het planeet object (superclass) creatures aan te laten maken zonder de kennis te hebben van de alien of cyborg (derived classes). Dit zorgt ervoor dat het planeet object geen logica van het creëren van de specifieke objecten, waardoor de klasse niet hoeft worden aangepast als er een nieuwe creature wordt toegevoegd.

## 3.3 Builder Pattern

De cyborg bestaat uit veel complexe componenten zoals ledematen. Dit willen we niet in de constructor hebben aangezien deze dan te groot wordt en daarom wordt de builder toegepast. De builder maakt het dan ook mogelijk om de complexe objecten te kunnen creëren. Op deze manier kunnen we de benodigde parameters om tot eenvoudige stappen die achter elkaar uitgevoerd kunnen worden.

## 3.4 Observer Pattern

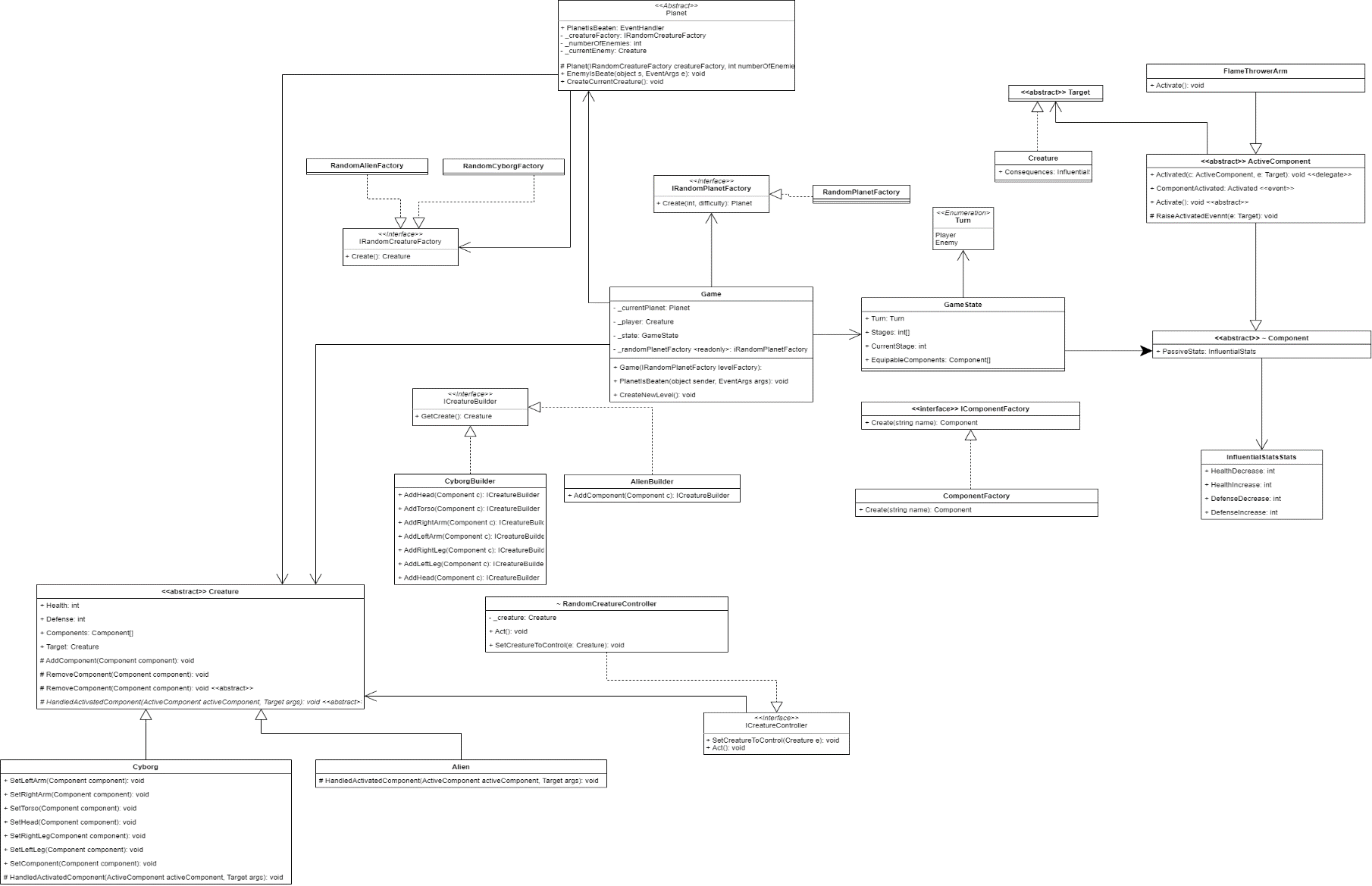
Er zijn situaties waarin meerdere creatures bestaan uit hetzelfde component. Hierdoor wordt dan ook de observer pattern toegepast om de creatures (die bestaan uit de desbetreffende components) op de hoogte te stellen dat ze zijn gebruikt.

## 3.5 Decorator

Elk component die in de UI wordt getekend heeft zijn eigen vorm. Nu kun je een algemene klasse hebben die kijkt naar het soort component en op basis daarvan zijn vorm zal gaan tekenen. Een mooiere oplossing hiervoor is de decorator. De decorator zorgt ervoor dat de componenten zelf beschikken over de manier waarop zij worden getekend.

# 4 Klassendiagrammen

In dit hoofdstuk worden de klassendiagrammen van het ontwerp gevisualiseerd. Daarnaast wordt er ook beschreven welke klassen gerelateerd zijn aan welke ontwerppatronen.



Figuur Klassendiagram Kern-project

Diagram

Description automatically generated

Figuur 2 Klassendiagram grafische schil (in UWP)

Figuur 1 toont het klassendiagram van de te realiseren Stendinator game. Bij het ontwerpen van het klassendiagram is rekening gehouden met het implementeren van de gekozen Design Patterns die in hoofdstuk 3 uiteen zijn gezet.

Door de naamgeving is duidelijk waar in het klassendiagram van figuur 1 de builder en de factory worden toegepast. Minder voorhand liggend zijn de strategy, observer en de decorator pattern.

De strategy komt van pas wanneer de speler ledematen (componenten) wil gaan uitwisselen tussen de levels door. De observer wordt gezet tussen de component en de creature(s). De creatures observen de component, omdat de creatures moeten opletten hoeveel schade een ‘vijandig’ component gaat aanbrengen. De decorator pattern wordt toegepast in het GUI project (figuur 2). Dit patroon is duidelijk gemaakt door de naamgeving.