Bowling Scoring System Design og Implementering Specifikation

1.Problemstilling:

Angående rekrutteringsproces hos SKAT/Udvikling og forenklingsstyrelsen ønsker man at have en løsning til en programmeringsopgave, der handler om implementering af Bowling-points algoritme og evt. dokumentation for løsningen. Link til opgaven kan findes på https://github.com/skat/bowling-opgave

2. Arkitektur og teknologivalg:

Følende teknologier er på grund af egen erfaring og hardware/software til rådighed tilvalget:

- Programmeringssprog: C# og .NET Framework
- Løsningsdesign/arkitektur: Windows Desktop Applikation
- UI: Windows WPF Framework

3. Data models:

For at løse opgaven har man brug for en række data modeller til understøtning for Bowling-point algoritme, UI, forretningslogik samt to REST API: GET og POST samt 2 url/end points som midlertidigt tilbydes af SKAT. Disse metoder bruges for at hente de generede input data til beregningen og post til validering af de beregnede output data hos SKAT. Til integration med disse REST API følgende data model er konstrueret:

3.1. GET API data model:

GET API returnerer data i JSON-list og kræver derfor et tilpas data model i C#:

```
  public class Scores
  {
    [JsonProperty(PropertyName = "points")]
    public List<List<int>> Points { get; set; } //som array of array
[[3,7],[5,0],[10,0]]
    [JsonProperty(PropertyName = "token")]
    public string Token { get; set; } //as id of a scoring serie
    ...
  }
```

Scores-datastruktur bruges til at parse med de JSON outputdata returneret fra GET og præsenterer senere som "input data" til bowling-point algoritme.

3.2. POST API data model:

POST API forventer 2 JSON format datastrukturer, et for parameter og et for returnerede svar:

3.2.1. POST parameter:

```
public class Summaries
{
  [JsonProperty(PropertyName = "token")]
  public string Token { get; set; }
  [JsonProperty(PropertyName = "points")]
  public int[] Points { get; set; }
  ...
}
```

`Summaries` kaldes som parameter, hvor token er hentet fra GET API og `Summaries`.Points er beregnet vha. Bowling-point algoritme.

3.2.2. POST returnerede data:

```
public class Response
{
  [JsonProperty(PropertyName = "success")]
  public bool Success { get; set; }
  ...
}
```

Hvilken er pakket inde i content-property af en standard HttpResponseMessage-strukturen.

3.3. Forretningslogik data model:

FrameScore, som identificerer hver enkel omgang samt med sine scoringspointer og implementerer IFrameScore-interface:

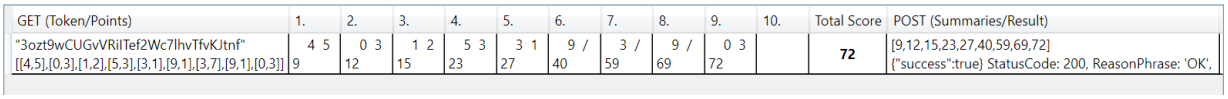
```
public interface IFrameScore
{
  List<int> Points { get; }
  int First { get; }
  int Total { get; }
  int FrameNo { get; }
}

public class FrameScore : IFrameScore
{
  ...
}
```

3.4. UI View Modeller:

- FrameView: Præsenterer pointer + sum for hver omgang
- ScoreView: Visual Data model for hver enkelt spil

```
public class FrameView
{
  private IFrameScore frameScore;
  public int Score { get; private set; } //Frame's Summary point
   ...
 }

public class ScoreView
{
  public ScoreView(...) //Constructor
  {
    ...
  }

  public string Bowler { get; private set; }
  public FrameView FrameView_01 { get; private set; }
  public FrameView FrameView_02 { get; private set; }
  public FrameView FrameView_03 { get; private set; }
  public FrameView FrameView_04 { get; private set; }
  public FrameView FrameView_05 { get; private set; }
  public FrameView FrameView_06 { get; private set; }
  public FrameView FrameView_07 { get; private set; }
  public FrameView FrameView_08 { get; private set; }
  public FrameView FrameView_09 { get; private set; }
  public FrameView FrameView_10 { get; private set; }
  public int Total { get; private set; }
  public string Comment { get; private set; }
}
```

- Bowler i den implementering viser output data for token, pointer fra GET API.
- FrameView_XX viser Score pointer og summer for hver enkelt omgang
- Total viser total scoring pointer for hver spil
- Comment viser data for token, summer, validerede resultat til/fra POST API.



GET (Token/Points)	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	Total Score	POST (Summaries/Result)
"3ozt9wCUGvVRilTef2Wc7lhvTfvKJtnf" [[4,5],[0,3],[1,2],[5,3],[3,1],[9,1],[3,7],[9,1],[0,3]]	4 5 9	0 3 12	1 2 15	5 3 23	3 1 27	9 / 40	3 / 59	9 / 69	0 3 72		72	[9,12,15,23,27,40,59,69,72] {"success":true} StatusCode: 200, ReasonPhrase: 'OK',

4. REST API Implementering:

Både GET API og POST API implementeres vha. komponent HttpClient fra .NET System.Net.Http framework. Response data er pakket inde i HttpResponseMessage.content property efter returneret og kan parse til vores data model vha. hhv. `content.ReadAsAsync<Scores>()` og `content.ReadAsAsync<Response>()`

4.1. GET API:

GET API implementeres vha. HttpClient metoden `GetAsync(url)` med url/end point som parameter:

```
async public static Task<Scores> Get(string url)
{
  Scores result = null;
  try
  {
    using (HttpClient client = new HttpClient())
    {
      using (var response = await client.GetAsync(url))
      {
        using (var content = response.Content)
        {
          result = await content.ReadAsAsync<Scores>();
        }
      }
    }
  }
  catch (Exception ex)
  {
    //Exception handling
    ...
  }

  return result;
}
```

4.2. POST API:

POST API implementeres vha. HttpClient metoden `PostAsJsonAsync(url, summaries)` med url/end point og `Summaries` som parameter:

```
async public static Task<PostResponse> Post(string url, Summaries summaries)
 {
   PostResponse result = null;
   try
   {
     using (HttpClient client = new HttpClient())
     {
       using (var responseMessage = await client.PostAsJsonAsync(url, summaries))
       {
         result = new PostResponse { httpResponse = responseMessage };
         using (var content = responseMessage.Content)
         {
           var response = await content.ReadAsAsync<Response>();
           result.contentResponse = response;
         }
       }
     }
   }
   catch (Exception ex)
   {
     //Exception handling
     ...
   }

   return result;
 }
```

5. Implementering af Bowling-point algoritme:

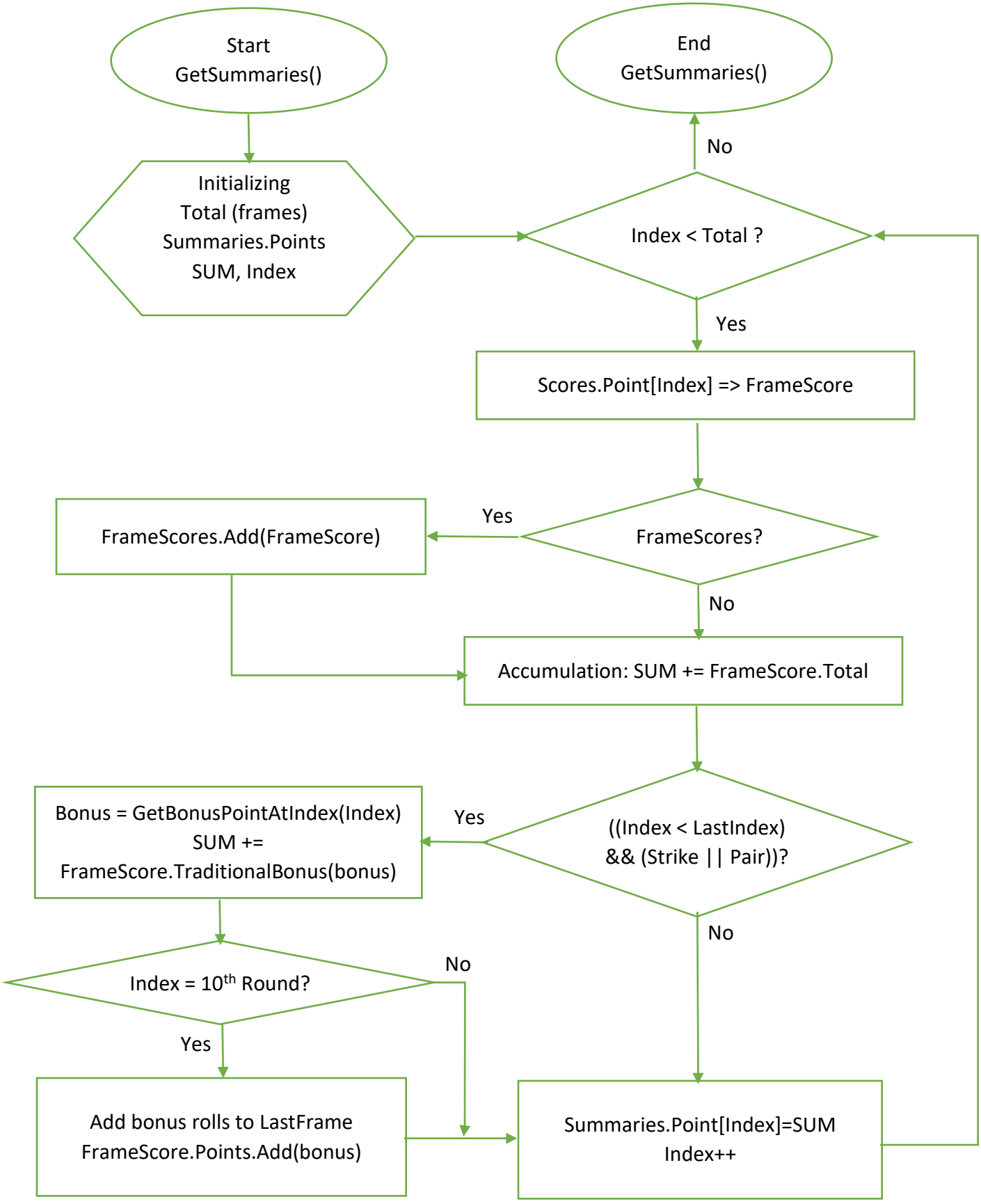
Implementering af algoritmen skal følge den traditionelle 10-kegel regel:

- https://en.wikipedia.org/wiki/Ten-pin_bowling#Traditional_scoring
- https://da.wikipedia.org/wiki/Bowling
- Simulator: http://www.bowlinggenius.com/

Samt med at den også kan beregne hvis antal omgang er mindre end 10.

- Input data: Scores.Points i form af et array[array[2] af heltal] eller ækvivalent med en List<List<int>> eks. [[5,4],[7,3],[6,0]]
- Output data: Summaries.Points i form af et array af heltal (af max 10 omgang), eks. [9,25,31]
- Collection af FrameScores (List<FrameScore>) som valgfri parameter, der præsenterer for evt. visualisering af bowling pointer i et skema.

5-1. Flowchart:

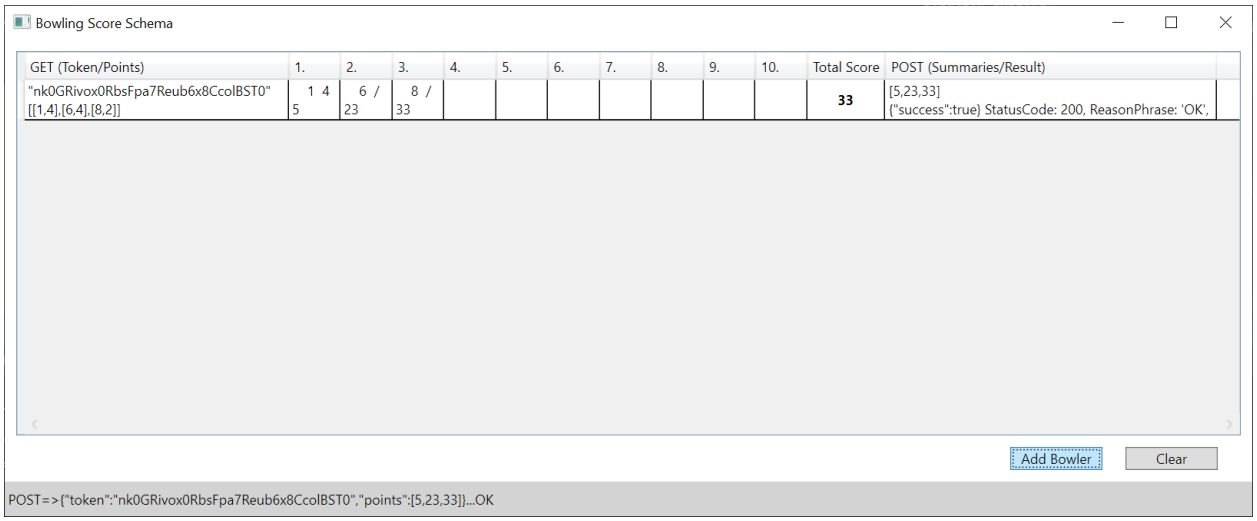


5.2. Implementering:

```
public class TraditionalScoring
{
  const int MaxFrame = 10;
  public static Summaries GetSummaries(
    Scores scores, List<IFrameScore> frameScores = null)
  {
    int total = Math.Min(scores.Points.Count, MaxFrame);
    var result = new Summaries {
      Token = scores.Token,
      Points = new int[total] };
    try
    {
      var sum = 0;
      var lastIndex = total - 1;
      for (int i=0; i <= lastIndex ; i++)
      {
        var score = new FrameScore(scores.Points.ElementAt(i), i);
        if (score != null)
        {
          if (frameScores != null)
            frameScores.Add(score);
          sum += score.Total;

          if (i < scores.Points.Count - 1 &&
              score.Total == 10)
          {
            var bonusPoints = Scores.GetBonusPointsAtIndex(i, scores);
            sum += FrameScore.TraditionalBonus(score, bonusPoints);
            if (i == 9)
            {
              score.Points.AddRange(bonusPoints);
            }
          }
        }

        result.Points[i] = sum;
      }

      return result;
    }
    catch (Exception e)
    {
      Console.WriteLine(e.Message);
    }

    return null;
  }
}
```

6. UI Implementering:

Windows Desktop Application med anvendelse af WPF, Datagrid-komponenter er valgt som den enkelte løsning til at demonstrere med hensyn til test og visualisering af opgaven:



UI er enkel designet med en datagrid, der præsenterer Bowling Score Schema/Sheet og 2 knapper:

- ”**Add Bowler**”: henter, beregner og validerer en ny spil.
- ”**Clear**”: Renser indhold af datagrid om det ønskes

7. Unit Test Implementering:

Der er også implementering af Unit Test til automatisering af test efter bygning af kode. De test ud over almindelig scoring algoritme dækker også over for:

- Strike scoring. Mock data: [[10,0],[3,6]] => [19,28]
- Maksimal scoring med 10 omgang, der skulle returnere 300 som den maksimale score
- Pair scoring. Mock data [[7,3],[7,2],[4,2]] => [17,26,32]

Alle test skulle have passet som vist efter kørsel i Visual Studio 2017: