Opdrachten bij textadventure

Dit document beschrijft de opdrachten voor de textadventure.

Belangrijk: Gebruik de exacte namen die in dit document staan.
Dit geldt voor:

- Classes
- Methods
- Properties

Inhoudsopgave

1 User stories voor de Textadventure

Een **user story** beschrijft wat een speler wil kunnen doen.
Implementeer de user stories door de opdrachten te maken.

User stories bestaande functionaliteit

Dit kan de textadventure al:

Verken de Kamers:

- Als speler wil ik kamers kunnen verkennen.
- Ik gebruik commando's: "go north", "go south", "go east", "go west".
- Ik wil een beschrijving van de huidige kamer zien.
- Ik wil weten welke uitgangen er zijn.

Ontvang Hulp:

- Als speler wil ik hulp kunnen vragen.
- Ik typ "help" om de beschikbare commando's te zien.

Ongeldig Commando:

- Als speler wil ik feedback krijgen.
- Dit gebeurt als ik een verkeerd commando typ.

Stoppen met spelen:

- Als speler wil ik kunnen stoppen.
- Ik typ "quit" om het spel te beëindigen.

User stories nieuwe functionaliteit

Dit ga je toevoegen:

Bekijk de Kamer:

- Als speler wil ik het "look" commando gebruiken.
- Dit toont een beschrijving van de omgeving.
- Ik zie welke items er liggen.

Bewegen tussen Verdiepingen:

- Als speler wil ik naar boven kunnen gaan ("up").
- Ik wil naar beneden kunnen gaan ("down").
- Dit werkt in kamers met meerdere verdiepingen.

Mogelijk meerdere Spelers:

- Als speler wil ik in de toekomst met anderen kunnen spelen.
- Meerdere spelers kunnen in dezelfde kamer zijn.

Speler Status:

- Als speler verlies ik gezondheid als ik naar een andere kamer ga.
- Ik kan mijn gezondheid bekijken met "status".
- Met "status" zie ik ook mijn backpack inhoud.

Items:

- Als speler kan ik items in kamers vinden.
- Ik kan items oppakken en gebruiken.
- Mijn backpack heeft een maximaal gewicht.
- Ik kan alleen items dragen binnen dat gewicht.
- Ik gebruik "take" om items op te pakken.

- Ik gebruik "drop" om items neer te leggen.
- Ik gebruik "use" om items te gebruiken.

Derde woord in Commando's:

- Als speler kan ik drie woorden gebruiken.
- Bijvoorbeeld: "use key east".
- Dit geeft meer mogelijkheden.

Endgame:

- Als speler kan ik het spel winnen.
- Ik los problemen of puzzels op.
- Of ik versla vijanden.

2 Opdrachten

Breid de code uit met nieuwe classes.

Tips voor de verhaallijn:

- Houd het kort en krachtig.
- Maximaal enkele regels tekst per Room.
- Ongeveer 10 Rooms is genoeg.
- Een klein verhaaltje is leuk.
- Maar je schrijft code, geen boek!

2.1 Look

Implementeer het "look" commando.
Dit toont opnieuw de beschrijving van de kamer.
Handig als de speler dit vergeten is.

Bestanden die je moet wijzigen:

- CommandLibrary.cs
- Game.cs

2 Up and down

Voeg "up" en "down" toe als uitgangen.
Sommige Rooms hebben meerdere verdiepingen.

Bestanden die je moet wijzigen:

- Game.cs

3 Player

Maak een Player class.
Een Player heeft een currentRoom.

```
class Player
{
  // auto property
  public Room CurrentRoom { get; set; }

  // constructor
  public Player()
  {
    CurrentRoom = null;
  }
}
```

Nieuwe bestanden:

- Player.cs

Bestanden die je moet wijzigen:

- Game.cs

De currentRoom is nu onderdeel van Player.
De Game class heeft alleen nog een parser en een player.
Haal currentRoom weg uit Game.

```
class Game
{
  private Parser parser;
  private Player player;

  public Game ()
  {
    parser = new Parser();
    player = new Player();
    CreateRooms();
  }
}
```

Pas ook CreateRooms() aan:

```
private void CreateRooms()
{
  // ...
  player.CurrentRoom = outside;
}
```

Er zijn meerdere plekken waar currentRoom werd gebruikt.
Die geven nu errors.
Verander dit naar: player.CurrentRoom

Oude situatie:

Nieuwe situatie:

Lees de file over UML om deze diagrammen te begrijpen.

Player status

Voeg een health field toe aan Player.

```
class Player
{
  // fields
  private int health;

  // constructor
  public Player()
  {
    health = 100;
  }
}
```

Maak deze methods in Player:

```
class Player
{
  // methods
  ... Damage(... amount) { /*...*/ } // speler verliest
health
  ... Heal(... amount) { /*...*/ } // speler krijgt health
  ... IsAlive() { /*...*/ } // checkt of speler nog leeft
}
```

Vraag: Welk type zijn de argumenten?
Welk type zijn de return waarden?
Vul de juiste keywords in bij de `...`.
Implementeer de methods.

Maak ook een "status" commando.
Dit toont hoeveel health de Player heeft.

Bestanden die je moet wijzigen:

- Player.cs
- CommandLibrary.cs
- Game.cs

2.4 Gewond

De speler is gewond.
Elke keer als de speler naar een andere kamer gaat, verliest deze health.
Implementeer dit.

Het spel is afgelopen als de speler dood is.
Controleer in Game.cs in de Play() method of de speler nog leeft.
Als de Player dood is, mag deze geen commando meer geven.

Implementeer ook dat de speler kan winnen.
Dan eindigt het spel ook.

Bestanden die je moet wijzigen:

- Game.cs

2.5 Items

In een Room liggen mogelijk Items.
De speler kan Items oppakken.

Maak de class Item:

```
class Item
{
  // fields
  public int Weight { get; }
  public string Description { get; }

  // constructor
  public Item(int weight, string description)
  {
    Weight = weight;
    Description = description;
  }
}
```

Nieuwe bestanden:

- Item.cs

Maak een paar Items in CreateRoom() in Game.

2.6 Inventory

Maak de class Inventory.
Een Inventory bevat een lijst met Items.
Gebruik het type `Dictionary`.

Een Inventory kan maximaal xx kilo bevatten.
Implementeer ook de controle op het maximale gewicht.

> **Tip:** Maak eerst simpele versies van Put() en Get().
> Maak ook alvast Take() en Drop() in Game.cs.
> Zie Take and drop.
> Test je code stap voor stap.
> Voeg daarna de gewicht-controle toe.

```
class Inventory
```

```
{
  // fields
  private int maxWeight;
  private Dictionary<string, Item> items;

  // constructor
  public Inventory(int maxWeight)
  {
    this.maxWeight = maxWeight;
    this.items = new Dictionary<string, Item>();
  }

  // methods
  public bool Put(string itemName, Item item)
  {
    // TODO implementeer:
    // Check het gewicht van het Item
    // Is er genoeg ruimte in de Inventory?
    // Past het Item?
    // Zet Item in de Dictionary
    // Return true/false voor succes/mislukt

    return false;
  }

  public Item Get(string itemName)
  {
    // TODO implementeer:
    // Zoek Item in de Dictionary
    // Verwijder Item uit Dictionary (als gevonden)
    // Return Item of null

    return null;
  }
}
```

Om te weten hoeveel ruimte er nog is, maak je hulp-methods:

```
class Inventory
{
  // methods
  public int TotalWeight()
  {
    int total = 0;

    // TODO implementeer:
    // Loop door alle items
    // Tel alle gewichten op

    return total;
  }

  public int FreeWeight()
  {
    // TODO implementeer:
    // Vergelijk MaxWeight en TotalWeight()

    return ...;
  }
}
```

Nieuwe bestanden:

- Inventory.cs

Inventory voor Room

Een Room heeft een Inventory.
We noemen dit een "chest" (kist).

Je kunt een getter maken voor de Inventory.
Iedereen mag zien welke Items er liggen.
Een setter is niet nodig.

```
class Room
```

```
{
  // field
  private Inventory chest;

  // property
  public Inventory Chest
  {
    get { return chest; }
  }

  // constructor
  public Room(string description)
  {
    // een Room kan veel items bevatten
    chest = new Inventory(999999);
  }
}
```

Bestanden die je moet wijzigen:

- Room.cs

Inventory voor Player

In Player maak je TakeFromChest() en DropToChest().
Hier handel je de uitwisseling van Items af.
Ook communiceer je met de speler.

Voorbeelden:

- "Item is not in Room"
- "Item doesn't fit in your inventory"
- "You don't have that Item"

De Inventory van de Player noemen we "backpack".
De backpack is private.
Niemand mag daar zomaar bij.

```
class Player
{
  // field
  private Inventory backpack;

  // constructor
  public Player()
  {
    // 25kg is best zwaar om de hele dag te dragen
    backpack = new Inventory(25);
  }

  // methods
  public bool TakeFromChest(string itemName)
  {
    // TODO implementeer:
    // Haal het Item uit de Room
    // Zet het in je backpack
    // Bekijk de return values
    // Past het Item niet? Zet het terug in de chest
    // Laat de speler weten wat er gebeurt
    // Return true/false voor succes/mislukt

    return false;
  }

  public bool DropToChest(string itemName)
  {
    // TODO implementeer:
    // Haal Item uit je backpack
    // Zet het in de Room
    // Bekijk de return values
    // Laat de speler weten wat er gebeurt
    // Return true/false voor succes/mislukt

    return false;
  }
```

```
    }
```

Vraag: Hoe kan de speler zien welke Items in de Room liggen?
Hoe kan de speler zien welke Items in de backpack zitten?

Maak een Show() method in Inventory.
Deze returnt een string met alle Items in de Inventory.

Breid het "status" commando uit.
Laat ook zien welke Items de Player in de backpack heeft.

Bestanden die je moet wijzigen:

- Inventory.cs
- Player.cs
- Game.cs
- CommandLibrary.cs

2.7 Take and drop

Nu kun je de "take" en "drop" commando's testen.
Maak deze in de class Game.

```
class Game
{
  // methods
  private void Take(Command command)
  {
    // TODO implementeer
  }

  private void Drop(Command command)
  {
    // TODO implementeer
  }
}
```

Bestanden die je moet wijzigen:

- Game.cs
- CommandLibrary.cs

Het class diagram ziet er nu zo uit:

2.8 Use an Item

Hoe gebruikt de Player een Item?
Maak de method Use(string itemName) in Player.

```
class Player
{
  // methods
  public string Use(string itemName)
  {
    // TODO implementeer
  }
}
```

Maak ook het "use" commando.

Bestanden die je moet wijzigen:

- Player.cs
- Game.cs
- CommandLibrary.cs

2.9 Third Command Word

Voeg een "thirdWord" toe aan Command.
Je wilt immers kunnen tikken:

```
use key east
```

Of, als je meerdere wapens hebt:

```
attack guard axe
```

Implementeer dit.

Bestanden die je moet wijzigen:

- Command.cs
- Parser.cs

3 Endgame

Dit is een goed moment voor extra functionaliteit.

Ideeën:

Een gezonde speler kan gewond raken:

- Gebruikt de speler een "bad Item"?
- Liggen er medkits in de Rooms?

Guards in Rooms:

- De speler moet vechten met een guard.
- Anders komt de speler er niet langs.

Locked Rooms:

- Sommige Rooms zijn op slot.
- De speler heeft een sleutel nodig.

Vraag: Wat kun jij nog meer toevoegen aan je textadventure?