MSO Assignment 3

Tobias de Bruijn (4714652)
Herkansing
2024-01-26

Inhoudsopgaven

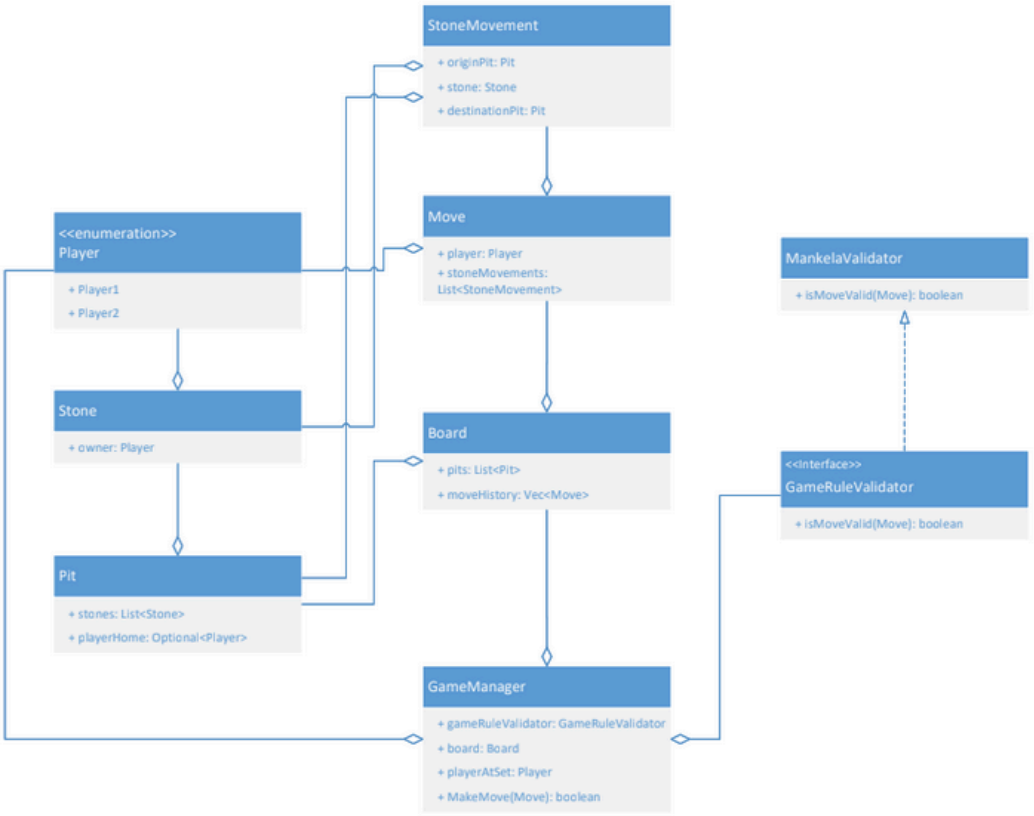
Inhoudsopgaven **1**
Het spel **2**
Software ontwerp **3**
Originele UML 3
Vernieuwd UML 4
Design Patterns 5
Alternatieve user interfaces 5
Code Quality **6**
Testing **7**
Taakverdeling **8**

Het spel

Mankela en Wari zijn beide volledig geïmplementeerd in het programma. In het README.md bestand staat omschreven hoe het spel gestart dient te worden.

Software ontwerp

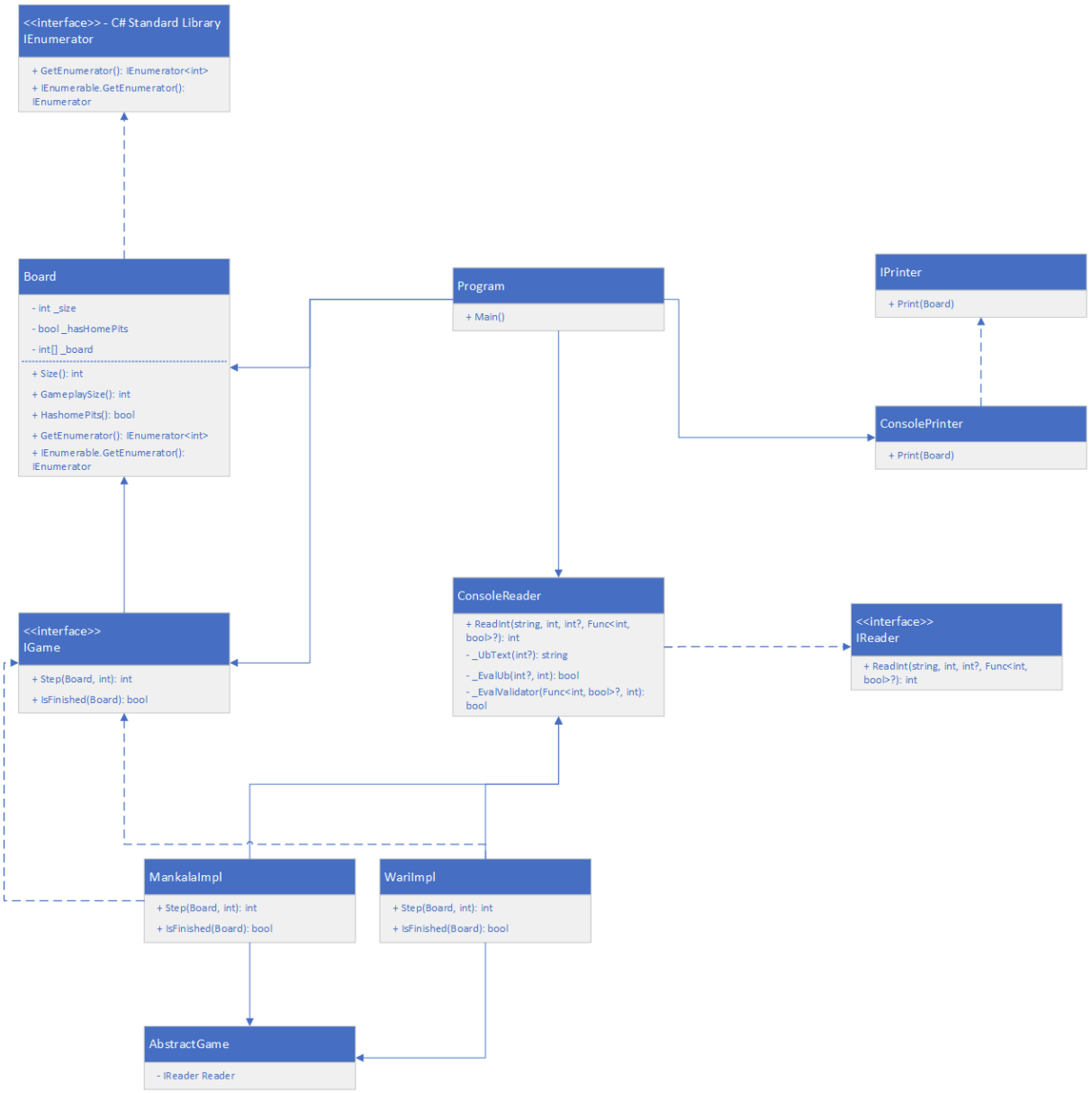
Originele UML



Vernieuwd UML

Bij het maken van dit project voor de herkansing heb ik gemerkt dat het originele ontwerp van assignment 2 niet fijn werkt.
Zo is het erg lastig om de tegenovergestelde kuiltjes te vinden en zit er veel onnodige code in het programma en ontwerp.

Hierom heb ik gekozen om vanaf nul te beginnen. In dit geval is het bestaande ontwerp aanpassen aan de nieuwe eisen, vele malen meer werk dan het opnieuw maken.



Design Patterns

Er is maar één designpatroon gebruikt, het strategy pattern. Dit patroon is toegepast op de game implementatie, de user input en de interface. De specifieke implementatie van een interface wordt tijdens runtime pas besloten. In het geval van IPrinter en IReader is dat nu constant, maar als er in de toekomst een andere UI opgezet wordt, kan het zonder ingrijpende veranderingen volledig dynamisch veranderd worden.

Voor de game implementaties is zowel gebruikgemaakt van een interface (IGame) als een abstracte klasse (AbstractGame). De interface definieert de functies die een game implementatie moet hebben. AbstractGame bestaat om de IReader mee te kunnen geven aan elke game implementatie, zonder dat dat veld op elke implementatie los gedefinieerd moet worden.

Alternatieve user interfaces

Het bord wordt aan de gebruiker gepresenteerd via de IPrinter interface. Bij het opstarten van het programma wordt er een ConsolePrinter instantie aangemaakt, een implementatie van IPrinter. Om bijvoorbeeld WinForms te gebruiken dient er een adapter gemaakt te worden die IPrinter implementeert, en dat goed omzet naar WinForms functies.

Verder wordt de IReader interface gebruikt om input op te vangen van de gebruiker. Bij het opstarten van het programma wordt er een ConsoleReader instantie aangemaakt, een implementatie van IReader. Om bijvoorbeeld WinForms te gebruiken dient er een adapter gemaakt te worden die IReader implementeert, en dat goed omzet naar WinForms functies.

Code Quality

Standaarden

Ik heb gebruik gemaakt van het standaard C# formaat zoals voorgeschreven door Microsoft: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions.

Op punt ben ik hier vanaf geweken: de positie van de haakjes.
C# schrijft voor dat de haakjes op nieuwe regels staan. Ik vind dat niet erg prettig werken, en heb ze liever op dezelfde regel.

Naast deze afwijken wordt de standaard in zijn geheel gevolgd. Dit wordt grotendeels automatisch gewaarborgd door Rider, zoals omschreven in het volgende kopje.

De naamgeving van functies en variabelen volgt ook de voorgeschreven standaarden. Ook dit is automatisch gecontroleerd door Rider.

Automatische code kwaliteit

Mijn IDE omgeving is JetBrains Rider, ingebouwd zit ReSharper. Tijdens het werk aan de software zijn de waarschuwingen van Rider gelijk verwerkt, vaak nog automatisch ook. Ook is de Roslyn-analyser losgelaten op het project.

Een voorbeeld van een verbetering voorgesteld door ReSharper is het omdraaien van if statements. Een voorbeeld van zo'n draaien komt uit de Mankela implementatie:

```
if (board[i] != 0) {
   player1Empty = false;
   break;
}
```

Kan omgedraaid worden tot:

```
if (board[i] == 0) continue;
player1Empty = false;
break;
```

Ook het compacter maken van if-statements naar ternary statements is een goed voorbeeld:

```
if (ub == null) {
   return "\u221e";
} else {
   return ub.ToString();
}
```

Naar

```
return ub == null ? "\u221e" : ub.ToString();
```

Hierna komt Rider met nog een suggestie:

```
return ub == null ? "\u221e" : ub.ToString()!;
```

Voorkomende uit de waarschuwing: Possible null reference return. Ik weet dat ToString niet null geeft, dus is een ! van toepassing.

Een nog extremer voorbeeld, inclusief refactoring tussenstappen:

```
private static bool _EvalUb(int? ub, int val) {
    if (ub == null) {
        return true;
    } else {
        // Using negation to make the bounds check more clears
        return !(ub < val);
    }
}
```

Naar

```
private static bool _EvalUb(int? ub, int val) {
    return ub == null ? true : !(ub < val);
}
```

Naar

```
private static bool _EvalUb(int? ub, int val) {
    return ub == null || !(ub < val);
}
```

Verder kan dit nog handmatig gekrompen worden naar. (Dit kan overigens ook automatisch met de hotkey: ‘Convert to Expression body).

```
private static bool _EvalUb(int? ub, int val) => ub == null || !(ub < val);
```

Documentatie

Alle functies zijn waar nodig gedocumenteerd met C#’s ingebouwde documentatiefunctie. Dit is gedaan ofwel met /// ofwel met /** */. Afhankelijk van of de parameters extra uitleg nodig hadden.

Functies die van zichzelf spreken zijn niet gedocumenteerd.

Comments zijn toegepast waar het niet direct duidelijk is uit de code wat het precies doet of waarom afwegingen zijn gemaakt.

Testing

Niet geïmplementeerd

Taakverdeling

Niet van toepassing.