Project Hotel

Voor de Hotel Simulatie die voor het eerste blok in jaar 2 van HBO-ICT Zoetermeer gemaakt moet worden, hebben we een aantal keuzes gemaakt die van toepassing op de code zijn. Ook voor verschillende onderdelen van het hotel hebben we een paar keuzes gemaakt die beter gedocumenteerd kunnen worden dan met wat uitge-commente code. Vandaar dat daar een apart document voor is gemaakt.

Contents

[Lift 3](#_Toc527623063)

[Interactie met de klant 3](#_Toc527623064)

# Lift

Aan de lift waren een aantal requirements aan gebonden. De lift moest namelijk een slimme lift zijn, oftewel: de lift moest efficiënt klanten ophalen en wegbrengen naar hun verdieping zoals een echte lift dat ook doet. Daarvoor zijn we door een aantal designs gegaan, en hebben we besloten om de lift te laten werken met twee **HashSets<ElevatorShaft>** om de verdiepingen waar die naartoe moet bij te houden, een **Enum** voor de directie waar die op gaat en om de **int PositionY** te gebruiken om aan te geven op welke verdieping die zicht bevindt.

## Interactie met de klant

Omdat een klant natuurlijk sommige dingen niet kan weten van een lift (bijvoorbeeld welke verdiepingen die allemaal af moet gaan) hebben we gekozen om sommige elementen **private** te houden (de 2 **HashSets<ElvatorShaft>** en de **Enum**). De klant kan alleen informatie van de lift halen wanneer de **GetElevatorInfo()** methode is aangeroepen die in de Elevator class staan. Hierbij wordt gebruik gemaakt van de **System.ValueTuple** NuGet Package, waardoor meerdere waardes in een **Tuple** ge-returned kunnen worden *(zie de code hieronder)*.

public (char, int) GetElevatorInfo()

{

return (Direction, PositionY);

}

*GetElevatorInfo() methode uit de Elevator.cs class*

Deze methode kan aangeroepen worden de klant. De **(char, int)** voor de **GetElevatorInfo()** is de **Tuple** die ge-returned wordt naar de klant wanneer de methode wordt aangeroepen. De lift geeft dan de kant waar die toe gaat terug en de huidige **PositionY** (verdieping) waar die is. Met deze twee gegevens kan de klant uitrekenen welke route die moet nemen (via de trap of met de lift, zie hoofdstuk ***Pathfinding*** voor meer informatie).

## Route calculatie

De lift in het hotel is dus een “slimme” lift, maar wat maakt een lift nou “slim”? Dat is wat we ons eerst af gingen vragen voordat we begonnen met het schrijven van de **Elevator** code. Om uit te zoeken wat een slimme lift nou daadwerkelijk is, hebben we besproken hoe een lift klanten zo efficiënt mogelijk kan ophalen. Hierbij zijn we tot de conclusie gekomen dat de lift hiervoor 2 **Lists** hiervoor kan gebruiken, een **List** die bijhoud welke verdiepingen hij af moet gaan wanneer de lift omhoog gaat en een **List** die behoud welke verdiepingen hij af moet gaan wanneer die naar beneden gaat. We zijn begonnen met dit op te slaan in een **List<>**, maar zijn daarna op de hoogte gebracht van Tim Cocx *(leraar HBO-ICT Zoetermeer, onze Projectbegeleider voor dit project)* dat een **HashSet<>** een stuk makkelijker is. Voordat we de **HashSet<>** gebruikte, gebruikte we namelijk **System.Linq** om de **List<>** te sorteren, maar ook om duplicates eruit te halen.

*UpdateList() methode uit de Elevator.cs class*

private void UpdateList(char Direction, int CurrentFloor, List<ElevatorShaft> Route)

{

foreach (ElevatorShaft Shaft in Route)

{

if (Direction == 'U')

{

Up = Route.Distinct().OrderBy(x => x.PositionY).ToList();

}

else if (Direction == 'D')

{

Down = Route.Distinct().OrderByDescending(x => x.PositionY).ToList();

}

}

}

Hierboven is te zien hoe de **List**’s worden ge-update elke keer dat een **ElevatorShaft** werd toegevoegd. Door een **HashSet<>** te gebruiken in plaats van een **List<>** kan de **.Distinct()** query weggehaald worden, want een er kunnen geen duplicate waardes toegevoegd worden aan een **HashSet<>**. Voor de rest wordt de lijst geordend op de **PositionY** waarde van de **ElevatorShaft**