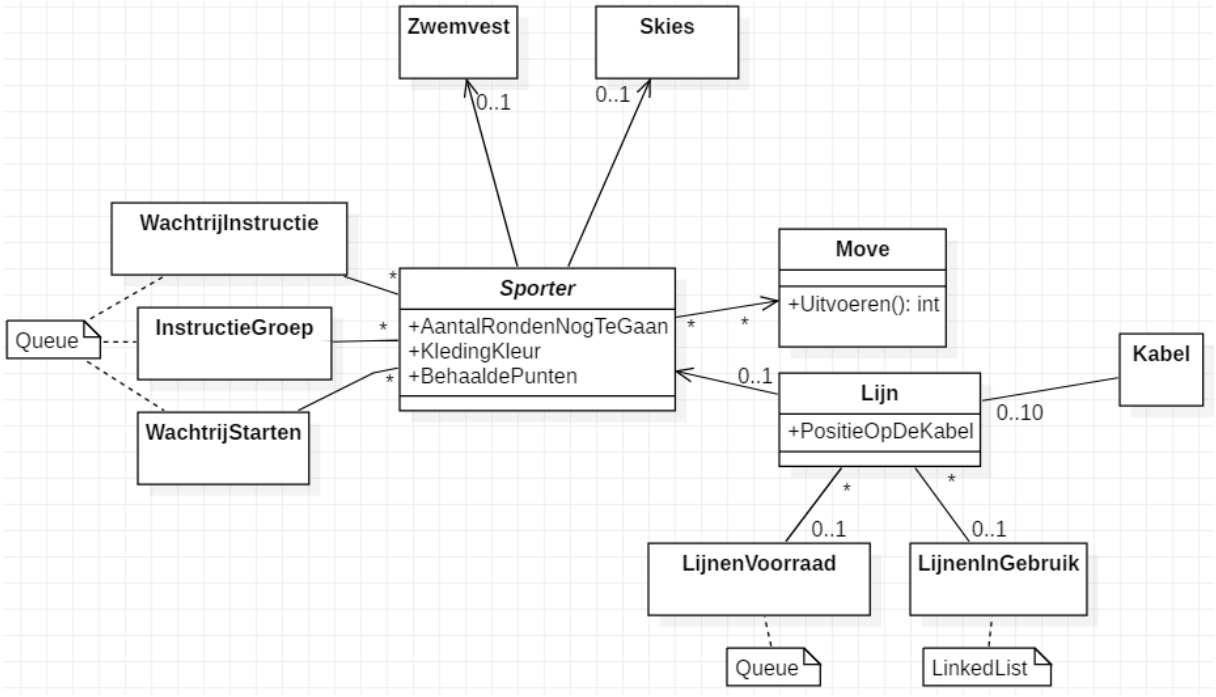
Opdracht Waterskibaan

In deze opdracht gaan we een kabelwaterskibaan simuleren. De opdracht maak je met een groepje van drie studenten. De opdracht bestaat uit 15 opgaven. Sommige opgaven mag je met zijn drieën uitvoeren, andere opgaven doe je individueel, waarbij ieder een deel van de opgave uitvoert. Jullie gaan gebruik maken van Git voor het ontwikkelen. Maak een GIT repository en deel die met je docent.

Kabelskiën doe je niet achter een boot, maar aan een kabelbaan. Met maximaal tien sporters tegelijk ski je aan een soort van sleeplift. De omloopkabel hangt aan masten boven het water en wordt aangedreven door een elektromotor. Een waterskiër kan middels een lijn aan de draaiende kabel gekoppeld worden. De lijn kan ook weer afgekoppeld worden. Je kunt de lijn dus vergelijken met de lijn die normaal gesproken aan een speedboot gekoppeld wordt.
Voordat de bezoekers mogen sporten moeten ze instructies hebben gehad. Als bezoeker heb je te maken met meerdere wachtrijen, eerst voor het ontvangen van een instructie en daarna moet je op je beurt wachten bij de kabel.

Op de kabel kunnen zo als vermeld maximaal 10 lijnen aangekoppeld worden. Om de sporters niet te dicht op elkaar te laten skiën, is de baan verdeeld in 10 gelijke stukken. Als sporter haak je aan op positie 0, tenminste als deze positie vrij is. De kabel draait en de lijn verplaatst zich steeds een positie verder, totdat positie 9 bereikt is. Als de lijn dan niet afgekoppeld wordt, komt de lijn weer in positie 0 terecht.

Het dragen van een zwemvest is verplicht. Sommige sporters kunnen moves, zoals een jump, omdraaien, op één been skiën, en met één hand de lijn vasthouden. Iedere move levert een aantal punten op. Een waterskiër mag meerdere rondjes achter elkaar maken, het aantal rondjes dat hij nog te gaan heeft, wordt bijgehouden. Hieronder zie je een en ander weergegeven in een domeinmodel in de vorm van een UML-klassendiagram.



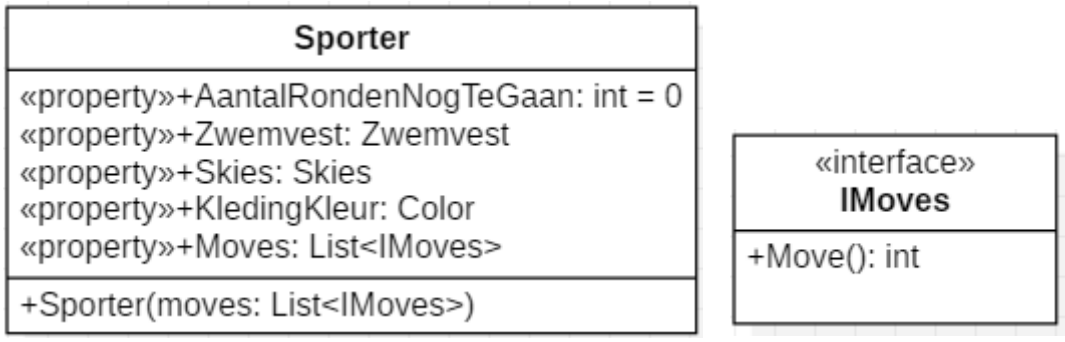
In eerste instantie gaan jullie deze opdracht uitwerking in de vorm van een Console-applicatie (.Net Framewok). In opgave 13 gaan jullie één en ander visualiseren. Dan gaan jullie de geschreven code integreren met een WPF-applicatie.

Let op!: Ten tijde van het schrijven van deze opdracht is er nog geen .Net Core versie voor WPF beschikbaar. Om vanuit de WPF applicatie de Console-applicatie te kunnen benaderen, dient dit een .Net Framework applicatie te zijn!

Opgave 1 - Basis klassen Gezamenlijk

Een aantal domeinklassen worden in de implementatie één op één overgenomen uit het domeinmodel. De moves worden vooralsnog geïmplementeerd middels een Interface.

Implementeer de klassen Sporter en het interface IMoves als volgt:



Zoals je ziet is er een constructor voor een Sporter, waarbij een lijst met moves meegegeven kan worden.

De volgende klassen implementeer je zoals in het domeinmodel is weergegeven:

- Lijn
- Skies
- Zwemvest

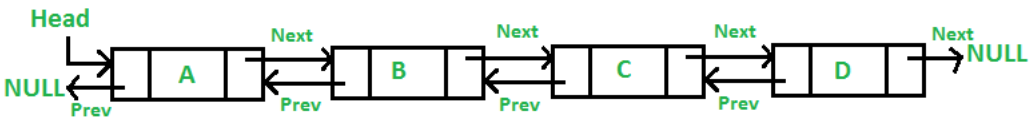
Uiteraard zet je de in het domeinmodel genoemde attributen om naar properties.

Opgave 2 Klasse Kabel – Gezamenlijk

De waterskibaan bestaat uit lijnen, een lijnenvoorraad en een kabel. In de simulatie worden lijnen uit de voorraad gehaald en op de kabel geplaatst. Als een sporter afhaakt aan het eind van de baan, dan wordt de lijn weer in de lijnenvoorraad geplaatst. In deze opgave wordt de klasse Kabel geïmplementeerd.

De op de kabel geplaatste lijnen worden bijgehouden in een `LinkedList`. Een `LinkedList` heeft als kenmerk dat een node in de lijst attributen heeft die verwijzen naar het vorige en het volgende element. De eerste node wijst niet naar een vorige node, maar naar null. Verder wijst de laatste node niet naar een volgende node, maar naar null. Zie ook hoofdstuk 19.4 Linked Lists p. 815

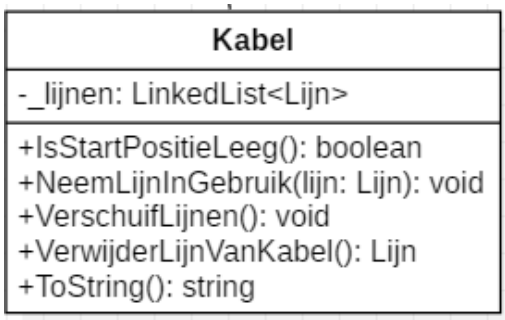
LINKEDLIST



Een node van de LinkedList is een Lijn die aan de kabel is gekoppeld. De volgorde waarin de lijnen in de LinkedList zijn opgenomen komt overeen met de volgorde van de lijnen op de kabel.

De kabel verschuift telkens, dus de lijnen die gekoppeld zijn aan de kabel krijgen steeds een nieuwe positie.

Implementeer nu de klasse Kabel:



IsStartPositieLeeg : bool
Deze methode gaat gebruikt worden om te bepalen of een lijn op dat moment aangekoppeld kan worden. Dat kan als er nog helemaal geen lijnen aan de kabel gekoppeld zijn, of als geen van de gekoppelde lijnen zich op positie 0 bevindt. Omdat de lijnen op basis van positie gesorteerd zijn in de LinkedList, hoef je dus eigenlijk alleen de positie van de eerste Node te bepalen.
Retourneer `true` als positie 0 leeg is, anders `false`.

NeemLijnInGebruik(Lijn: lijn) : void
Een lijn die in gebruik wordt genomen, krijgt altijd positie 0. De lijnen moeten in volgorde staan van positie (bijvoorbeeld bij vier nodes: 0,1,5,8). Neem de lijn alleen in gebruik als positie 0 op de kabel vrij is, anders lijn niet toevoegen.
Voeg de lijn toe aan de `LinkedList`.

VerschuifLijnen() : void
Verhoog de positie van alle lijnen in `_lijnen` met 1. De positie van een lijn kan nooit hoger worden dan 9. Een lijn op positie 9 die moet opschuiven, krijgt positie 0, zodat deze lijn weer aan een volgend rondje kan beginnen.

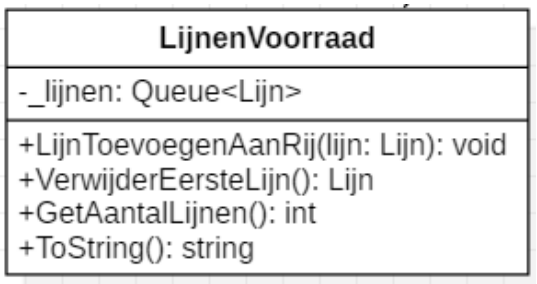
VerwijderLijnVanKabel() : Lijn
Het verwijderen van een lijn. Alleen een lijn op positie 9 kan van de kabel gehaald worden. De methode retourneert deze lijn. Als er geen lijn is die afgekoppeld kan worden, returneer dan `null`.

ToString()
Maak een `ToString` methode. De methode laat de positie van lijnen op de kabel zien. Als bijvoorbeeld positie 0, 1, 5 en 8 bezet zijn, genereerd de `ToString` methode de volgende string: 0|1|5|8. Als er geen lijnen op de kabel zijn, wordt een lege string terug gegeven.

Maak een 'private static void TestOpdracht2()' methode, en roep deze aan in je `Main()`. Toon de juiste werking van je code aan middels deze methode. Tip: als je al bekend bent met Unit-testen, test dan je code aan de hand van unit-testen!

Opgave 3 Klasse LijnenVoorraad - Gezamenlijk

De waterskibaan heeft een lijnenvoorraad. De lijnen die niet in gebruik zijn op de kabel bevinden zich in deze voorraad. De voorraad is een rij waaraan lijnen kunnen worden toegevoegd en kunnen verwijderd.
Implementeer de klasse `LijnenVoorraad` en voeg een passend override voor `ToString` toe.

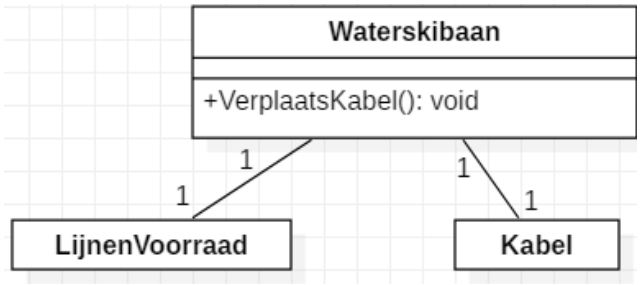


VerwijderEersteLijn(): Lijn
Indien er geen lijnen op voorraad zijn, returneer dan `null`.

ToString()
Maak een `ToString` methode. De methode returned het aantal lijnen op voorraad aangevuld met de tekst " lijnen op voorraad". Dus als er nog 7 lijnen op voorraad zijn, wordt de tekst "7 lijnen op voorraad" gereturned.

Maak een 'private static void TestOpdracht3()' methode, en roep deze aan in je `Main()`. Toon de juiste werking van je code aan middels deze methode. Tip: als je al bekend bent met Unit-testen, test dan je code aan de hand van unit-testen!

Opgave 4 Klasse Waterskibaan – Gezamenlijk



De klasse Waterskibaan heeft een methode `VerplaatsKabel`. Deze methode zorgt ervoor dat:

1) De lijnen één positie worden verschoven .
2) De lijn op de laatste positie (9) wordt los gemaakt van de kabel en toegevoegd aan de lijnenvoorraad (dus vooralsnog gaan we er van uit dat een sporter maar één rondje maakt en dan al losgekoppeld wordt).

In de constructor van de Waterskibaan wordt ervoor gezorgd dat de lijnenvoorraad initieel beschikt over 15 lijnen.

Maak een `ToString` methode. De methode returned een overzicht van de lijnenvoorraad en van de kabel.

Opgave 5 Implementaties IMove - Individueel

Sommige sporters kunnen moves, zoals een jump, omdraaien, op één been skiën, en met één hand de lijn vasthouden. Verzin een aantal implementaties voor `IMove`, spreek af wie welke moves maakt. Voor een goed uitgevoerd move krijgt een sporter het aantal voor die move vastgestelde punten.

Bij het uitvoeren van een move kan het gebeuren dat een move mislukt, dan worden helemaal geen punten toegekend. Bepaal random of een move lukt.

Maak een static klasse MoveCollection met een methode GetWillekeurigeMoves, die een willekeurig lijst van moves oplevert. De constructor van de klasse Sporter heeft een parameter moves. Bij het instantiëren van een sporter wordt een willekeurig aantal moves uit de MoveCollection aan moves als input gegeven.

Voor een sporter wordt er bijgehouden hoeveel punten er al behaald zijn. Voeg het aantal behaalde punten als attribuut toe aan de klasse Sporter.

Commit en push je code.

Opgave 6 klasse Sporter, Waterskibaan - Gezamenlijk

Implementeer de methode `Waterskibaan.SporterStart(Sporter sp)` die verantwoordelijk is voor het starten van een sporter.

Sporters starten op positie 0 op de waterskibaan. Zij kunnen alleen starten als er op die positie geen lijn is. Als dat zo is wordt de sporter aan een lijn uit de lijnenvoorraad gekoppeld en de lijn aan de kabel gekoppeld. Dan wordt ook bepaald (random) of een sporter één of twee rondjes gaat Skiën.

De sporter heeft ook kleding aan. Deze wordt gebruikt bij Opgave 13. Voor nu moet de variable `KledingKleur` een willekeurige waarde krijgen in de constructor.

Opgave 7 klasse Kabel – Gezamenlijk

De klasse Kabel moet zo worden aangepast dat de sporter meerdere rondjes kan gaan Skiën.

De methode `VerwijderLijnVanKabel` verwijdert alleen een lijn als het `AantalRondesNogTeGaan` van de sporter op 1 staat.

De methode `VerschuifLijnen` verlaagt het `AantalRondenNogTeGaan` als de positie van de lijn waar de sporter aan gekoppeld is, van 9 weer naar 0 gaat.

Opgave 8 Skies en Zwemvest, controle bij starten - Gezamenlijk

Implementeer de klassen `Skies` en `Zwemvest`

Een sporter die start moet een zwemvest en skies hebben. Controleer dit in de methode `Waterskibaan.SporterStart` en gooi een exceptie als één van beiden niet in orde is

Maak een 'private static void TestOpdracht8()' methode, en roep deze aan in je Main(). Toon de juiste werking van je code aan middels deze methode. Tip: als je al bekend bent met Unit-testen, test dan je code aan de hand van unit-testen!

Commit en push je code

Opgave 9 interfaces IWachtrij – Gezamenlijk

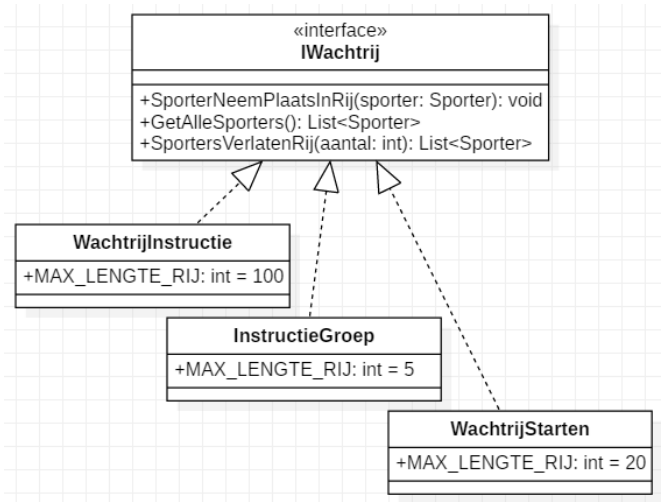
Interface IWachtrij

Een sporter kan niet zomaar starten. Voordat een sporter mag beginnen moet hij een instructie volgen. Vervolgens moet er een lijn vrij zijn. De instructies worden in groepjes gegeven. Instructies worden doorlopend gegeven. Als de sporter door de entree heen is, moet hij wachten tot de eerst volgende instructie. Als je pech hebt moet je nog even wachten, want er kunnen maximaal 5 sporters deelnemen aan een instructie.

De applicatie kent drie verschillende wachtrijen:

- `WachtrijInstructie`
- `InstructieGroep`
  n.b.: Omdat het wel zo eerlijk is om degene die het eerst in de rij stond voor de instructie, straks ook weer het eerst bij de rij voor het starten te laten aansluiten, implementeren we de InstructieGroep ook als een wachtrij.
- `WachtrijStarten`

Deze drie klassen implementeren alle het interface `IWachtrij`:



Gezamenlijk implementeer je nu dit interface IWachtrij. In de volgende opgave gaat iedere student één van de implementaties uitwerken.

Commit en push je code.

Opgave 10 implementaties interfaces IWachtrij – individueel

Zoals je uit het domeinmodel kunt afleiden, bevat iedere wachtrij een queue van sporters.

Implementeer de klasse `WachtrijInstructie`. Implementeer de klasse `InstructieGroep`. Implementeer de klasse `WachtrijStarten`. Commit en push je code.

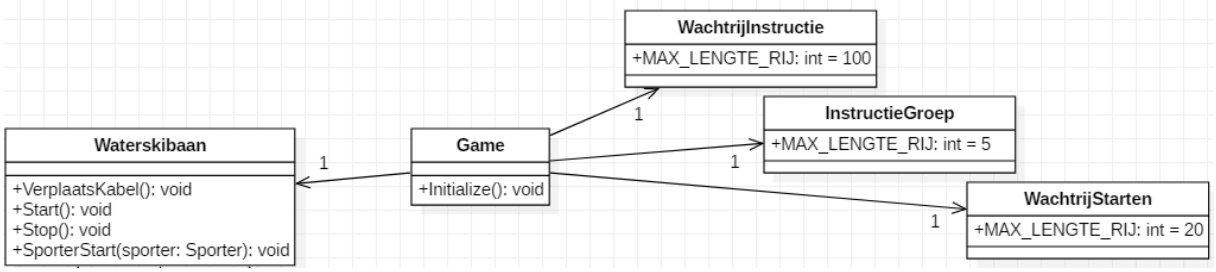
Is het wellicht een goed idee om een abstracte klasse Wachtij te definiëren waarvan al de drie implementaties erven? Zo ja, refactor dan je code.

Maak voor iedere wachtrij een ToString implementatie.

Maak een 'private static void TestOpdracht10()' methode, en roep deze aan in je Main(). Toon de juiste werking van je code aan middels deze methode. Tip: als je al bekend bent met Unit-testen, test dan je code aan de hand van unit-testen!

Opgave 11 Game – Gezamenlijk

Implementeer de klasse `Game`. De methode Initialize instanteert de Waterskibaan en de andere klassen zoals je ziet in het onderstaande UML diagram.



De klasse Game stuurt de gehele game aan. Hiervoor moet een gameloop worden gemaakt. De gameloop wordt geïmplementeerd in de Initialize methode.

De gameloop kun je implementeren met een loop met een `Thread.sleep`, maar een mooiere implementatie is een gameloop die gebruik maakt van `System.Timers.Timer` (https://docs.microsoft.com/en-us/dotnet/api/system.timers.timer?view=netframework-4.7.2).

Voorlopig houden we de simulatie eenvoudig. Elke seconde wordt er een sporter gemaakt, en die sporter wordt gestart (Waterskibaan.SporterStart(..)). Direct daarna wordt de `Waterskibaan.VerplaatsKabel` aangeroepen en de ToString van de Waterskibaan aangeroepen. Voorlopig doen we nog even niets met de wachtrijen.

Maak een 'private static void TestOpdracht11()' methode, en roep deze aan in je Main(). Toon de juiste werking van je code aan middels deze methode. In de methode maak je een instantie van `Game aan` en roep je de methode `Initialize` aan.

Opgave 12 Events wachtrijen- Individueel

Af en toe komt er een bezoeker bij de waterskibaan aan. Nadat er een instantie van de bezoeker is gemaakt, neemt deze plaats in de rij voor de instructie.

Declareer in `Game` een delegate met de signatuur `public delegate void NieuweBezoekerHandler(NieuweBezoekerArgs args)` en het event `public event NieuweBezoekerHandler NieuweBezoeker.` Bedenk zelf een goede implementatie voor de klasse NieuweBezoekerArgs. Zorg ervoor dat het event af en toe wordt uitgevoerd (bijv. ongeveer eens per 3 seconde; maak gebruik van de gameloop) en dat een nieuwe sporter in het event wordt meegeven als een parameter (NieuweBezoekerArgs).
Zodra het event is uitgevoerd moet de `WachtrijInstructie` het event ontvangen en de nieuwe bezoeker in de wachtrij plaatsen.

Als het event `InstructieAfgelopen` (dit event wordt door student 2 uitgewerkt) wordt uitgevoerd, dan worden maximaal vijf studenten verplaatst vanuit de `WachtrijInstructie` naar de `InstructieGroep`.

De sporters ontvangen instructie voordat ze de baan op mogen. De sporters die instructie krijgen bevinden zich in de verzameling `InstructieGroep`.

Declareer in `Game` een delegate met de signatuur `public delegate void InstructieAfgelopenHandler(InstructieAfgelopenArgs args)` en het event `public event InstructieAfgelopenHandler instructieAfgelopen`. Bedenk zelf een goede implementatie voor de klasse InstructieAfgelopenArgs. Zorg ervoor dat het event af en toe wordt uitgevoerd (bijv. ongeveer eens per 20 seconde; maak gebruik van de gameloop) en dat een lijst met sporters die de instructiegroep verlaten als parameter (InstructieAfgelopenArgs) wordt meegegeven.

Na de instructie nemen de sporters plaats in de `WachtrijStarten`. In onze applicatie wordt dat als volgt geïmplementeerd. Nadat het event `InstructieAfgelopen` is uitgevoerd, worden de geïnstrueerde sporters, die beschikbaar zijn in de InstructieAfgelopenArgs van het event `InstructieAfgelopen`, in de `WachtrijStarten` geplaatst.

In Opgave 4 is het starten en stoppen geïmplementeerd. Zodra de baan gestart is worden de lijnen verschoven. Voeg een event toe `LijnenVerplaatst`, bepaal zelf een geschikte locatie voor dit event.
Zorg ervoor dat het verplaatsen van de kabel af en toe wordt uitgevoerd (bijv. ongeveer eens per 4 seconde; maak gebruik van de gameloop) en dat dan ook het even LijnenVerplaatst wordt aangeroepen.

Als de startpositie op de kabel vrij is, dan wordt een sporter wordt uit de WachtrijStarten verwijderd. De sporter ontvangt skies en een zwemvest.

Daarna wordt er een lijn uit de LijnenVoorraad gehaald, en aan de kabel gekoppeld. De sporter wordt aan deze lijn gekoppeld. Hiervoor is de methode SporterStart uit de klasse Waterskibaan reeds beschikbaar.

Maak een ‘private static void TestOpdracht12()’ methode, en roep deze aan in je Main(). Toon de juiste werking van je code aan middels deze methode. In de methode maak je een instantie van `Game aan` en roep je de methode `Initialize` aan. Laat zien dat de verschillende wachtrijen gevuld worden en dat sporters van de ene naar de andere wachtrij verplaatsen.

Een sporter die bezig is op de baan moves kan doen, doet ook af en toe één van die moves. Als een sporter geen moves heeft, dan wordt er natuurlijk geen move uitgevoerd. Elke keer als de kabel zich verplaatst (methode VerplaatsKabel), moet er een kans van 25% zijn dat een sporter één van zijn moves doet. De move die wordt uitgevoerd door een sporter die wordt in een publieke property van de klasse Sporter vastgelegd: `HuidigeMove`. In de volgende opgave wordt daarvan gebruik gemaakt, zodat de move gevisualiseerd kan worden.

Commit en push je code.

Opgave 13 Visualiseren – Gezamenlijk

Bedenkt een schermontwerp van de waterskibaan. Dit schermontwerp moet voldoen aan de volgende eisen:

- Een lijn krijgt een nummer dat zichtbaar is op de baan.
- De twee wachtrijen zijn zichtbaar met daarin de aanwezige sporters per wachtrij.
- De instructiegroep is zichtbaar, met daarin de aanwezige sporters.
- Het verplaatsen van de lijnen.
- De sporters moeten er uniek uitzien, hiervoor kun je kledingkleur gebruiken.
- De moves die sporters op dat moment uitvoeren, zie ook Opgave 12.
- De hoeveelheidlijnen in de lijnenvoorraad (dit mag gewoon een getal zijn).

Om het bovenstaande te kunnen realiseren maak je in je solution een WPF applicatie. Vanuit deze WPF applicatie refereer je aan je aan het project dat je gemaakt hebt in opgave 1 t/m 12.

Opgave 14 Dashboard queries – Gezamenlijk

Voor Opgave vijftien is logging van de bezoekers nodig. Die wordt in deze opgave gemaakt.

Maak een nieuwe class Logger aan. Zorg ervoor dat de class Game een instantie heeft van Logger. De class Logger houdt een lijst bij van bezoekers. Maak hiervoor een NieweBezoeker event. Op het moment dat de NieuweBezoeker event wordt uitgevoerd moet de bezoeker ook worden toegevoegd aan de lijst.

De klasse Logger heeft ook toegang tot de kabel.

Opgave 15 Linq – Gezamenlijk

De volgende informatie moet worden weergegeven op het scherm.

1) Hoeveel bezoekers er in totaal zijn geweest.
2) De hoogste score van een sporter voor succesvol uitgevoerde moves.
3) Hoeveel bezoekers er zijn geweest met rode kleding. (zie tip 1 hieronder)
4) Een lijst van tien bezoekers gesorteerd op de lichtheid van de kleding. De sporter met de lichtste kleding staat bovenaan. (zie tip 2 hieronder)
5) Het totaal aantal rondjes dat alle bezoekers hebben gedaan.
6) Een lijst van de unieke moves van de bezoekers die aan een `Lijn` gekoppeld zijn.

De Opgave moet worden uitgevoerd met Linq.

Het boven water halen van de informatie moet gebeuren in de Logger class.

Tips:

1. Je kunt bepalen of een Color op een andere kleur lijkt m.b.v. de volgende code:

```
private bool ColorsAreClose(Color a, Color z, int threshold = 50)
{
    int r = (int)a.R - z.R,
        g = (int)a.G - z.G,
        b = (int)a.B - z.B;
    return (r * r + g * g + b * b) <= threshold * threshold;
}
```

2. Je kunt van een Color de R-waarde in het kwadraat + de G-waarde in het kwadraat + de B-waarde in het kwadraat bepalen. Hoe hoger deze waarde is, des te lichter is de kleur.