The Learning Triangle

Using artificial intelligence to play a game

Our vision

- main idea: game which consists of a randomized world and some creatures

- no player, just an algorithm

The LearningTriangle-Team

Steven Kovacs

Marco Müller

Project Manager

Tool Specialist

Configuration Manager

Implementer

Designer

Test Designer / Tester

Graphic Artist

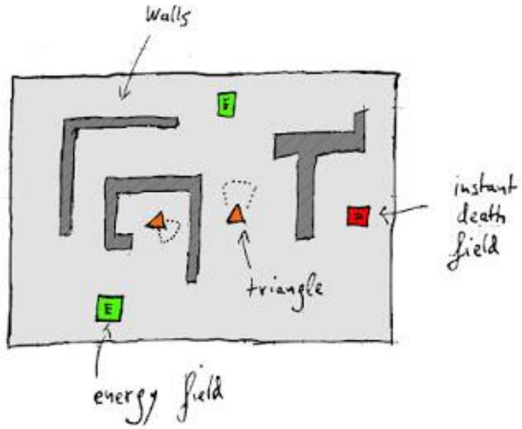
Business Designer

Requirements Specifier

→ We use RUP

Short view on the Game rules

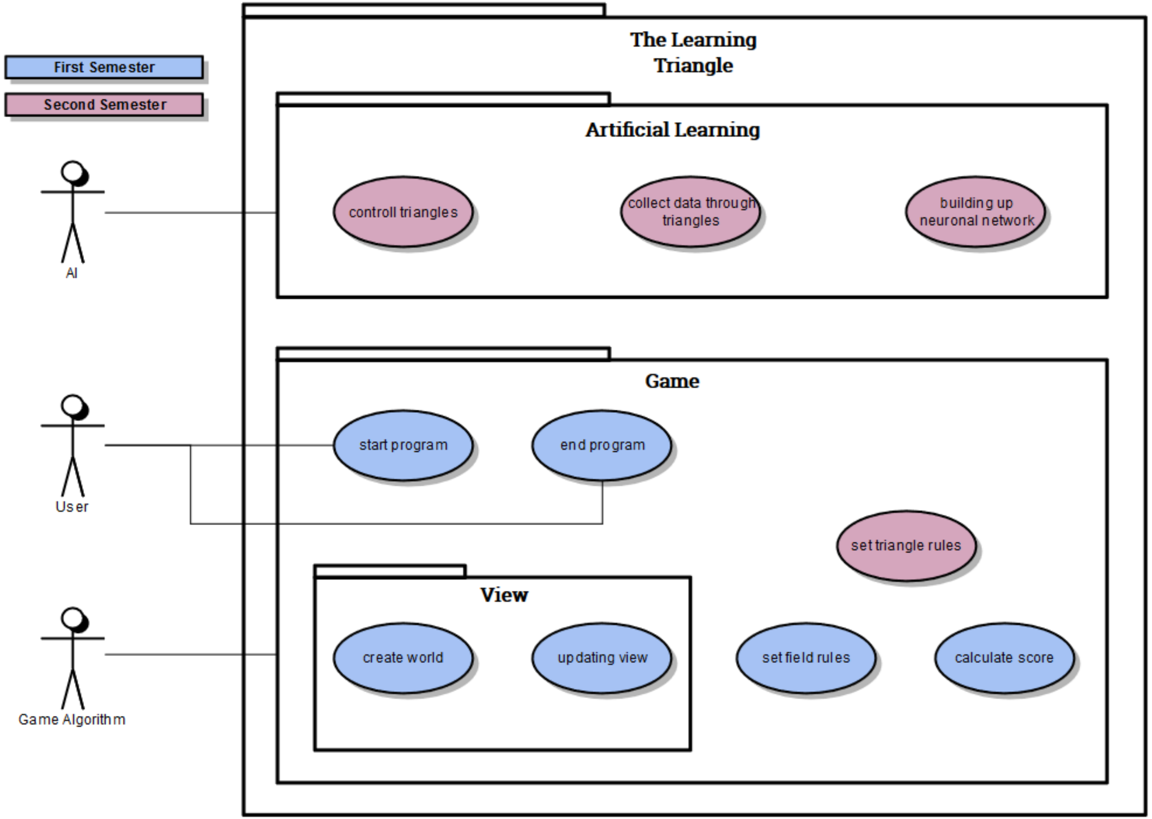
- Creatures are triangles and they want to survive
- each triangle has energy
- no energy means death
- special field types influence the triangle



- the AI has to manage the game and walk as much distance as possible

Defining SRS and our Use Cases

important things in a Software Architecture Document are the Scope and the Use Case Diagram



First Semester

Second Semester

The Learning Triangle

Artificial Learning

controll triangles

collect data through triangles

building up neuronal network

Game

start program

end program

set triangle rules

View

create world

updating view

set field rules

calculate score

AI

User

Game Algorithm

The way we develop

- We followed the idea of an *agile* project

→ you have to welcome a change while you are working on your project

- more contact with customer to follow his wishes

→ don’t develop over some years just to realize at the end that the customer wanted a program completely different

Things to do before programming

→ Project Management

Two main tools for us to be organized and follow the agile process:

Jira and **Github**

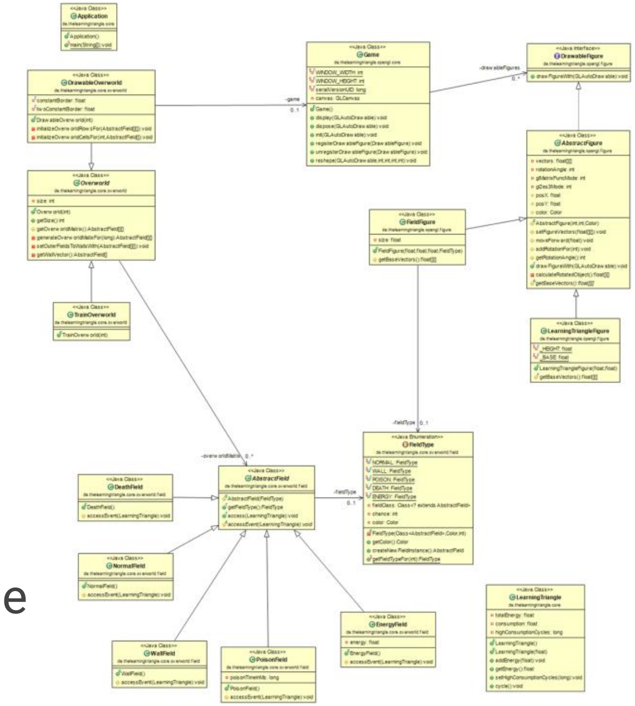
Our class diagram

We followed the MVC - Pattern, which means for us:

Model contains the data

Controller contains the algorithms

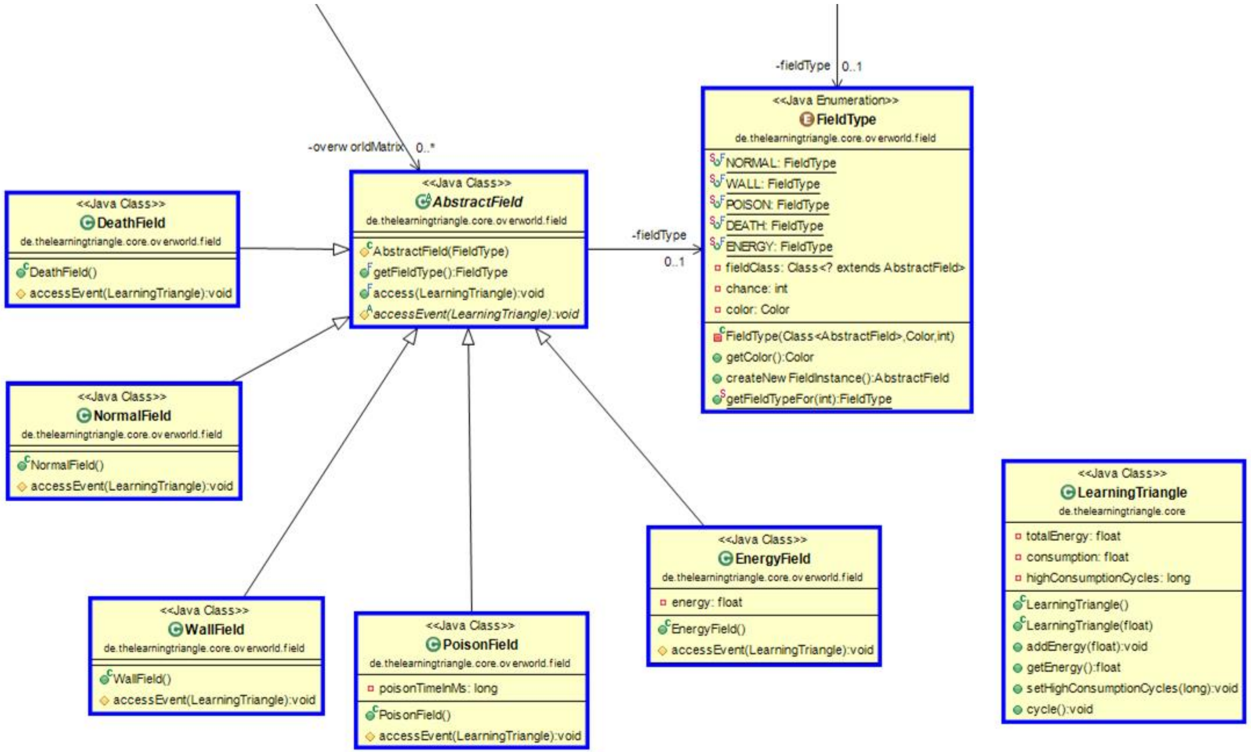
View contains the methods for showing the



<<Java Class>>
Application
de.thelearningtriangle.core
Application()
main(String[]):void
<<Java Class>>
DrawableOverworld
de.thelearningtriangle.core.overworld
constantBorder: float
twoConstantBorder: float
DrawableOverworld(int)
initializeOverworldRowsFor(AbstractField[][]):void
initializeOverworldCellsFor(int,AbstractField[]):void
-game
0..1
<<Java Class>>
Game
de.thelearningtriangle.opengl.core
WINDOW_WIDTH: int
WINDOW_HEIGHT: int
serialVersionUID: long
canvas: GLCanvas
Game()
display(GLAutoDrawable):void
dispose(GLAutoDrawable):void
init(GLAutoDrawable):void
registerDrawableFigure(DrawableFigure):void
unregisterDrawableFigure(DrawableFigure):void
reshape(GLAutoDrawable,int,int,int,int):void
-drawableFigures
0..*
<<Java Interface>>
DrawableFigure
de.thelearningtriangle.opengl.figure
drawFigureWith(GLAutoDrawable):void
<<Java Class>>
Overworld
de.thelearningtriangle.core.overworld
size: int
Overworld(int)
getSize():int
getOverworldMatrix():AbstractField[][]
generateOverworldMatrixFor(long):AbstractField[][]
setOuterFieldsToWallsWith(AbstractField[][]):void
getWallVector():AbstractField[]
<<Java Class>>
AbstractFigure
de.thelearningtriangle.opengl.figure
vectors: float[][]
rotationAngle: int
glMatrixFuncMode: int
glDes3Mode: int
posX: float
posY: float
color: Color
AbstractFigure(int,int,Color)
setFigureVectors(float[][]):void
moveForward(float):void
addRotationFor(int):void
getRotationAngle():int
drawFigureWith(GLAutoDrawable):void
calculateRotatedObject():float[][]
getBaseVectors():float[][]
<<Java Class>>
FieldFigure
de.thelearningtriangle.opengl.figure
size: float
FieldFigure(float,float,float,FieldType)
getBaseVectors():float[][]
<<Java Class>>
TrainOverworld
de.thelearningtriangle.core.overworld
TrainOverworld(int)
<<Java Class>>
LearningTriangleFigure
de.thelearningtriangle.opengl.figure
_HEIGHT: float
_BASE: float
LearningTriangleFigure(float,float)
getBaseVectors():float[][]
-fieldType
0..1
-overworldMatrix
0..*
<<Java Enumeration>>
FieldType
de.thelearningtriangle.core.overworld.field
NORMAL: FieldType
WALL: FieldType
POISON: FieldType
DEATH: FieldType
ENERGY: FieldType
fieldClass: Class<? extends AbstractField>
chance: int
color: Color
FieldType(Class<AbstractField>,Color,int)
getColor():Color
createNewFieldInstance():AbstractField
getFieldTypeFor(int):FieldType
<<Java Class>>
DeathField
de.thelearningtriangle.core.overworld.field
DeathField()
accessEvent(LearningTriangle):void
<<Java Class>>
AbstractField
de.thelearningtriangle.core.overworld.field
AbstractField(FieldType)
getFieldType():FieldType
access(LearningTriangle):void
accessEvent(LearningTriangle):void
-fieldType
0..1
<<Java Class>>
NormalField
de.thelearningtriangle.core.overworld.field
NormalField()
accessEvent(LearningTriangle):void
<<Java Class>>
LearningTriangle
de.thelearningtriangle.core
totalEnergy: float
consumption: float
highConsumptionCycles: long
LearningTriangle()
LearningTriangle(float)
addEnergy(float):void
getEnergy():float
setHighConsumptionCycles(long):void
cycle():void
<<Java Class>>
EnergyField
de.thelearningtriangle.core.overworld.field
energy: float
EnergyField()
accessEvent(LearningTriangle):void
<<Java Class>>
WallField
de.thelearningtriangle.core.overworld.field
WallField()
accessEvent(LearningTriangle):void
<<Java Class>>
PoisonField
de.thelearningtriangle.core.overworld.field
poisonTimeInMs: long
PoisonField()
accessEvent(LearningTriangle):void

We splitted the class diagram in these three parts

Model



-overw orldMatrix 0..*

<<Java Class>>
AbstractField
de.thelearningtriangle.core.ov erworld.field

AbstractField(FieldType)
getFieldType():FieldType
access(LearningTriangle):void
accessEvent(LearningTriangle):void

-fieldType 0..1

-fieldType 0..1

<<Java Enumeration>>
FieldType
de.thelearningtriangle.core.ov erworld.field

NORMAL: FieldType
WALL: FieldType
POISON: FieldType
DEATH: FieldType
ENERGY: FieldType
fieldClass: Class<? extends AbstractField>
chance: int
color: Color

FieldType(Class<AbstractField>,Color,int)
getColor():Color
createNew FieldInstance():AbstractField
getFieldTypeFor(int):FieldType

<<Java Class>>
DeathField
de.thelearningtriangle.core.ov erworld.field

DeathField()
accessEvent(LearningTriangle):void

<<Java Class>>
NormalField
de.thelearningtriangle.core.ov erworld.field

NormalField()
accessEvent(LearningTriangle):void

<<Java Class>>
WallField
de.thelearningtriangle.core.ov erworld.field

WallField()
accessEvent(LearningTriangle):void

<<Java Class>>
PoisonField
de.thelearningtriangle.core.ov erworld.field

poisonTimeInMs: long

PoisonField()
accessEvent(LearningTriangle):void

<<Java Class>>
EnergyField
de.thelearningtriangle.core.ov erworld.field

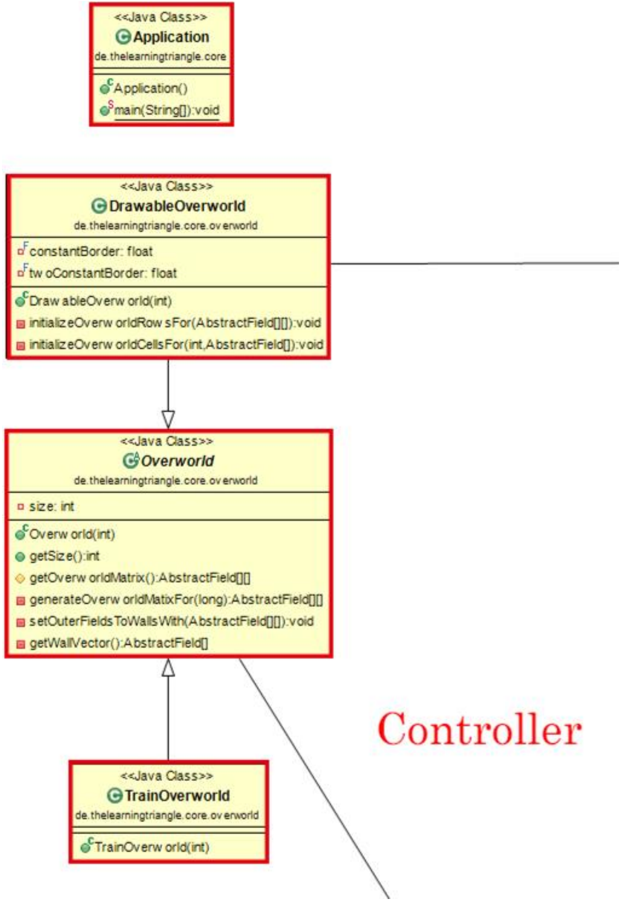
energy: float

EnergyField()
accessEvent(LearningTriangle):void

<<Java Class>>
LearningTriangle
de.thelearningtriangle.core

totalEnergy: float
consumption: float
highConsumptionCycles: long

LearningTriangle()
LearningTriangle(float)
addEnergy(float):void
getEnergy():float
setHighConsumptionCycles(long):void
cycle():void



<<Java Class>>
Application
de.thelearningtriangle.core

Application()
main(String[]):void

<<Java Class>>
DrawableOverworld
de.thelearningtriangle.core.overworld

constantBorder: float
twoConstantBorder: float

DrawableOverworld(int)
initializeOverworldRowsFor(AbstractField[][]):void
initializeOverworldCellsFor(int,AbstractField[]):void

<<Java Class>>
Overworld
de.thelearningtriangle.core.overworld

size: int

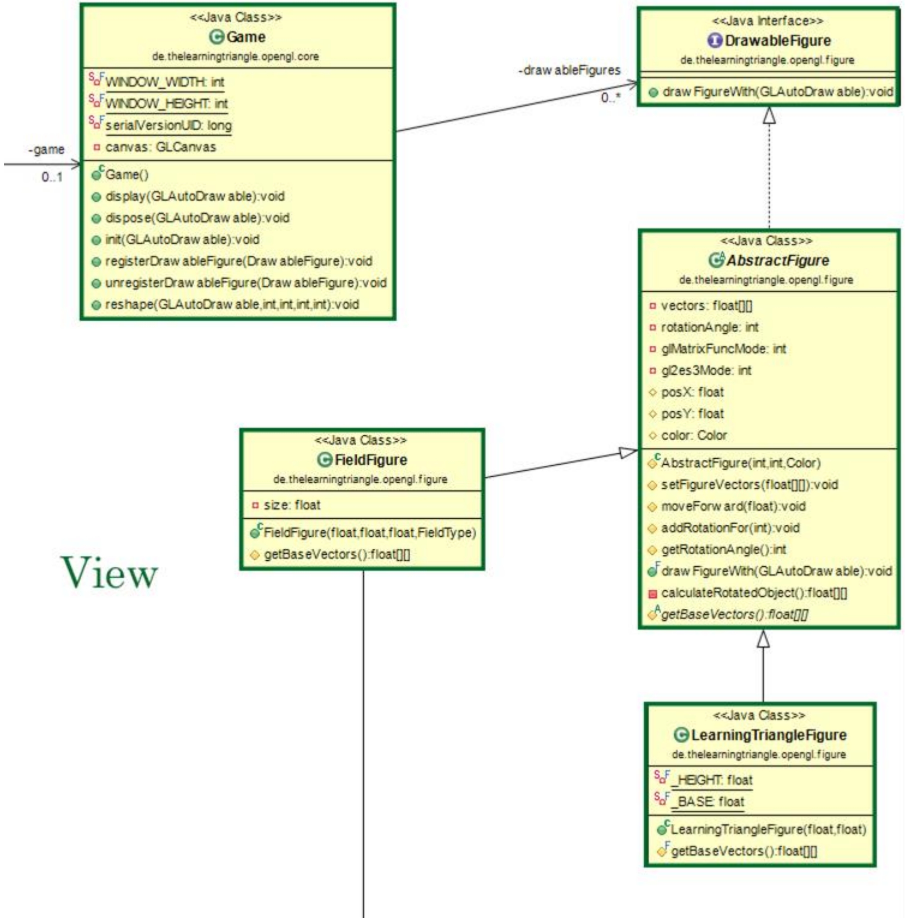
Overworld(int)
getSize():int
getOverworldMatrix():AbstractField[][]
generateOverworldMatixFor(long):AbstractField[][]
setOuterFieldsToWallsWith(AbstractField[][]):void
getWallVector():AbstractField[]

<<Java Class>>
TrainOverworld
de.thelearningtriangle.core.overworld

TrainOverworld(int)

Controller

View



<<Java Class>>
Game
de.thelearningtriangle.opengl.core

- WINDOW_WIDTH: int
- WINDOW_HEIGHT: int
- serialVersionUID: long
- canvas: GLCanvas

- Game()
- display(GLAutoDrawable):void
- dispose(GLAutoDrawable):void
- init(GLAutoDrawable):void
- registerDrawableFigure(DrawableFigure):void
- unregisterDrawableFigure(DrawableFigure):void
- reshape(GLAutoDrawable,int,int,int,int):void

<<Java Interface>>
DrawableFigure
de.thelearningtriangle.opengl.figure

- drawFigureWith(GLAutoDrawable):void

<<Java Class>>
AbstractFigure
de.thelearningtriangle.opengl.figure

- vectors: float[][]
- rotationAngle: int
- glMatrixFuncMode: int
- gl2es3Mode: int
- posX: float
- posY: float
- color: Color

- AbstractFigure(int,int,Color)
- setFigureVectors(float[][]):void
- moveForward(float):void
- addRotationFor(int):void
- getRotationAngle():int
- drawFigureWith(GLAutoDrawable):void
- calculateRotatedObject():float[][]
- *getBaseVectors():float[][]*

<<Java Class>>
FieldFigure
de.thelearningtriangle.opengl.figure

- size: float

- FieldFigure(float,float,float,FieldType)
- getBaseVectors():float[][]

<<Java Class>>
LearningTriangleFigure
de.thelearningtriangle.opengl.figure

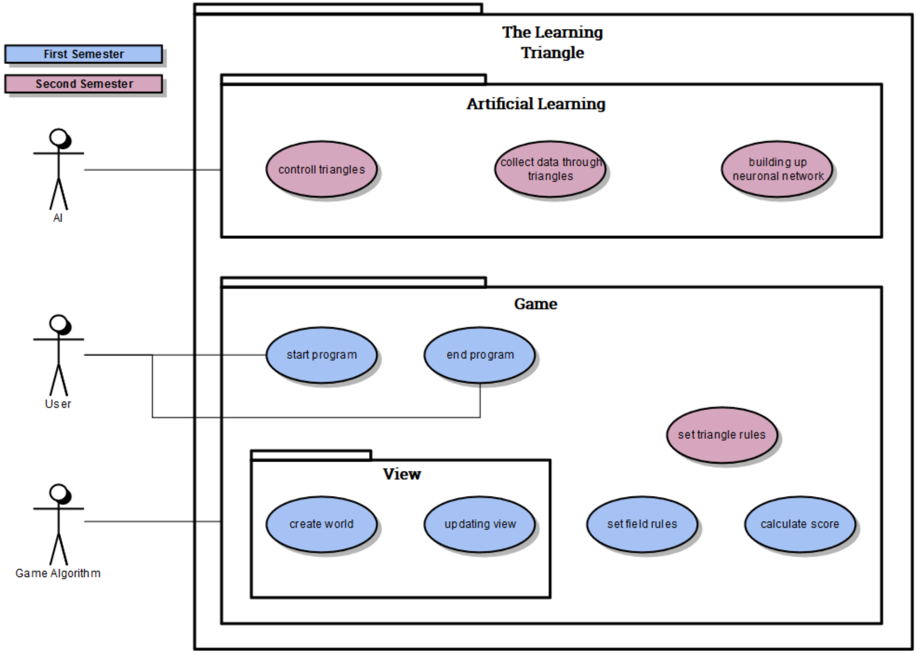
- _HEIGHT: float
- _BASE: float

- LearningTriangleFigure(float,float)
- getBaseVectors():float[][]

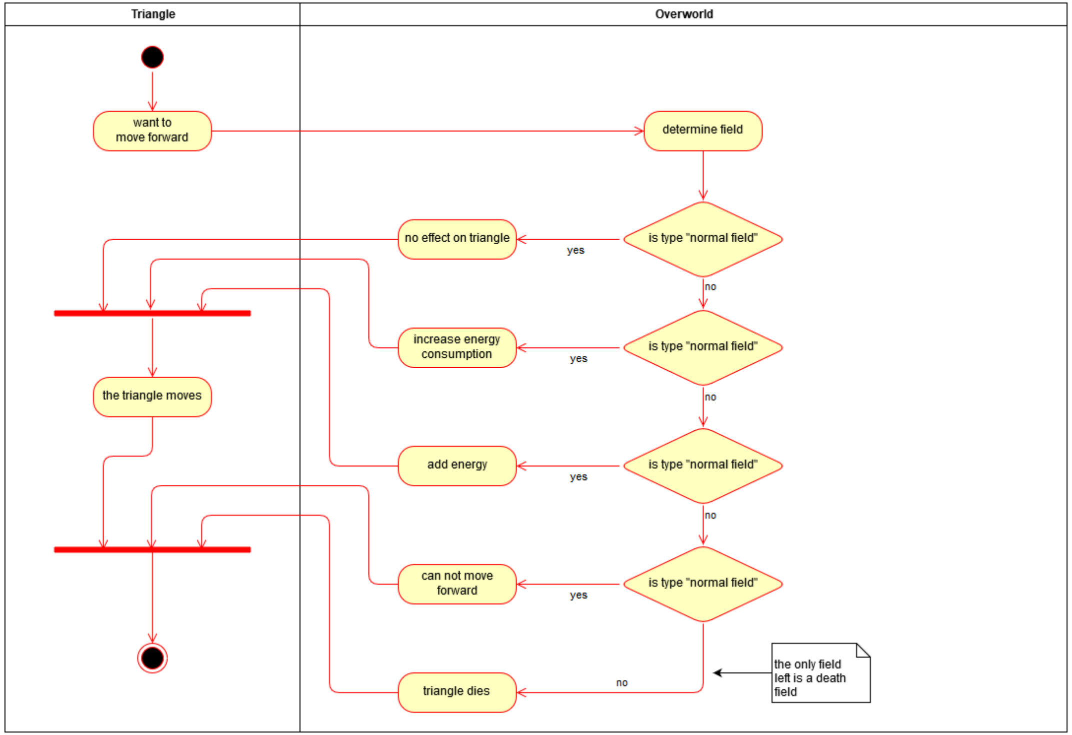
Our functionality

start/end game are just there to show the minimal user interaction

- create world (random)
- updating view
- ***set field rules***
- calculate score



Use Case - Set field rules



activity diagram

```
Feature: Game rules
             In order to set the game rules
As a triangle
I want to define the events for the different types of
fields in the overworld

Scenario: Normal Field
             Given I want to move in any direction
             When I would move on a normal field
             Then I move forward

Scenario: Poison Field
             Given I want to move in any direction
             When I would move on a poison field
             Then I move forward
             And my energy consumption becomes higher
```

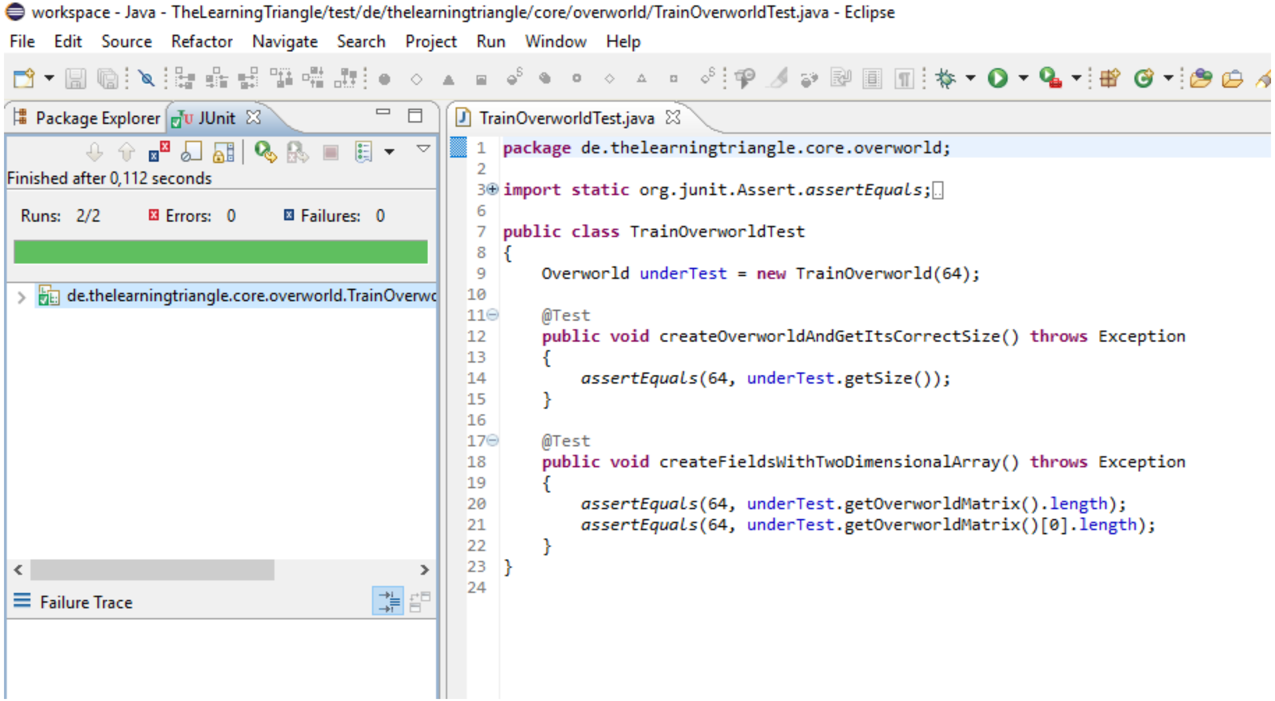
```
Scenario: Energy Field
             Given I want to move in any direction
             When I would move on an energy field
             Then I move forward
             And my energy becomes higher

Scenario: Wall Field
             Given I want to move in any direction
             When I would move on a poison field
             Then I don't move forward

Scenario: Death Field
             Given I want to move in any direction
             When I would move on a death field
             Then I don't move forward
             And I die
```

Testing the functionalities

- we wrote .feature files, but they aren’t our way to test

→ we use the Mocking-Framework Mockito

One example


workspace - Java - TheLearningTriangle/test/de/thelearningtriangle/core/overworld/TrainOverworldTest.java - Eclipse
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
Package Explorer
JUnit
TrainOverworldTest.java
Failure Trace

Finished after 0,112 seconds

```
Runs:  2/2        Errors:  0        Failures:  0
```

de.thelearningtriangle.core.overworld.TrainOverwc

```java
package de.thelearningtriangle.core.overworld;

import static org.junit.Assert.assertEquals;

public class TrainOverworldTest
{
    Overworld underTest = new TrainOverworld(64);

    @Test
    public void createOverworldAndGetItsCorrectSize() throws Exception
    {
        assertEquals(64, underTest.getSize());
    }

    @Test
    public void createFieldsWithTwoDimensionalArray() throws Exception
    {
        assertEquals(64, underTest.getOverworldMatrix().length);
        assertEquals(64, underTest.getOverworldMatrix()[0].length);
    }
}
```

Time spend

Finally, our demo of the program