Setting up a track

Taakgroep: Coding a simulation

Context

In de vorige levels werd je nog stap voor stap meegenomen, vergelijkbaar zoals je gewend was tijdens het 1e schooljaar. Vanaf deze level wordt er steeds meer zelfstandigheid en oplossingsgerichtheid van je verwacht passend bij het niveau van dit schooljaar. Gebruik je verstand en creativiteit om tot oplossingen te komen. Wees vrij om extra methodes en/of variabelen aan te maken als jij dit nodig vindt, zolang je maar blijft voldoen aan de SOLID principes.

Kom je er even niet uit ga even 10 minuten iets anders doen, vaak helpt dat om dan weer een frisse blik te krijgen. Vergeet niet om de tooling, zoals de `debugger`, te gebruiken om inzicht te krijgen over de exacte werking van je software. Blijft het lastig? Leg je probleem eens voor aan een medestudent. Kom je er met geen mogelijkheid meer uit neem dan contact op met je docent.

Tijdens de vorige levels heb je al het nodige grondwerk gedaan. Een model geïmplementeerd, een start gemaakt met de klasse `Race` en een begin gemaakt met je test project. In deze level ga je de klasse `Race` verder uitbreiden zodat je een hele race kan simuleren. Tenslotte ga je de race visualiseren op de console.

Taak: Creating the track

Voordat de klasse `Race` verder kan uitbouwen zal je eerst aan de slag moeten met de klasse `Track`. Zonder een track kan je geen race simuleren.

Aanpak

Tijdens het implementeren van het UML diagram heb je al de klasse `Track` aangemaakt. Deze klasse heeft volgens het UML diagram 2 properties. De constructor van deze klasse heeft 2 parameters `name` en `sections`.

- Mocht je dit nog niet gedaan hebben geef, in de constructor, de property `Name` de waarde van de parameter `name`.

  Nu denk je wellicht, ik ga de waarde van de parameter `sections` gebruik om de property `Sections` te zetten. Als je goed kijkt is de type van de parameter een `array` van `SectionTypes`. De property is van het type `LinkedList<Sections>`. Je kan dus niet zomaar de parameter gebruiken voor de property. Je zult hier dus iets voor moeten doen.

Volgens de SOLID principes heeft elke methode maar 1 verantwoordelijkheid. De verantwoordelijkheid van de constructor is niet om een array te converteren naar een LinkedList. Dus ga je een nieuwe methode maken in de klasse `Track` om een array van `SectionTypes` om te zetten naar `LinkedList<Sections>`.

- Geef de klasse `Track` een methode met als parameter `SectionTypes[]` en als return type `LinkedList<Section>`
- Maak in de methode een nieuwe variable van het type `LinkedList<Section>`
- Maak voor elke waarde in de parameter een `Section` aan en voeg deze toe aan de variabele
- Retourneer de variabele
- Roep nu de methode in de constructor waarbij je de return waarde van de methode gebruikt als waarde voor de property `Sections`

Je klasse `Track` is nu gereed om gebruikt te worden.

Ondersteunende informatie

Wil je meer weten over de verschillende niveaus? Bekijk dan eens de Domein beschrijving op de website van HBO-I.

Taak: Setting up the race

Om je werk eenvoudiger te maken werk je vaak parallel aan verschillende onderdelen.
De `debugger` geeft veel inzcht over de werking van je code. Toch is het ook prettig om daadwerkelijk je code "in actie" te zien. Daarom ga je alvast een baan visualizeren zodat je bij de je volgende taken een beter beeld krijg over de werking van je code.

In deze taak ga je werken aan je `console project` en je baan met tekst en (ASCII) tekens visualiseren. Uiteraard kan je helemaal los gaan en al je creativiteit inzetten. Dit is meteen ook een valkuil. Als software ontwikkelaar moet je altijd de beschikbare tijd en het einddoel bewaken. De tip is bijna altijd: maak eerst wat er minimaal verwacht wordt. Wanneer er tijd over is kan je altijd nog je werk beter, efficienter en mooier maken. Stel daarom altijd eerst vast wat er precies verwacht wordt en hoeveel tijd je hiervoor krijgt.

Aanpak

- Maak in de console project een `static` klasse aan. Deze klasse is verantwoordelijk voor de visualisatie.

  De keuze voor `static` is omdat er maar 1 instantie van hoeft te bestaan. Je mist nu wel een constructor dus zal je een static methode moeten aanmaken voor de initializatie van je klasse.

- Maak in de klasse een static methode `Initialize`.

  Bedenk hoe jij je baan visueel wilt weergeven. Let wel dat een baan opgedeelt is in verschillende secties.
  Een voorbeeld van sectie met een finish:

```
----
  #
  #
----
```

  Een bocht kan je bijvoorbeeld zo visualiseren:

```
--\
   \
\  |
|  |
```

  Je bent uiteraard helemaal vrij om je eigen visualisatie te bedenken en gebruiken. Tip is wel om elke sectie even hoog als breed te maken, bijvoorbeeld 4x4 tekens. Dit maakt het een stuk eenvoudiger om alle secties aan elkaar te plaatsen. Zorg er wel voor dat er 2 deelnemers naast elkaar getoond moeten kunnen worden. Aan jou is de keuze om nu alle secties uit te tekenen of straks methodes te schrijven om secties horizontaal en verticaal te kunnen spiegelen.

Wanneer je klaar bent ga je alle visualisaties een plek geven in de klasse. Hiervoor ga je gebruik maken van `arrays` van `string`. Je kan voorstellen dat alle teksten snel je klasse onoverzichtelijk en moeilijk leesbaar maakt. Om hier wat aan te doen ga je gebruik maken van `region`. Hiermee kan je delen code tekstueel samenvoegen zonder dat het invloed heeft op de werking van de code.

- Voeg aan de klasse de onderstaande code toe:

```
#region graphics


#endregion
```

Tussen `#region graphics` en `endregion` ga je nu alle variabelen met de visualisaties aanmaken.

- Voeg in de `region` de volgende code toe:

```
private static string[] _finishHorizontal = { "----", "  # ", "  # ", "----" };
```

Hiermee heb je private array van string aangemaakt met de naam `_finishHorizontal`. Zoals je wellicht opvalt bestaat de array uit 4 elementen. Als je deze elementen onder elkaar opschrijft krijg je de visualisatie van een sectie met een finish.

Uiteraard heb jij gekozen voor een andere visualisatie dus wees vrij om de elementen van de array aan te passen zodat het aansluit bij jouw eigen visualisatie.

- Voor alle ontworpen secties maak een private string array aan binnen de `region`. Geef de variabele een duidelijk naam.

Wanneer je alle private variabelen hebt aangemaakt kan je deze allemaal handig "verbergen" door op het minnetje te drukken aan de kantlijn bij `#region graphics`. Door weer op het plusje te drukken opent de `region`, met alle code, weer.

- Geef de klasse nu een methode `DrawTrack` met een `Track` als parameter.

Je kan eindelijk beginnen met een complexe uitdaging, van object van het type `Track` naar uitvoer op de console. Het is handig en effectief om, naast alle informatie dankzij de `debugger`, te zien wat er staat te zien is op het scherm.

- Vervang de code in static methode `main`, waarmee je de naam van de `Track` op het scherm zet, met de aanroep naar de `DrawTrack` methode.

Wanneer je nu het console project start zie je niets meer staan op de console.

De volgende stap vraagt het nodige denkwerk, creativiteit en doorzettingsvermogen van je. Je bent vrij om extra methoden en variabelen aan te maken om de taak af te ronden, houd je wel aan de SOLID principes.

- Implementeer nu de methode `DrawTrack` zodat de meegegeven parameter van het type `Track` weergegeven wordt op de console.

  Wanneer je klaar bent met je taak wil je uiteraard meteen aan de slag om extra unit tests te schrijven om zo de code coverage weer op het afgesproken niveau te krijgen. Dit is alleen een typisch voorbeeld waarbij handmatig wordt getest. Waar jij dagen bezig bent om tests te ontwikkelen om het beeld op de console te controleren kan een mens dit vaak in 1 oogopslag. Tijdens een project beschrijf jij, in een testplan dan ook de tests die handmatig uitgevoerd moeten worden.

Ondersteunende informatie

Wil je meer weten over de `region` keyword, kijk dan op deze pagina.

Taak: Placing the participants

Alle gegevens die nodig zijn heb je tot je beschikking in de klasse `Race`. De property `Track` bevat de racebaan en de property `Participants` alle deelnemers die meedoen met de race. Voordat de race kan beginnen moeten alle deelnemers een plek krijgen op de baan. Je gaat in deze taak de klasse `Race` uitbreiden zodat alle deelnemers een logische startpositie krijgen toegewezen.

Het is raadzaam om eerst na te denken voordat je gaat programmeren. De taak klinkt eenvoudig; geef alle deelnemers een logische startpositie. Toch zal je met meerdere factoren rekening moeten houden. De belangrijkste regel is dat er maar 2 deelnemers op een sectie geplaatst mogen worden. Daarnaast moeten alle deelnemers een startpositie krijgen passend bij de type race, dus niet willekeurig geplaatst op de baan.

Wanneer je een strategie hebt bedacht en deze aan het programmeren bent gebruik de debugger regelmatig. De `trial and error` methode gaat je veel tijd, energie en veel frustratie kosten! Nogmaals, gebruik de `debugger`!

Aanpak

- In de klasse `Race` maak een methode aan die als doel heeft alle deelnemers op de baan een start positie te geven.
- Implementeer de methode. Gebruik de properties `Track` en `Participants` als input. Sla de meegegeven start posities aan de deelnemers op in de variable `_positions`, gebruik hierbij de methode `GetSectionData` als hulpmiddel. Tip: gebruik de property `SectionType` van `Section` om een logische start posities te selecteren.
- Breid de constructor van de klasse uit en roep de methode aan.

Taak: Setting up the race

Wanneer je nu het console project start zie je nog geen deelnemers op de baan ondanks dat je ze nu wel geplaatst hebt. In deze taak ga je de deelnemers ook weergeven. Wanneer je programmeert is het verstandig om slim gebruik te maken van alles wat je tot je beschikking hebt. C# en .NET bieden al veel hulpmiddelen waarvan je goed gebruik kan maken in deze taak.

Aanpak

- In de klasse, die de baan visualiseert op de console, pas je de strings welke je gebruikt om je secties weer te geven. Bij elke sectie voeg je een teken toe die je nog niet hebt gebruikt; bijvoorbeeld "1" en "2". Deze tekens zijn placeholders voor de linker en rechter deelnemers op de sectie. Zie het voorbeeld hieronder.

```
----
  2#
 1 #
----
```

- Maak een methode aan met als parameters van de types `string`, `IParticipant` en `Iparticipant`. De return value is van het type `string`.

  Je zult nu een keuze moeten maken hoe je elke bestuurder gaat weergeven op de baan. Denk bijvoorbeeld aan de eerste letter van de naam.

- Implementeer de methode waarbij de tekens vervangen worden bij de `string` parameter. Retourneer de aangepast string. Tip: gebruik de string `Replace` methode hiervoor.

  Code schrijf je vaak in fases. Bij elke oplevering "groeit" je software. Dit betekent ook dat je vaak je eigen code, of die van een ander, moet aanpassen en/of uitbreiden. Wanneer je aan de SOLID principes hebt gehouden, bijbehorende tests en duidelijke commentaar hebt gebruikt kost dit veel minder tijd. Bij de volgende taak zal je ook je eigen code moet aanpassen en uitbreiden om de nieuwe functionaliteit te implementeren.

- Pas nu de bestaande code in de klasse aan waarbij je de nieuwe methode gebruikt zodat de deelnemers getoond worden op de baan.

  Je hebt in deze level veel gedaan. Laat aan de visualisatie, waarbij de deelnemers getoond worden, zien aan je docent om deze taak af te laten tekenen.

Ondersteunende informatie

Meer informatie over de string `Replace` methode kan je hier vinden.