# Overzicht lessen PROG

P1

Herhaling : Functions, Classes en Arrays (op 123) **les 1**

Herhaling : Action event (op 4) **les 2**

Debugging, Breakpoints (op 5) **les 3**

DRY & SRP (op 6 op 7) **les 4**

OOP: Inheritance (op 8) **les 5**

OOP: Encapsulation (getter, setters) (op 9) **les 6**

Class diagrams (SD2A) (op11) **les 7**

Wk 9 (8 nov)

+ Class diagrams (SD2B) (op11)

P2

Wk 1

Toetsen Basics <https://forms.office.com/e/ZPcvqSNxDT> (op 12)

Wk 2

OOP: Abstraction & Polymorphism (abstract class, override) **les 8**

Wk3

OOP: Abstraction & Polymorphism (abstract class, override) (op 10)

Wk4

Code Conventions (op 13) **les 9**

Wk 5

Uitval i.v.m kerst

Wk 6

Enums (op 14) **les 10**

Wk 7

Delegates (op 15) **les 11**

Wk 8

Stack & Heap , garbage collection (op 16) **les 12**

Wk 9

Keuze: (op 17) **les 13**

Singleton Pattern (makkelijker)

Object pooling (moeilijk)

Factory?

..

Design patterns – Factory Method

Delegates

Profiling?GPU instancing?

# Lesrooster aandachtspunten

SD2B 1e les woensdag blok 1:

* evt studenten te laat door trein
* Studenten nog fris
* +-20 studenten

SD2A 1e les woensdag blok 2:

* Studenten hebben in blok 1 aan project gewerkt, even schakelen dus
* +-20 studenten

SD2A 2e les vrijdag blok 1:

* evt studenten te laat door trein
* Studenten nog fris
* +-20 studenten

SD2B 2e les woensdag blok 2:

* Studenten hebben in blok 1 aan wiskunde gewerkt, even schakelen dus
* +- 20 studenten

# Lesplan les 1 : Herhaling : Functions, Classes en Arrays

## Doelen:

De studenten kunnen **functies** schrijven, **argumenten** aan deze functies meegeven en de functies ook weer data terug laten geven(**return**).

De studenten weten dat een **Method** ook een functie is die voor een object kan worden uitgevoerd.

De studenten weten wat een **class** is en dat hiermee objecten geïnstantieerd kunnen worden.

De studenten weten hoe je in C# normaal een **class** maakt en een **constructor**.

De studenten kunnen werken met de **Instantiate** functie van Unity ipv constructors aanroepen en begrijpen dat dit hetzelfde doel heeft.

De studenten weten hoe ze een **Array** en een **List** kunnen maken en wat het verschil is tussen de Array en de List.

De studenten kunnen door lijsten heen loopen.

## Situatie:

De studenten hebben deze onderwerpen vorig schooljaar al behandeld in de PROG lessen. Echter zal bij velen deze informatie niet goed zijn blijven hangen..

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

* **functie** opdracht
* **class** opdracht
* **array/List** opdracht

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen. Studenten leven de opdracht in via simulise

Studenten die kennis van onderdelen nog missen uitleg en oefenmateriaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 1 : Herhaling : Functions, Classes en Arrays

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:10 | Landen/Energizer | Kahoot basic programming: <https://create.kahoot.it/details/96f41b13-37bd-47b8-9953-611bfa8e96fe> |
| 00:10 – 00:20 | Orientatie vragen stellen: functies, classes, arrays | Vragen om uitleg van willekeurige student over een functie. Wat is een functie? Waarom gebruik je ze? Wat is het verschil tussen een function en een Method? Wat is een argument? Wat is een return value? |
| 00:20 – 00:50 | Opdracht uitwerken: Studenten maken zelf een **method** met **arguments** en **return values** | Opdracht 1 Functions, Methods, Parameters en return |
| 00:50 – 01:00 | terugkijken | willekeurige student laat zijn functie zien op de beamer. |
| 01:00 – 01:10 | Orienterende vragen over classes:  Wat is een class? Waarom gebruik je deze? Wat is een constructor? Gebruik je een constructor in Unity? |  |
| 01:10 – 01:40 | Studenten maken Opdracht 2 over classes en gebruiken van instantiate ipv de constructor | Opdracht 2 Classes, Constructors en Instantiate |
| 01:40 -01:45 | Terugkijken: Student laat zijn uitwering van opdracht 2 zien op de beamer. |  |
| 01:45- 01:50 | Orienterende vragen over Arrays en Lists: Wat is een Array/List? Waarom gebruik je deze? Wat is het verschil tussen een Array en List? |  |
| 01:50 – 01:55 | Huiswerk opgeven:  Afmaken van alle opdrachten en ook de opdracht over Arrays (1,2 en 3) | Opdracht 3 Array, List & Loops |
| 1:55 | **inpakken** |  |

# Lesplan les 2: Action events

## Doelen:

Verifiëren of iedereen **functions**, **classes** en **Arrays** begrijpt en of er nog vragen zijn.

De studenten kunnen een **Action Event** gebruiken om functies tussen verschillende scripts te “triggeren” en daarbij eventueel data mee te geven.

## Situatie:

Sommige studenten hebben vorig jaar al een video gevolgd over **Action Events**.

**Vorige** **les** hebben we aandacht besteed aan het herhalen van **functions**, **classes** en **arrays**. De kans is aanwezig dat hier nog veel onduidelijkheid over bestaat en dat dit dus nog aandacht nodig heeft.

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 2: Action events

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:10 | Landen/Energizer | Action Event Spel met briefjes en trigger woorden, alle studenten krijgen de op dracht om op te springen en iets over zichzelf te vertellen wat de rest nog niet weet als er een woord wordt genoemd wat op zijn briefjes staat.(action event) |
| 00:10 – 00:20 | Orientatie vragen stellen over: Action Events | Vragen aan de studenten of ze begrijpen waar het spel over ging? Vragen wie er weet wat een action event is. Vragen om een korte uitleg van een student. |
| 00:20 – 00:40 | Keuze maken: les doorlezen op readme, Video kijken over Action Events of Artikel lezen voor mensen die al een Action Event kunnen gebruiken(**verdiepen**) | Digitaal materiaal via Simulise>readme delen  <https://github.com/djsjollema/lessen-gamedevelopment/tree/main/M5/Prog/02_Herhaling_Action_Events_Debugging#opdracht-4-action-events>  <https://youtu.be/IKBg97A7Sbw?si=VZhFt4WQyNUpB3aF>  of  <https://gamedevbeginner.com/events-and-delegates-in-unity/> |
| 00:40 – 00:60 | Opdracht uitwerken: passen zelf een Action event toe | Opdracht via simulise(>github) delen |
| 01:05 – 01:10 | Terugkijken op Action Events | willekeurige student laat zijn resultaat zien op de beamer. |
| 01:10 – 01:20 | Terugkijken op **functions** door willekeurige student een recap te laten geven op **functions** |  |
| 01:20 -01:30 | Terugkijken op **classes** door willekeurige student een recap te laten geven op **classes** |  |
| 01:30 -01:40 | Terugkijken op **Arrays** door willekeurige student een recap te laten geven op **Arrays** |  |
| 01:40 – 01:45 | Iedereen de opdracht van vorige les ingeleverd? | simulize |
| 01:55 | inpakken |  |

# Lesplan les 3: Debugging & Breakpoints

## Doelen:

Checken of alle opdrachten zijn gemaakt. Is iedereen op schema? Toetsen of iedereen de basis beheerst.(**functies, classes, arrays, action events**)

De studenten begrijpen waarom breakpoints gebruiken in veel gevallen veel sneller kan zijn dan Debug.Log commands invoeren

De studenten kunnen debuggen via **breakpoints** in MS Visual studio, hierbij zijn ze specifiek in staat om na een breakpoint lijn voor lijn door de code heen te stappen en alle data te inspecteren.

## Situatie:

De studenten zijn bekend met het Debug.Log commando en hebben dat in het Mythe project vaak moeten gebruiken om te debuggen..

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 3: Debugging & Breakpoints

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:15 | Energizer: 10 verstopte bugs zoeken on het lokaal | Papieren Bugs verstoppen, elke bug heeft een prijs (kitkat) |
| 00:15 – 00:20 | Oriëntatie vragen stellen over: Debuggen  Iedereen de opdracht van vorige les ingeleverd? | Vragen aan studenten hoe ze bij Mythe bugs hebben opgespoord. Wie heeft er al eens gewerkt met breakpoints? |
| 00:20 – 00:40 | Uitleg over debuggen | Miro les <https://miro.com/app/board/uXjVNeHQqLI=/> |
| 00:40 – 1:20 | Debug opdracht uitvoeren | Projectje met een bug erin als opdracht bereikbaar en in te leveren via simulize |
| 01:20 – 01:30 | Terugkijken: wie heeft de bug gevonden? Wie heeft de bug opgelost? evt laten zien op de beamer. Evt zelf oplossing tonen | Oplossing van bug |
| 01:30 – 01:35 | Iedereen de opdracht van vorige les ingeleverd? | simulize |
| 01:35 - 01:55 | TD issues bespreken en ondersteuning in TD code. Studenten zonder vragen mogen gaan. |  |
| 01:55 | inpakken |  |
|  |  |  |
|  |  |  |

# Lesplan les 4: SRP & DRY

## Doelen:

De studenten weten waar SRP en DRY voor staat.

De studenten begrijpen waarom je SRP en DRY goed toe moet passen in je code.

De studenten herkennen wanneer code niet DRY of SRP geschreven is.

De studenten kunnen een manier bedenken hoe de code DRY en SRP kan worden gemaakt.

## Situatie:

De studenten hebben in p4 van leerjaar 1 allemaal spaghetti code geschreven en zijn veelvuldig tegen structuurproblemen aangelopen. Dit is dus een mooie kapstok om het belang van structuur aan te kaarten.

De meeste studenten kennen de termen SRP en DRY nog niet.

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 4: SRP & DRY

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:15 | Energizer: Ga van links naar rechts staan als je de meeste spaghetti code hebt geschreven tijdens mythe. Zorg voor bewijs hiervan. | Laptops van studenten |
| 00:15 – 00:20 | Oriëntatie vragen stellen over: SRP & DRY  Wat maakte je code spaghetti code? Wat was het gevolg voor je proces?  Waar staat DRY/SRP voor? Waar zou WET voor staan (Write Everything Twice / Waste Everyones Time) |  |
| 00:20 – 00:40 | Uitleg over SRP & DRY | Miro les <https://miro.com/app/board/uXjVMpW9alY=/> |
| 00:40 – 1:30 | SRP & DRY opdracht uitvoeren refactoren code naar SRP & DRY  *Lange opdracht dus even waarschuwen bij 15 min en 7 min over!* | Projectje met non SRP non DRY class als opdracht bereikbaar en in te leveren via simulize |
| 01:35 – 01:50 | Terugkijken: 3 Willekeurige studenten laten hun oplossingen op de beamer zien en lichten dit toe. Ik laat mijn oplossing ook zien. | Gerefactorde versie van de code |
| 01:50 – 01:55 | Iedereen de opdracht van vorige les ingeleverd? | simulize |
| 01:55 | inpakken |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Lesplan les 5: OOP Inheritance

## Doelen:

De studenten weten waar OOP voor staat.

De studenten weten dat OOP bestaat uit 4 basis principes: Inheritance, Abstraction, Encapsulation en Polymorphism

De student is zich bewust dat hij in Unity al vaak overerving heeft gebruikt op de **Monobehaviour** class.

De student weet wat **Inheritance** inhoud en kan zelf een eigen class overerven.

De studenten zijn zelf in staat om technisch onderzoek te doen

## Situatie:

De studenten zijn onbewust bezig geweest met inheritance namelijk op de monobehaviour class. Een mooie kans om ze hier dus bewust van te maken.

De studenten zijn nog niet bekend met OOP wellicht dan een enkeling er een vaag beeld van heeft.

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 5: OOP Inheritance

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:15 | Energizer: plak stickers op 2 objecten in het lokaal.  bedenk per object 3 eigenschappen en functies  langs 10 stickers en vragen wat de eigenschappen en functies van de objecten zijn.. | Stickers die weer makkelijk los te halen zijn |
| 00:15 – 00:20 | Oriëntatie vragen stellen over: OOP  Waar staat OOP voor? Wie heeft er wel eens van gehoord? Wat zijn Objecten eigenlijk? Wat betekend Georiënteerd?  Iedereen opdracht van vorige les ingeleverd? |  |
| 00:20 – 01:30 | In groepen (5\*4) uitzoeken wat OOP inhoud en eind van de les presenteren in een PowerPoint.  Waar staat het voor?  Waar bestaat het uit? Onderdelen  Wat is het doel van OOP?  Wat is inheritance?  Hebben jullie al eens inheritance gebruikt?  Geef een code voorbeeld van inheritance.  Werken waar je wilt | Tafels in groepsopstelling, studenten met laptops en internet. Powerpoint.(inleveren op simulise)  Groepsopdracht bereikbaar via simulise.  Google, chat gpt, youtube, vragen stellen aan docenten. |
| 01:30 – 01:55 | Presentaties van 5 min per groep (5 groepen van 4) | Beamer / scherm |
| 1:55 | Inpakken + alles ingeleverd op simulize? |  |

# Lesplan les 6: OOP Abstractie

## Doelen:

De studenten weten wat abstraction is.

De studenten begrijpen dat abstraction zorgt voor bekopte onderhoudbare en uitbreidbare code.

De studenten begrijpen het verschil tussen abstract en concreet.

De studenten kunnen een abstract class maken en deze uitbreiden met concrete classes

De studenten weten wat Interfaces zijn en kunnen deze toepassen in hun structuur

## Situatie:

Over het algemeen hebben studenten tijdens het designen van hun eigen games de neiging om ideeën abstract te verworden en niet concreet. Bijvoorbeeld: “je moet een eindbaas verslaan om de game te winnen”.

Dit is een mooi bruggetje om te maken naar de concepten abstract en concreet..

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 5: OOP Abstractie

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:15 | Energizer: studenten maken een tekening nav een abstracte instructie **“teken een eindbaas die wordt verslagen door een speler”** vs **“teken een reisachtige cycloop met 1 oog die door een halfling met een sling in zijn oog wordt geraakt en daardoor wordt uitgeschakeld”**. | Tekenvellen stiften |
| 00:15 – 00:20 | Oriëntatie vragen stellen over: Abstractie  Welke opdracht was abstract? Welke was concreet?  Wat is nu dan het verschil tussen concreet en abstract?  Wat is het verschil tussen abstracte en concrete kunst? |  |
| 00:20 – 00:40 | Uitleg over abstractie van classes | Miro bord |
| 00:40 – 01:40 | Uitvoeren opdracht abstracte en concrete classes maken | Opdracht bereikbaar via simulize en ook daar weer inleveren |
| 01:40 – 01:55 | 3 studenten laten hun uitwerking op de beamer zien en lichten dit toe. | Beamer, studenten laptops |
| 1:55 | Inpakken + alles ingeleverd op simulize? |  |

# Lesplan les 7: OOP Encapsulation

## Doelen:

De studenten weten wat Encapsulation is.

De studenten begrijpen dat encapsulation zorgt voor het beschermen van code

De studenten kennen de verschillende scope modifiers (private , public, protected en internal)

De studenten begrijpen dat een getter en een setter extra controle geeft op data die de class inkomt en uitgaat

De studenten kunnen getters en setters maken.

De studenten weten wat een assembly is. (DLL)

De studenten kunnen in unity een assembly definieren

## Situatie:

De studenten hebben tot nog toe gebruik gemaakt van public en private en dan met name met het doel om variabelen wel of niet zichtbaar te maken in de inspector van Unity.

In die zin hebben ze wel voorkennis van private en public maar is het ook een risico dat ze denken het te begrijpen maar het eigenlijk niet zo is.

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 7: OOP Encapsulation

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:15 | Energizer: ?? |  |
| 00:15 – 00:20 | Oriëntatie vragen stellen over: Encapsulation  Wie weet wat encapsulation inhoud? |  |
| 00:20 – 00:40 | Uitleg over encapsulation | Miro bord |
| 00:40 – 01:40 | Uitvoeren opdracht getters en setters maken en protected variabelen en functies gebruiken | Opdracht bereikbaar via simulize en ook daar weer inleveren |
| 01:40 – 01:55 | 3 studenten laten hun uitwerking op de beamer zien en lichten dit toe. | Beamer, studenten laptops |
| 1:55 | Inpakken + alles ingeleverd op simulize? |  |

# Lesplan les 8: Class Diagram

## Doelen:

De studenten weten wat Class Diagram is.

De studenten begrijpen dat een class diagram kan zorgen dat je achteraf inzicht krijgt in de structuur van je code.

De studenten begrijpen dat een class diagram kan zorgen dat je vooraf beter na kunt denken over hoe je de structuur op kunt zetten.

De studenten begrijpen dat een class diagram onderdeel is van UML

De studenten begrijpen het belang van een geformaliseerde taal zodat je op een exacte manier de structuur kunt ontwerpen

De studenten weten dat er tools zijn om class diagrammen te modelleren en dat je zelfs een code base kunt genereren via UML.

De studenten weten hoe je **attribute** weergeeft

De studenten weten hoe je **operation** weergeeft

De studenten kennen de verschillende relatie vormen

De studenten weten hoe je verschillende relaties weergeeft

## Situatie:

De studenten nog helemaal geen ervaring met het aanbrengen van structuur in hun code. Het modelleren van hun structuur met UML is ook nieuw

Belangrijk is ook dat ze nog nooit met een modelling tool hebben gewerkt en dat deze ook niet altijd heel fijn werken. De studenten hebben hier dus extra hulp bij nodig.

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 8: Class Diagram

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:15 | Energizer: ?? |  |
| 00:15 – 00:20 | Oriëntatie vragen stellen over: Class Diagram  Wie weet wat een class diagram is? |  |
| 00:20 – 00:40 | Uitleg over class diagrams | <https://miro.com/app/board/uXjVNbo1Yzs=/> |
| 00:40 – 01:40 | Uitvoeren opdracht maken van een eigen class diagram op basis van hun TD project of een voorbeeldproject in *2 tallen maken* | Opdracht bereikbaar via simulize en ook daar weer inleveren |
| 01:40 – 01:55 | 3 studenten laten hun uitwerking op de beamer zien en lichten dit toe. | Beamer, studenten laptops |
| 1:55 | Inpakken + alles ingeleverd op simulize? |  |

# Lesplan les 9: OOP Polymorphism

## Doelen:

De studenten weten wat Polymorfisme is.

De studenten begrijpen dat polymorfisme zorgt dat code uitbreidbaar maar niet aanpasbaar is

De studenten weten wat een interface is

De studenten weten wat overriden is

De studenten begrijpen het verschil tussen een abstract class en een interface

De studenten kunnen een interface maken en classes implementeren die aan de eisen van de interface voldoen

## Situatie:

De studenten hebben nog helemaal geen kennis van interfaces en polimorfisme.

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 9: OOP Polymorphism

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:15 | Energizer: ?? |  |
| 00:15 – 00:20 | Oriëntatie vragen stellen over: Polymorphism  Wie weet wat polymorphism inhoud? |  |
| 00:20 – 00:40 | Uitleg over Polymorphism | Miro bord |
| 00:40 – 01:40 | Uitvoeren opdracht Interface maken en classes die deze implementeren | Opdracht bereikbaar via simulize en ook daar weer inleveren |
| 01:40 – 01:55 | 3 studenten laten hun uitwerking op de beamer zien en lichten dit toe. | Beamer, studenten laptops |
| 1:55 | Inpakken + alles ingeleverd op simulize? |  |

# Lesplan les 10: Design Patterns, Factory Method

## Doelen:

De studenten weten wat Design patterns zijn.

De studenten weten dat er heel veel design patterns zijn die allemaal specifieke struktuur problemen op kunnen lossen.

De studenten weten dat er meerdere design patterns geschikt zijn om toe te passen in een tower defense game en dat deze gecombineerd kunnen worden.

De studenten weten dat de factory method pattern ervoor zorgt dat je op een goede manier objecten in je game kan creeren zoals towers , bullets en enemies.

## Situatie:

De studenten zijn allemaal erg druk met hun eigen towerdefense games en lopen ook vast tegen problemen aan met het op meerdere manieren creëren van verschillende gameobjecten.

Het design pattern zou dus goed toepasbaar moeten zijn op hun eigen project.

## Toetsing:

d.m.v verschillende praktijkopdrachten in de les kunnen de studenten laten zien dat ze de onderdelen nog beheersen.

## Vervolgacties:

Studenten die kennis van onderdelen nog missen in kaart brengen.

Studenten die kennis van onderdelen nog missen uitleg en oefen materiaal meegeven.

Volgende les checken of oefeningen zijn gedaan en of de stof is begrepen.

# Lesplan les 10: Design Patterns, Factory Method

|  |  |  |
| --- | --- | --- |
| tijd | activiteit | materiaal |
| 00:05 – 00:15 | Energizer: ?? |  |
| 00:15 – 00:20 | Oriëntatie vragen stellen over: Polymorphism  Wie weet wat polymorphism inhoud? |  |
| 00:20 – 00:40 | Uitleg over Polymorphism | Tutorial en opdracht op simulize > [github](https://github.com/djsjollema/lessen-gamedevelopment/tree/main/M5/Prog/10_Design_Patterns_Factory_Method) |
| 00:40 – 01:40 | Uitvoeren opdracht Interface maken en classes die deze implementeren | Opdracht bereikbaar via simulize en ook daar weer inleveren |
| 01:40 – 01:55 | 3 studenten laten hun uitwerking op de beamer zien en lichten dit toe. | Beamer, studenten laptops |
| 1:55 | Inpakken + alles ingeleverd op simulize? |  |