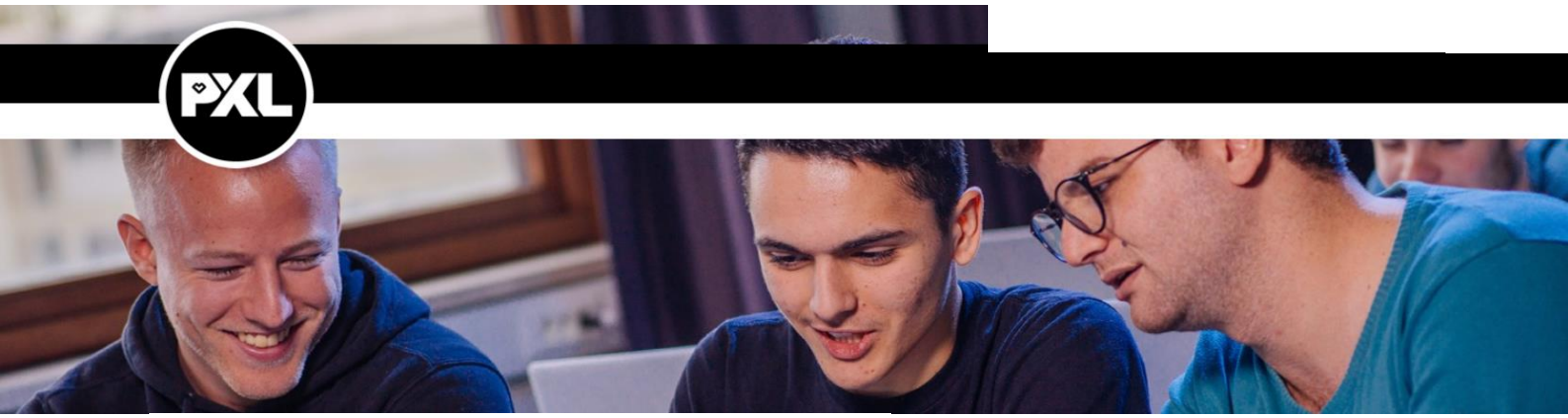


GRADUAAT IN HET
PROGRAMMEREN

Opdracht bowling WPL1 - draaiboek

Project

1PRW

Klas

Werkplekleren 1 – 2020/21

Opleidingsonderdeel

DE HOGESCHOOL
MET HET NETWERK

INHOUD

1. Overzicht opdracht

1.1 Doel van het project

Het doel van dit project is om de kennis opgedaan in de lessen in de praktijk om te zetten.

1.2 Planning en fasen

De volledige toepassing zal worden geïmplementeerd in verschillende fasen. In elke fase worden telkens nieuwe functionaliteiten toegevoegd aan het project. Op deze manier worden ook de complexiteit en nodige skills stapsgewijs opgebouwd.
Dit project zal zich onderscheiden van het BSS project door meer zelfstandigheid van de student te eisen.

Lesweek 10	Fase 1	Basisversie
Lesweek 11	Fase 2	Vereenvoudiging
Lesweek 12	Fase 3	Uitbreiding
Lesweek 13	Oplevering	Finalisering

1.3 Indienen opdracht

Hoewel het project opgedeeld is in verschillende fasen, moet enkel het eindresultaat opgeleverd worden.
Zie het hoofdstuk over het indienen van de opdracht later in dit document.

1.4 Permanente evaluatie

De evaluatie is gebaseerd op onderstaande criteria:

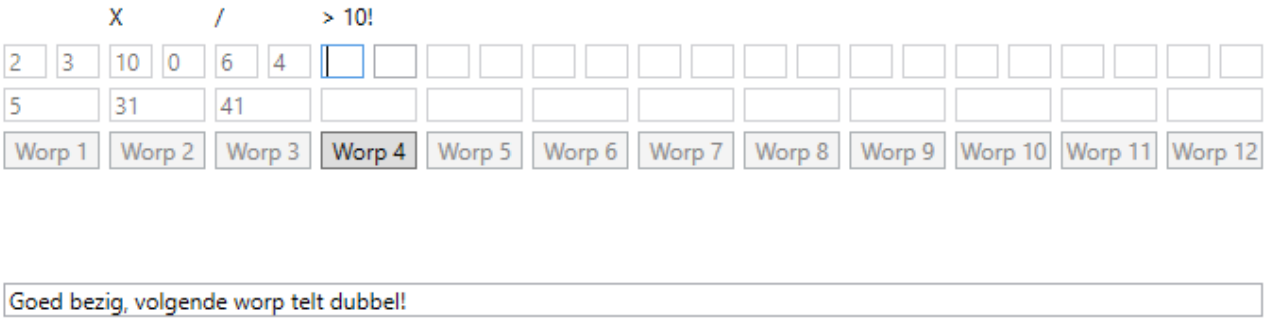
Onderdeel	Omschrijving
Is de gevraagde functionaliteit aanwezig?	Zijn de alle knoppen aanwezig? Wordt de validatie correct toegepast? Is er een gepaste lay-out en stijl gebruikt? Zijn de besturingselementen juist afgestemd?
Correcte werking	Worden de tussentijdse resultaten correct geteld? Treden er onverwachte fouten op tijdens het spelen van het spel? Worden strikes en spares correct bijgehouden?
Kwaliteit code	Worden de juiste programmeer technieken gebruikt? (vb. lists, classes, …) Is de code gestructureerd en duidelijk leesbaar? Is de code voldoende gedocumenteerd?

2. Fase 1

1.1 Basisversie

Maak een eigen class library **Puntentelling** dat de punten van een bowlingspel bijhoudt. Je maakt een method waar je minstens 2 parameters aan mee geeft, nl. het aantal pins die omver zijn geworpen per worp.
In het geval van een strike (zie 3.5 Bowling regels) mag de 2de parameter leeg zijn. In alle andere gevallen niet. De classlib moet alle worpen zelf achterliggend bijhouden in geheugen, het is dus niet nodig om weg te schrijven naar een database of een tekstbestand.
Het spel is beëindigd na de 10de, 11de of 12de worp (afhankelijk van strike/spare/open frame), hou hier rekening mee!
Voorzie dat je het tussentijdse resultaat kan opvragen zodat je applicatie dit ook mooi kan tonen (zoals een echt bowlingspel).
Daarnaast maak je een WPF project **BowlingGUI** dat deze class library gebruikt. Meer details vind je hieronder.
Beide projecten steek je in dezelfde solution **Bowling**.

1.2 GUI (scherm)



1.3 Code

- Per worp vul je in het kleine vakje de score in. Op het einde van de 2de worp klik je op de betreffende knop en je tussenscore wordt berekend.
- Zorg ervoor dat je enkel getallen kan ingeven die tussen 0 en 10 vallen.
- Zorg ervoor dat het totaal van de 2 worpen niet meer dan 10 kan zijn. Geef een melding in het overeenkomstige info veld.
- Maak een indicatie igv een strike of spare.
- Hou rekening dat indien er een strike of spare gegooid wordt, de punten extra worden geteld EN dat er eventueel een uitbreiding komt in het aantal spellen indien dit het laatste spel zou zijn waarin dit gegooid wordt.

1.4 Programmeertips

- Gebruik een class om per frame de gegevens bij te houden
- Gebruik een dictionary om de frames bij te houden
- Hou het totaal aantal worpen bij om het speleinde aan te duiden (standaard is dit 10)
- Maak 2 boolean variabelen om te weten of er een strike of spare gegooid is
- Gebruik je dictionary object om punten bij de vorige worp te kunnen tellen. Haal het vorige frame op d.m.v. de key te gebruiken in je dictionary object.

- Staar je niet blind op de puntentelling!! We spreken af dat we het spel vereenvoudigen door telkens maar 1 frame terug te gaan. Dus als een speler 2 strikes of spares achtereen gooit, dan moet je maar 1 frame teruggaan om het resultaat te beïnvloeden.
- De puntentelling kan bijvoorbeeld als volgt:

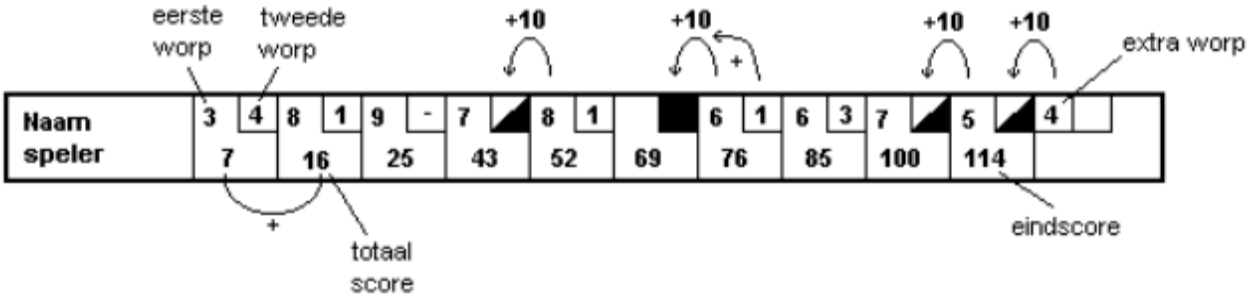
```
public class Frame
{
    3 references
    public byte worpEen { get; set; }
    2 references
    public byte worpTwee { get; set; }
    5 references
    public int resultaat { get; set; }

    1 reference
    public Frame(byte worp1, byte worp2, int resultaat)
    {
        worpEen = worp1;
        worpTwee = worp2;
        this.resultaat = worp1 + worp2 + resultaat;
    }
}
```

- Je frames kan je bijhouden in een dictionary zoals in volgend voorbeeld:

```
private Dictionary<byte, Frame> dictFrames = new Dictionary<byte, Frame>();
Frame newFrame = new Frame(worp1, worp2, vorigFrame == null ? 0 : vorigFrame.resultaat);
dictFrames.Add(frameNr, newFrame);
```

1.5 Bowling regels



Als je een **strike** gooit, dan heb je 10 punten verdiend, maar daarbij worden nog de eerste twee volgende beurten als bonus bij opgeteld. En omdat na een strike de pins om zijn is de beurt om en tellen deze punten pas mee bij je volgende beurt. Gooi je daarna een 'open frame' door bijvoorbeeld eerst 6 en dan 3 pins om te gooien, heb je in frame 1 een extra score van 9 punten. Dus 10 + 9 = 19. In het geval dat je na een eerste strike weer een strike gooit, dan zal de tweede bonusgooi pas in de derde frame plaatsvinden. Gooi je daar ook een strike dan scoor je in frame 1 de maximale score van 30 punten.
Als je een **spare** gooit, dan heb je ook 10 punten gehaald, en krijg je 1 x bonuspunten extra uit de volgende beurt. Na een spare kun je maximaal 20 punten krijgen als je de beurt erna alles in één keer omgooit (en dan heb je weer een strike).
Gooi je een **open frame** (dus niet alles om na twee beurten, dan heb je alleen de punten gescoord van het aantal pins dat je hebt omgegooid. Dit is dan dus altijd minder dan 10. En je krijgt geen bonuspunten bij de volgende worp.
De maximale score die je kan halen per 10 frames is 300 punten. Hiervoor moet je wel 12 strikes achter elkaar gooien (10 frames en 2 extra worpen na de 1e frame).

2 Fase 2

2.1 Vereenvoudiging van code

In plaats van alle code te herhalen per knop, maak je een functie die je telkens oproept.

2.2 Programmeertips

- Noem je knoppen "_1" tot en met "_12", overeenkomstig het framenummer dat ze vertegenwoordigen.
- Koppel 1 en hetzelfde click event achter elke knop. In de eerste regel haal je met onderstaande code het framenummer op:

```
byte frameNr = byte.Parse(((Button)sender).Name.Substring(1));
```

- Je kan een object op de GUI dynamisch ophalen. Voorbeeld:

```
(TextBox)this.FindName($"tbWorp{frameNr}")
```

  Als het frameNr 1 is, geeft je dit de textbox met de naam "tbWorp1" terug.

3 Fase 3

3.1 Uitbreiding

Zorg ervoor dat je alles kan wegschrijven naar een CSV bestand. Maak hiervoor een class aan die hiervoor zorgt. Zorg ervoor dat je enkel een export kan doen wanneer spel gedaan is! Je mag dit doen door een knop te gebruiken.

4 Fase 4

4.1 Testing en finalizing

Test je hele applicatie nog eens volledig door. Klik op alle knoppen, textboxen, e.d. en probeer je applicatie te doen crashen. Denk out-of-the-box, probeer te denken zoals een eindgebruiker denkt, stap af van de logica die je zelf als ontwikkelaar erin hebt gestoken. Alleen op deze manier kan je een “bullet-proof” applicatie opleveren. Dit noemen ze ook wel “monkey-testing”: zoveel mogelijk op alle knoppen tegelijk klikken. Als je applicatie dit overleeft, dan zit je goed.

Controleer daarnaast ook of je toepassing functioneel werkt zoals het hoort:

- Klopt de puntentelling?
- Zit de validatie goed?
- …

4.2 Oplevering

Zorg dat je alles mooi in een zip file steekt en dat je het geheel op tijd upload. Kijk op Blackboard voor meer info!