# SWC Mini-projekt: YATZY

Dette lille projekt går ud gå at forstå og videreudvikle en C# applikation, som implementerer de første skridt på vejen til et fuldt YATZY-spil.

## Introduktion

Yatzy er et terningespil, som de fleste nok kender grundprincipperne for[1]. Elementerne og reglerne er så tilpas simple, at det ikke er så svært at komme i gang med at implementere i form af et C# projekt. I projektet **Yatzy** (findes i "Unsolved" repository: https://github.com/perl-easj/Unsolved-csharp-projects) er der taget de første skridt til at implementere spillet.

Hent projektet **Yatzy**, og åbn det i Visual Studio.

## Projektet Yatzy

Projektet **Yatzy** rummer fra start et antal klasser. Detaljer omkring implementationen af den enkelte klasse findes som kommentarer i koden.

- **Die**: Repræsenterer en enkelt terning.
- **DiceCup**: Repræsenterer et raflebæger, som kan rumme et antal terninger.
- **IDiceEvaluator**: Definerer et (meget lille) interface for en "evaluator", som er en klasse der kan evaluere et sæt terninger i.f.t. en af de kombinationer, der giver point i Yatzy (f.eks. "Tre Ens").
- **ChanceEvaluator** og **OnePairEvaluator** (I mappen **Evaluators** i projektet): To implementationer af **IDiceEvaluator**, som evaluerer et sæt terninger i.f.t. hhv. "Chance"- og "Et Par"-kombinationerne.
- **GameManager**: En klasse som håndterer selve afviklingen af et spil Yatzy.

Alle klasserne er som sådan fungerende, men selve spillet er meget skrabet fra start. Faktisk sker der ikke andet end dette, når metoden **Run** fra **GameManager**-klassen afvikles (se **Main** i **Program.cs**):

*Udfør dette 10 gange:*

- *Ryst bægeret, og udskriv hvad terningerne viser*
- *Evaluér terningekastet i forhold til "Chance" og "Et Par", og udskriv resultatet*

Altså rummer **Run** p.t. blot en lille test. Opgaven i dette projekt er at udvide implementationen af spillet.

I resten af teksten kommer der et antal opgaver relateret til **Yatzy**-projektet. De bliver gradvist sværere. Se hvor langt du kan komme, og arbejd gerne sammen med en makker om at løse opgaverne.

[1] Ellers se http://spillereglerne.dk/maxi-yatzy/

# Opgaver – del A

Åbn klassen **Die**

- Hvor mange instance fields rummer klassen? Hvad er typen af hver instance field?
- Hvor mange metoder rummer klassen? Hvor mange properties? Hvad er forskellen generelt på metoder og properties?
- Hvad sker der i constructoren? Hvad er formålet generelt med en constructor?
- Hvad betyder ordet **static**? Hvorfor er **_generator** mon defineret til at være **static**?

Åbn klassen **DiceCup**

- Hvad er typen af **_dice**? Hvad bestemmer, hvor mange **Die**-objekter der bliver indsat i **_dice**?
- Hvad er formålet med metoden **ToString**? Hvad betyder ordet **override**?
- Prøv at se om du kan redegøre for, hvad der sker i **DiceValues**.
- Prøv at se om du kan redegøre for, hvad der sker i **DiceCountByValue**.

Åbn klassen **IDiceEvaluator**

- Hvordan adskiller en interface-definition sig generelt fra en klasse-definition?
- Kan man lave et objekt af typen **IDiceEvaluator**?
- Kan lave definere en variabel af typen **IDiceEvaluator**?
- Hvorfor kan interfaces være nyttige?

Åbn klasserne **ChanceEvaluator** og **OnePairEvaluator**

- Hvilke relationer har disse klasser til andre klasser/interfaces i **Yatzy**-projektet? Hvordan vil du beskrive koblingen mellem **…Evaluator**-klasserne og de to klasser **Die** og **DiceCup**?
- Prøv at se om du kan redegøre for, hvad der sker i **Evaluate** i **ChanceEvaluator**.
- Prøv at se om du kan redegøre for, hvad der sker i **Evaluate** i **OnePairEvaluator**.

Åbn klassen **GameManager**

- Hvilken type har **_diceEvaluators**? Hvad formål tjener dette instance field?
- Prøv at se om du kan redegøre for, hvad der sker i constructoren for **GameManager**.
- Prøv at se om du kan redegøre for, hvad der sker i metoden **Run**.

Du har nu (forhåbentligt) et overblik over, hvordan Yatzy-projektet er struktureret, og hvilke formål de enkelte klasser i projektet har.

## Opgaver – del B

Implementér nogle flere ”evaluators”, d.v.s. klasser som:

- Arver fra **IDiceEvaluator**-interfacet.
- Implementerer logikken for at evaluere det givne sæt terninger, i.f.t. en bestemt kombination (f.eks. ”Fuldt Hus”).

Du vælger selv, hvor mange og hvilke ”evaluators” du vil implementere. Husk nu at få testet, om din implementation virker korrekt!

## Opgaver – del C

En vigtig del af at spille Yatzy er naturligvis at holde styr på de points, hver spiller har scoret.

Tænk over, hvilke klasser det kunne give mening at definere, i.f.t. at holde styr på points, og hvilke relationer de vil have. Et par forslag kunne være:

- **ScoreBoardEntry**: Svarende til en enkelt pointgivende kombination, f.eks. ”To Par”
- **ScoreBoard**: Alle de kombinationer, som indgår i et spil, knyttet til en enkelt spiller.
- **GameManager**: Kan udvides med en collection af **ScoreBoard** objekter (en for hver spiller)

Begynd på implementationen af nogle af disse klasser.

## Opgaver – del D

En spiller skal også kunne interagere med spillet. I Yatzy vil en spiller – når det er spillerens tur – kunne udføre følgende trin:

1. Slå et slag med raflebægeret
2. Udvælge nogle terninger som skal gemmes, d.v.s. ikke indgå i det næste slag
3. Slå et slag med raflebægeret, med de tilbageværende terninger
4. Udvælge nogle terninger som skal gemmes
5. Slå et slag med raflebægeret, med de tilbageværende terninger
6. Vælge hvilken kombination det endelige udfald af terningerne skal noteres under (evt. notere scoren 0 i en kombination)

Tænk over, hvordan denne algoritme kan implementeres i **GameManager**-klassen. Hvordan skal spilleren mere specifikt kunne interagere med applikationen? Prøv gerne at skitsere/implementere en ”rigtig” GUI til spillet, d.v.s. med controls defineret i XAML, definition af Commands, osv. (NB: Dette er naturligvis en ganske stor opgave, så prøv blot at komme så langt som tiden tillader).