Zyklus 1: Konzept

Inhalt

Fallen (5p) 2
Beschreibung der Aufgabe 2
Beschreibung der Lösung 2
Methoden und Techniken 2
Ansatz und Modellierung 2
Grundeigenschaften 3
UML 3
Beschreibung der Fallen 4
Monster (5p) 5
Beschreibung der Aufgabe 5
Beschreibung der Lösung 5
Methoden und Techniken 5
Ansatz und Modellierung: 5
Grundeigenschaften 7
UML 7
Beschreibung der Monster 8
Items (5p) 9
Beschreibung der Aufgabe 9
Beschreibung der Lösung 9
Methoden und Techniken 9
Ansatz und Modellierung 9
Beschreibung der Items 10

Fallen (5p)

Beschreibung der Aufgabe

- Fallen an zufälligen Positionen, pro Level (1-n Fallen)
- Auslösen der Fallen bei Kollision (Monster/Held/Item)
- Verschiedene Fallen Effekte (Schaden/Monster Versteck/Teleport)
- Sichtbare/unsichtbare Fallen
- Aktivierungsrate (1 bis n)
- Einige Fallen sollen per Schalter deaktivierbar sein (zufällige Positionen)
- Effekt vor Implementierung „unsichtbare Fallen => sichtbar“ als Zauber/Trank

Beschreibung der Lösung

Realisierung mehrerer Fallen Typen (Schaden/Monster Versteck/Teleport), die Zufällig im Dungeon Level verteilt sind, es soll Fallen geben die sichtbar sind und wiederum Fallen die nicht sichtbar sind, die sichtbaren und unsichtbaren Fallen sollen eine Animation erhalten (unsichtbare Fallen erst beim Auslösen).

Es soll Fallen mit und ohne Schalter geben, die Schalter werden wie die Fallen zufällig im Dungeon verteilt. Wird ein Schalter betätigt wird die entsprechende gebundene Falle ausgelöst und die dazugehörige auslöse Animation angezeigt.

 Die Teleport Fallen sollen nie einen auslöse Schalter haben, die Teleport Falle kann unendlich oft ausgelöst werden.

 Die Monster Falle wird einmalig ausgelöst und beinhaltet ein zufälliges Monster, das im späteren Verlauf (nach Implementierung von Angriffen) den Helden angreifen soll.

 Die einfache Falle, die nur Schaden verursacht soll einen zufälligen Wert n-m an Schaden an dem Helden oder an Monstern verursachen.

Methoden und Techniken

Der Code wird mit Javadoc ggf. mit normalen Kommentaren (private) dokumentiert.

Für die Realisierung der Fallen wird das Type Object Pattern verwendet => Übergeordnete Muster Klasse.

Ansatz und Modellierung

Es wird eine abstrakte Trap Klasse erstellt, von der die anderen Fallen Arten erben.
Die abstrakte Trap Klasse enthält einige Methoden und Attribute, die alle Fallen haben, die speziellen Eigenschaften jedes Fallen Typs werden in eigenen Klassen realisiert.

Alexey Khokhlov
Michel Witt
Ayaz Khudhur

Grundeigenschaften

Alle Fallen:

- PositionComponent position // Position der Falle
- Boolean visibility // sichtbar oder nicht sichtbar
- Animation worldAnimation // Darstellung im Level
- Boolean placedSwitch // Falle hat einen Schalter

Teleport:

- PositionComponent newPosition // Teleportier Position

Monster Falle:

- Monster monster // Monster das aus der Falle kommt
- Switch switch // dazugehöriger Schalter

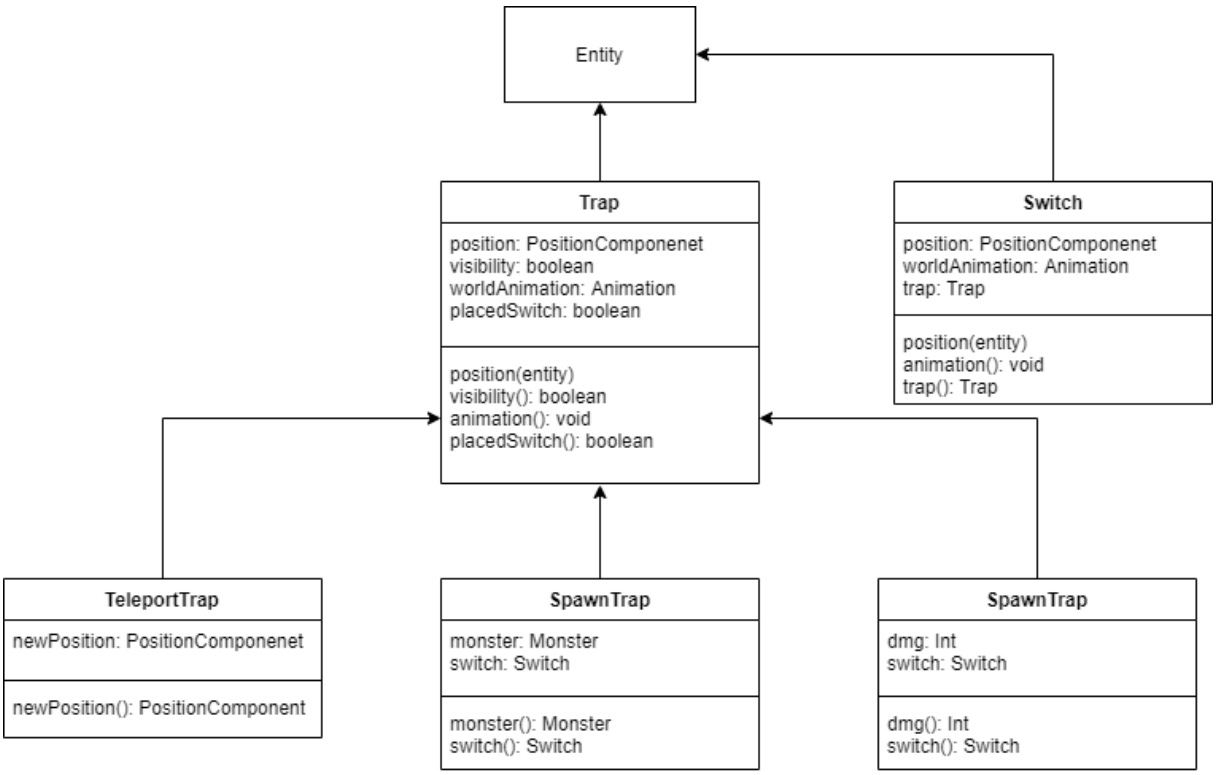
Einfach Falle:

- Int dmg // Schaden der verursacht wird, wenn die Falle ausgelöst wird
- Switch switch // dazugehöriger Schalter

Schalter:

- PositionComponent position // Position des Schalters
- Animation worldAnimation // Darstellung im Level

UML



Alexey Khokhlov
Michel Witt
Ayaz Khudhur

Beschreibung der Fallen

Teleport:
Bewegt den Spieler oder ein Monster an eine andere Position.

Monster Falle:
Ein zufälliges Monster erscheint aus der Falle, die Stärke des Monsters soll zum Helden Level Skalieren.

Einfach Falle:
Verursacht schaden am Spieler/Monster, dieser Schaden soll Prozentual abgezogen werden 10%, Komma Werte bei der Schadensermittlung werden aufgerundet.

Schalter:
Deaktiviert die zugewiesene Falle.

Alle Schalter und Fallen werden im Dungeon zufällig verteilt (zufällige Anzahl an Fallen).

Alexey Khokhlov
Michel Witt
Ayaz Khudhur

Monster (5p)

Beschreibung der Aufgabe

- Mindestens drei verschiedene Monster Typen
- Zufällige Anzahl an Monstern pro Level
- Bewegungs und Idle Animation
- Zufälliges laufen im Dungeon Level
- Wahrscheinlichkeit für mehr Monster pro Dungeon Level Fortschritt
- Stärkere Monster je höher der Dungeon Level Fortschritt

Beschreibung der Lösung

Realisierung mehrerer Monster Typen, die Zufällig im Dungeon Level verteilt werden und sich zufällig bewegen. Für die Bewegung der Monster wird das AI-System von der Aufgaben Beschreibung genutzt und um das Idle-AI System ergänzt.

Die Monster erhalten Bewegung und Idle Animationen, je nach Monster können die Frames jeder Animation variieren.

Die Monster sollen genauso wie der Held eine gewisse menge an Lebenspunkten und Schaden verursachen können => das eigentliche Angreifen wird später Implementiert „Fernkampf/Nahkampf“, die einzelnen Werte wie HP und Schaden werden jetzt vordefiniert, damit es im späterem „Angriffssystem“ verwendet werden kann.

Monster Typen:

 => Beißer

 => Zombie

 => Kleiner Drache

Methoden und Techniken

Der Code wird mit Javadoc ggf. mit normalen Kommentaren (private) dokumentiert.

Für die Realisierung der Monster wird das Type Object Pattern verwendet => Übergeordnete Muster Klasse.

Ansatz und Modellierung:

Es wird eine Monster Klasse erstellt, die an jedem Monster Typ im Spiel vererbt wird. Die Basis Monster Klasse erbt die Klasse Entity.

Grund:
Die einzelnen Monster Typen beinhalten unterschiedliche Animationen, es wäre möglich eine Monster Klasse zu programmieren und einen zufälligen Monster Animationssatz zuzuweisen, das oben genannte System empfinde ich besser, da ggf. bestimmte Monster so noch zusätzliche Eigenschaften besitzen können, die andere Monster nicht haben.

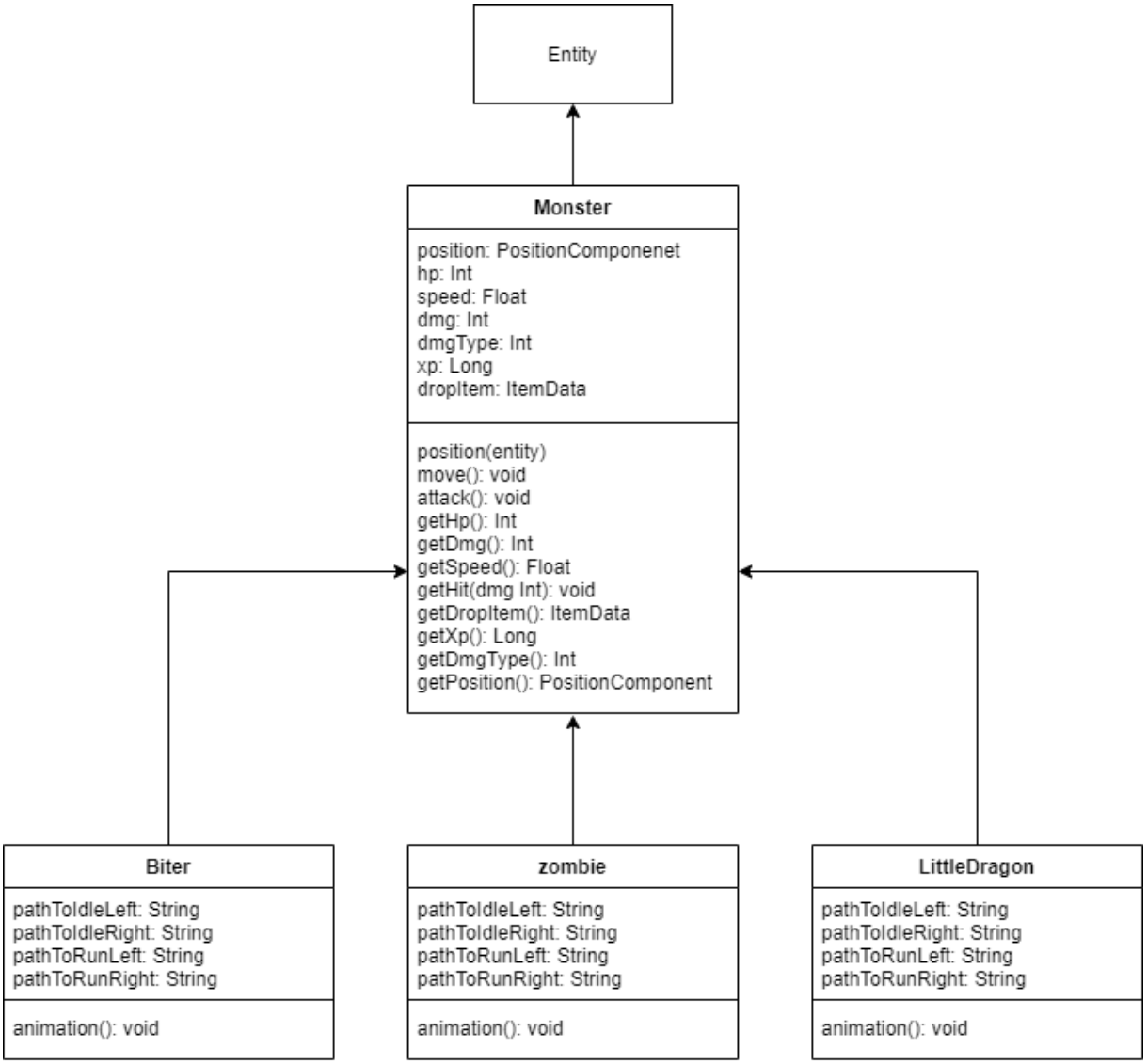
Alexey Khokhlov
Michel Witt
Ayaz Khudhur

Alexey Khokhlov
Michel Witt
Ayaz Khudhur

Grundeigenschaften

- PositionComponent position // Position des Monsters
- Int hp // Lebenspunkte eines Monsters
- Float speed // Bewegungsgeschwindigkeit eines Monsters
- Int dmg // Schaden den das monster verursacht
- Int dmgType // Fern- oder Nahkampf
- long xp // Erfahrungspunkte die ein Monster „abgibt“
- ItemData dropItem // Zufälliges Item das gedropt wird
- String pathToIdleLeft // Idle Animation links gedreht
- String pathToIdleRight // Idle Animation rechts gedreht
- String pathToRunLeft // Lauf Animation links
- String pathToRunRight // Lauf Animation rechts

UML



Alexey Khokhlov
Michel Witt
Ayaz Khudhur

Beschreibung der Monster

Beißer:

- 15 Hp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht
- 2 Sp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht
- Geschwindigkeit von 0,1f
- 10xp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht

Zombie:

- 25 Hp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht
- 4 Sp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht
- Geschwindigkeit von 0,25f
- 20xp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht

Kleiner Drache:

- 40 Hp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht
- 7 Sp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht
- Geschwindigkeit von 0,2f
- 30xp*1,0 // pro Dungeon Level wird der Multiplikator um 0,1 erhöht

Alle Monster werden im Dungeon zufällig verteilt (zufällige Anzahl an Monster).

Alexey Khokhlov
Michel Witt
Ayaz Khudhur

Items (5p)

Beschreibung der Aufgabe

- Mindestens drei verschiedene Typen von Gegenständen
- Jeder Gegenstand soll verwendbar sein (Funktionen selbst definieren)
- Jeder Gegenstand soll grafisch dargestellt werden
- Taschen mit begrenzter Anzahl an Slots
- Taschen können mehrere verschiedene Gegenstände aufbewahren

Beschreibung der Lösung

Realisierung mehrerer Item Typen:

-  Tränke zur Heilung
-  Waffen mit bestimmten Schadenswerten
-  Taschen in dem Items aufbewahrt werden können

Eine Tasche, Waffe und einige Tränke sollen dem Helden zum spiel Start zur Verfügung stehen (Grundausrüstung).

Items die sich im Inventar befinden haben ein nicht animiertes Item Bild => Inventar Icon, Items die im Dungeon liegen z.B. Tränke haben eine „blubber“ Animation.

Waffen die Ausgerüstet sind sollen am Helden dargestellt werden, ggf. auch mit Animation.

Methoden und Techniken

Der Code wird mit Javadoc ggf. mit normalen Kommentaren (private) dokumentiert.

Ansatz und Modellierung

Bzw. Beschreibung der vorhandenen Item Klassen:
- ItemDataGenerator.java:
Hier werden die einzelnen Items definiert bzw. in einer Liste gesammelt mit Titel, Beschreibung, Inventar Icon, Welt Textur (World Loot), einsammeln des Items, Fallen lassen des Items, benutzen des Items und Schadensskalierung.

- ItemData.java:
Das ist die Item Klasse, die ein Item „definiert“. Die Klasse ItemDataGenerator erstellt eine Liste aus Items die in der Klasse ItemData erstellt wurden (Attribute siehe ItemDataGenerator.java). Für das „Erhalten“, „Fallen lassen“ und „benutzen“ werden die Interfaces IOnCollect, IOnDrop und IOnUse benutzt.

- WorldItemBuilder.java:
Erstellt das Item als Entity, damit es im Dungeon verwendet und angezeigt werden kann.

Beschreibung der Klasse InventoryComponent:

Alexey Khokhlov
Michel Witt
Ayaz Khudhur

Diese Klasse erstellt ein Inventar mit einer fest angelegt Anzahl an Slots (Übergabeparameter). Die Klasse enthält die Methoden addItem und removeItem mit dem Items dem Inventar hinzugefügt und wieder entfernt werden können.

Die Methoden filledSlots() und getMaxSize(), werden im weiteren spielverlauf erstmal nicht benötigt (), da es die Methode emptySlots() gibt, das alleine schon ausreicht um festzustellen ob ein Item aufgesammelt werden kann oder nicht.

Die Methode getItems() gibt eine Liste aller Items zurück.

Beschreibung der Items
Tränke:
Stellt 10-20% Hp wieder her.

Waffen:

- Zufällige Sp von 5 bis 10 multipliziert mit 1,0 (+0,1 Multiplikator pro Dungeon Level)

Taschen:

- Zufälliger Inventarplatz 10 bis 20

Alle Items werden im Dungeon zufällig verteilt (zufällige Anzahl an Items), können aber auch von Monstern gedroppt werden.

Alexey Khokhlov
Michel Witt
Ayaz Khudhur