Augmented Reality Characters (ARCs)

Finale Präsentation

Kaveh Yousefi

Projekt - Idee

- Erstellung einer *Augmented-Reality*-Anwendung.
- Ein Spiel unter Verwendung von
  - NyARToolkit
  - Java 3D
  - Java Media Framework (JMF)
  - Webcam.

Projekt - Spielprinzip (1/2)

- Einsatz des NyARToolkit in Kombination mit Java 3D.
- Verschiedene Marker kennzeichnen verschiedene Charaktere.
- Charaktere sind „Kämpfer“:
  - Können angreifen
  - und sich verteiden.
- Jeder der zwei Spieler erhält zufällig einen Satz Marker-Karten, damit Charaktere.
- Ziel: Bezwingen aller gegnerischen Charaktere.

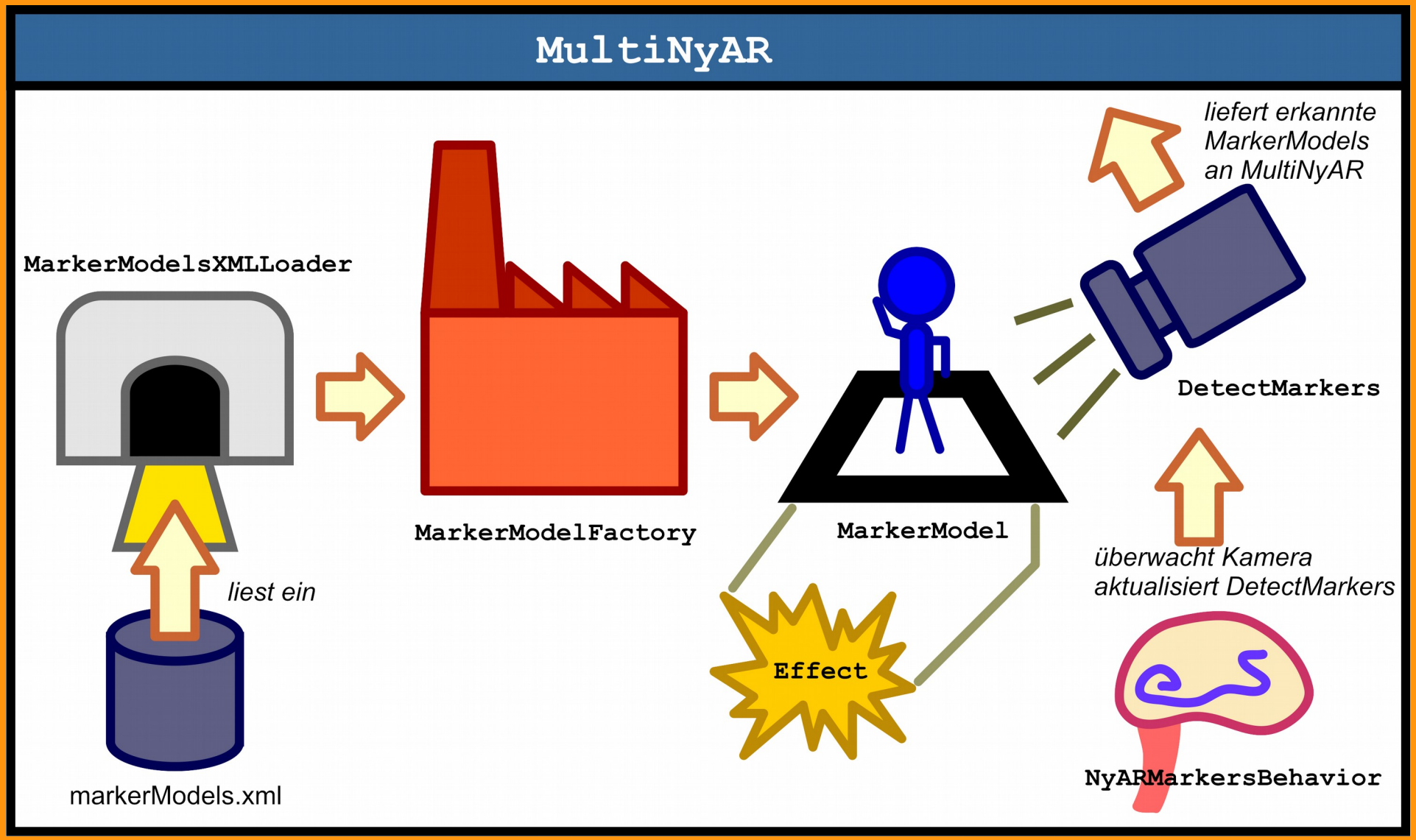
Projekt - Spielprinzip (2/2)

- Jede Spielfigur besitzt ein Grenzvolumen um ihren Marker.
- Jeder Spiel besitzt ein Verteidigungsfeld.
  - spezieller Marker mit Grenzvolumen.
- Angriff: Spieler bewegt Grenzvolumen einer seiner Figuren in Grenzvolumen einer gegnerischen Figur.
- Verteidigungsmodus: Spieler lässt eine Figur mit dem eigenen Defensiv-Marker kollidieren.

Projekt - Basis

- Die Umsetzung basiert auf einem Projekt von Dr. Andrew Davison.
  - Zugreifbar unter http://fivedots.coe.psu.ac.th/~ad/jg/ch165/.
- Infrastruktur wurde übernommen und weiterentwickelt.
  - Dies spiegelt sich in vielen Klassennamen wieder.

Projekt - Architektur



Informelle Darstellung der bedeutsamsten Klassen.

Marker

- Physische Marker müssen der *NyARToolkit*-Anwendung bekannt gemacht werden.
- Dies geschieht über eine Datei, welche eine digitale Abbildung der Marker-Grafik enthält.
  - Eine solche Datei kann man beispielsweise unter http://flash.tarotaro.org/blog/2009/07/12/mgo2/ erstellen.
- Die Erkennung der physischen Marker über die Kamera bewerkstelligt die *NyARToolkit*-Klasse

`jp.nyatla.nyartoolkit.detector.NyARDetectMarker`.

Marker-Kamera-Interaktion

- Die Klasse `adtest01.NyMarkersBehavior` lauscht auf Änderungen an der Kamera.
  - Sie zeichnet das von der Kamera aufgenommene Bild auf eine Hintergrundfläche (`Background`-Knoten).
- Die Aktualisierung des internen Zustandes (vor allem das Zeichnen des Kamerabildes) erfolgt in einem festgelegten Intervall.
- Die Marker-Erkennung wird durch Aktualisierung der `DetectMarkers`-Instanz angestoßen.

MarkerModel

- Zentrale Entität.
- Mit Marker verbundene 3D-Teilszene.
  - Repräsentiert alles auf einem Marker darstellbare.
- Beinhaltet unter anderem:
  - Pfad der Marker-Muster-Datei.
  - Geometrien innerhalb einer `TransformHierarchy`.
  - Daten über die repräsentierte Spielfigur.
  - Daten über den repräsentierten Effekt.
- Enthält Methoden zum Ändern der Position und des Zustandes.

MarkerModel-Datenhaltung

- Die Datenhaltung der Marker-Objekte (*MarkerModels*) erfolgt semi-automatisch über eine XML-Datei *markerModels.xml*.
  - Bietet bessere Erweiterbarkeit.
- Die Klasse `adtest01.MarkerModelsXMLLoader` ist zuständig für das Einlesen der XML-Datei.
  - Erzeugt eine Abbildung von *MarkerModel*-ID auf *MarkerModel*-Objekt.

MarkerModel-Bereitstellung (1/2)

- Die aus der XML-Datei erworbenen Modelle benötigen weitere Daten, z.B die Instanzen der eingesetzten Effekte.
  - Eine weitere Abstraktionsebene wird angeboten.
- Die Klasse `adtest01.MarkerModelFactory` kombiniert
  - die eingelesenen *MarkerModels*
  - mit den Effekt-Klassen
  - und eigens geschaffenen Geometrien (z.B. Rotationskörper, der ein Raumschiff repräsentiert).

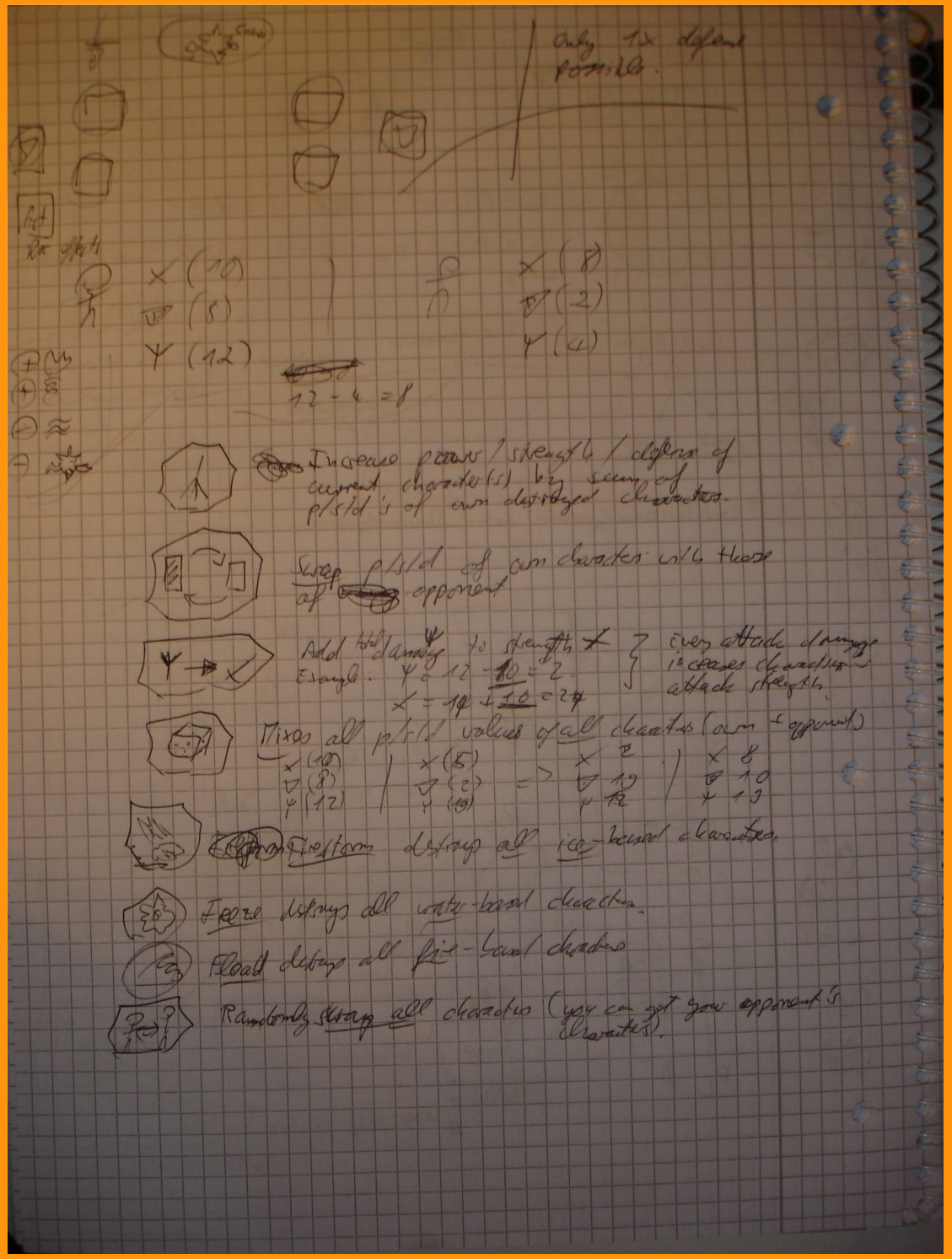
MarkerModel-Bereitstellung (2/2)

- Die *MarkerModels*, vorbereitet mitsamt Effekten und Geometrien, sind nun über die Methode `getMarkerModelByName(…)` zugreifbar, indem die *MarkerModel*-ID spezifiziert wird.
  - Einfachere Verwaltung der *MarkerModels*.
  - Trennung von Dateieinspeisung durch XML-Leser und Modell-Generierung durch „Modell-Fabrik“.
    - Modelle könnten beispielsweise aus einer Datenbank oder direkt (ohne Persistierung) angelegt werden, ohne dem Klienten der Klasse dies bewusst zu machen.

MarkerModel-Einbindung

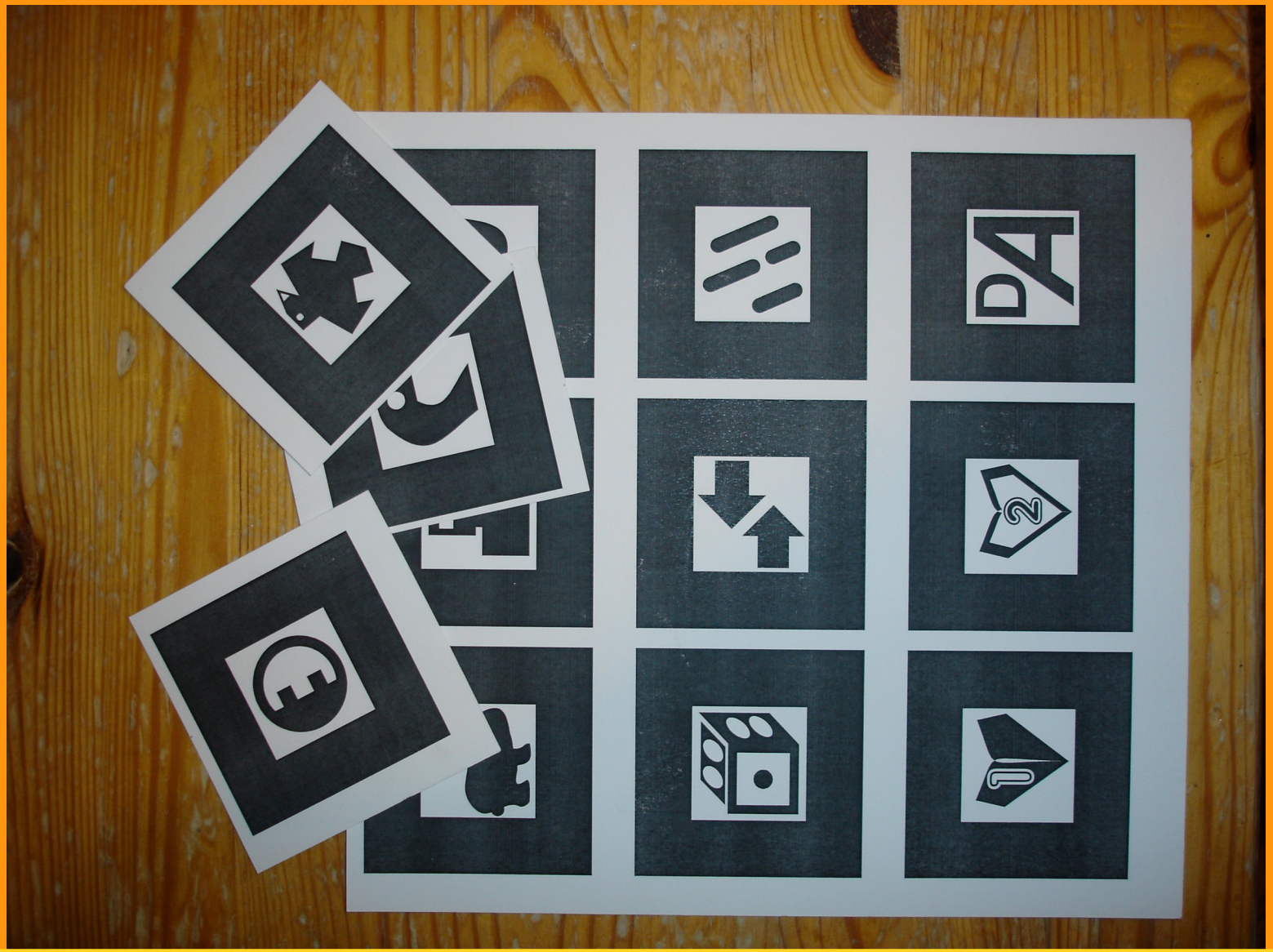
- Das Registrieren der *MarkerModels* mitsamt ihrer Geometrien erfolgt in der Hauptklasse `adtest01.MultiNyAR`, genauer in ihrer Methode `createSceneGraph()`.
- Diese entnimmt die fertigen *MarkerModel* über `MarkerModelFactory.getMarkerModelByName(…)`, indem sie als Parameter die *MarkerModel*-ID laut XML-Datei ausweist.

Entwicklung (1/6)



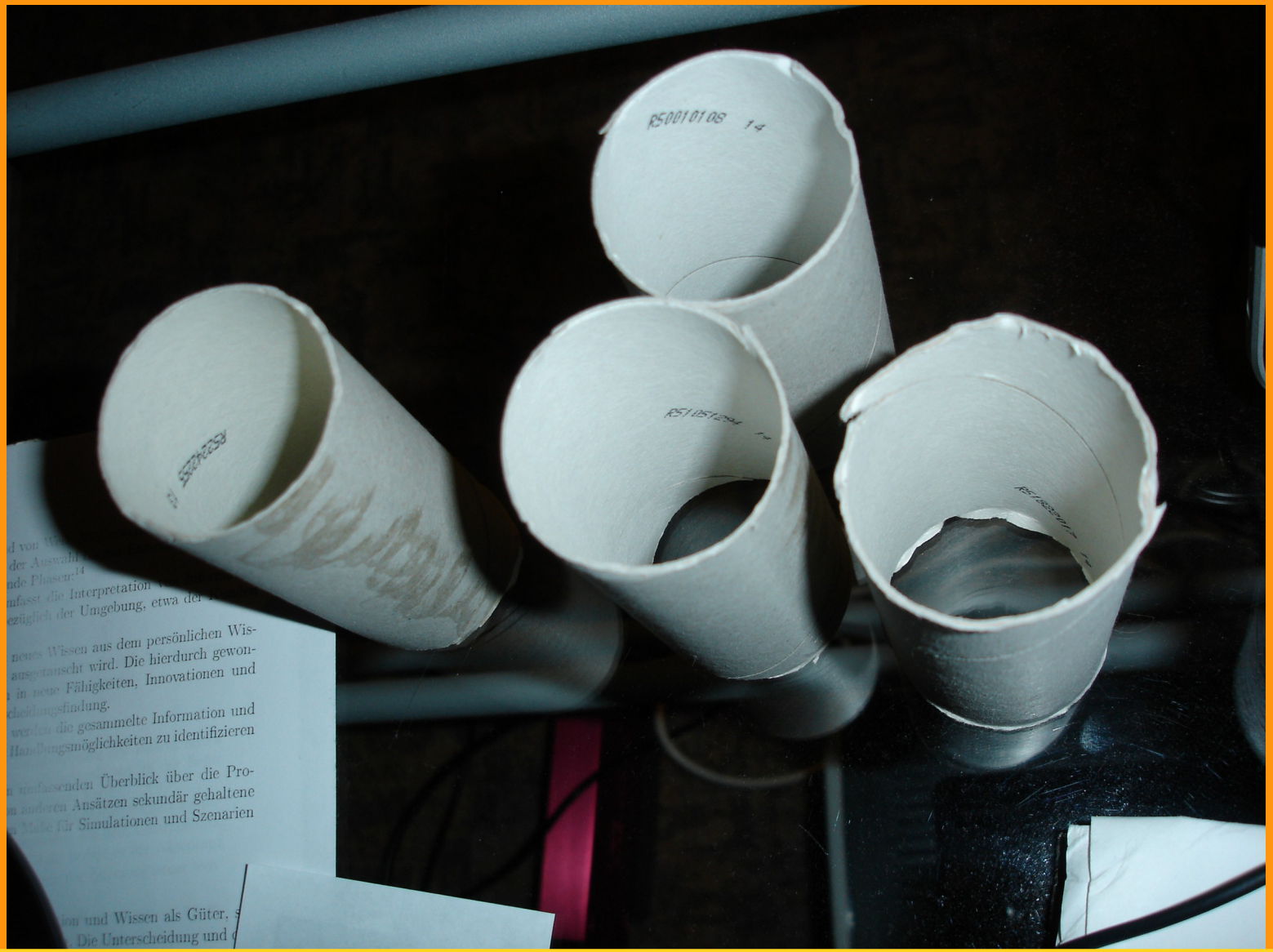
Frühe Notizen zu möglichen Charaktereigenschaften und Effekten im Spiel.

Entwicklung (2/6)



Ausgedruckte und teilweise ausgeschnittene Marker.

Entwicklung (3/6)



Papierrollen wurden durch Zerteilung in der Mitte in Sockel umfunktioniert.

Entwicklung (4/6)

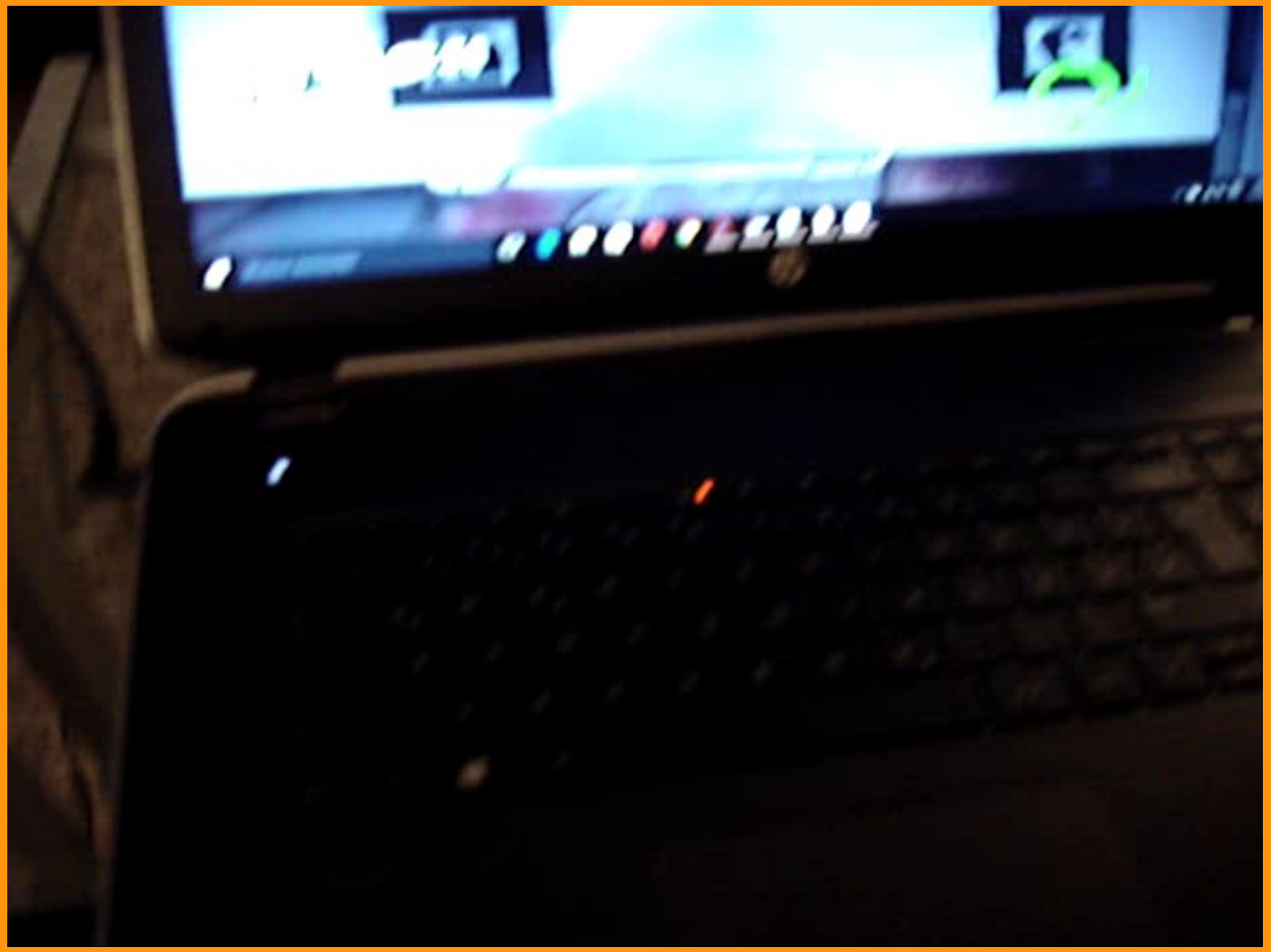
Marker auf ihrem Sockel.
Lediglich Spielfiguren und Defensiv-Marker benötigen einen solchen; Effekt-Marker, da unabhängig von Kollisionen, sind als Blättchen gehalten.

Entwicklung (5/6)

Eine ausgemusterte Tischlampe wurde zur Halterung für die Kamera umfunktioniert.

Bild links: Gesamtansicht der Tischlampe.
Bild unten: Detailansicht der Kamerabefestigung.

Entwicklung (6/6) - Video



Erweiterung – Effekte (1/3)

- Je Effekt wird eine Java-Klasse benötigt.
- Diese befindet sich im Paket `ky.gamelogic`.
- Vorgehen:
  1) Implementiere das *Interface* `Effect`.
  2) Erzeuge für den Effekt einen neuen Eintrag in der Datei *myresources\markerModels.xml*.

```
<model key="MARKER-MODEL-ID"
       name="MARKER-MODEL-NAME">
  <marker-file path="Data/MARKER-PATTERN-FILE"                    />
  <graphics type="custom-3d-model">
    <custom-model name=""                                         />
  </graphics>
  <marker-type type="effect"                                      />
  <element     type="EFFECT-ELEMENT"                              />
  <effect      name="EFFECT-ID"                                   />
  <collision   allowed="false"                                    />
</model>
```

Erweiterung – Effekte (2/3)

- Registriere den Effekt in der Klasse `adtest01.MarkerModelFactory` in der Methode `createEffectFactory()`.

```
private EffectFactory createEffectFactory
(
  MultiNyAR     multiNyAR,
  DetectMarkers detectMarkers
)
{
  EffectFactory effectFactory = null;

  effectFactory = new EffectFactory ();
  effectFactory.addEffect ("characterSwapper", new CharacterSwapper (multiNyAR, detectMarkers));
  effectFactory.addEffect ("rain",             new RainEffect       (multiNyAR, detectMarkers, new TransformHierarchy ()));
  effectFactory.addEffect ("statisticsMixer",  new StatisticsMixer  (multiNyAR, detectMarkers));
  effectFactory.addEffect ("defenseForAttack", new DefenseForAttack (multiNyAR, detectMarkers));
  effectFactory.addEffect ("resurrection",     new Resurrection     (multiNyAR, detectMarkers));
  effectFactory.addEffect ("undefend",         new Undefend         (multiNyAR, detectMarkers));

  effectFactory.addEffect ("EFFECT-ID",        new OwnEffect        (multiNyAR, detectMarkers));

  return effectFactory;
}
```

Erweiterung – Effekte (3/3)

- Registriere den Marker (MarkerModel) in der Hauptklasse `adtest01.MultiNyAR` innerhalb deren Methode `createSceneGraph()`.

```
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("bunny",     Player.SEARCHING_PLAYER), sceneBG, detectM
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("starship", Player.SEARCHING_PLAYER), sceneBG, detectM
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("bird",     Player.SEARCHING_PLAYER), sceneBG, detectM
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("snake",    Player.SEARCHING_PLAYER), sceneBG, detectM
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("boy",      Player.SEARCHING_PLAYER), sceneBG, detectM
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("robot",    Player.SEARCHING_PLAYER), sceneBG, detectM

addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("defenseMarker1", Player.PLAYER_1), sceneBG, detectMar
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("defenseMarker2", Player.PLAYER_2), sceneBG, detectMar

addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("characterSwapper", Player.NO_PLAYER), sceneBG, detect
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("defenseForAttack", Player.NO_PLAYER), sceneBG, detect
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("rain",            Player.NO_PLAYER), sceneBG, detect
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("statisticsMixer",  Player.NO_PLAYER), sceneBG, detect
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("resurrection",     Player.NO_PLAYER), sceneBG, detect
addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("undefend",         Player.NO_PLAYER), sceneBG, detect


addAndRegisterMarkerModel (markerObjectFactory.getMarkerModelByName ("MARKER-MODEL-ID",  Player.NO_PLAYER), sceneBG, detect
```

Ergebnis

- Das Spiel ist stark eingeschränkt spielbar:
  - Latenz
    - Verhindert flüssige Abbildung von Bewegungen auf den Bildschirm.
  - Abhängigkeit von Lichtverhältnissen
    - Bspw. werden oftmals trotz günstiger Beleuchtung Marker in der Dämmerung nicht ohne Zusatzlicht erkannt.
  - Mangelhafte Erkennung der Marker
    - Diese verschwinden unter Umständen oder werden irrtümlich identifiziert (recht chaotisches Verhalten).
- Das als größtes identifizierte Risiko hat sich bewahrheitet.
- Fazit: Das NyARToolkit ist für die Spieleentwicklung ungeeignet.

Eigene Beurteilung der Technologie

Kriterium	Bewertung	Begründung
A) Anzahl Freiheitsgrade	10	6 Freiheitsgrad verfügbar.
B) Anzahl verfolgter Körper	10	Beinahe beliebig viele Objekte gleichzeitig erkennbar.
C) Größe überwachter Fläche	5	Abhängig von Kamera.
D) Genauigkeit	2	Abhängig von Kamera, Verwechslungsgefahr bei Markern.
E) Wiederholrate	5	Möglicherweise ein Grund für die Verzögerung(?).
F) Latenz	1	Starke Verzögerungen, kaum flüssige Bewegungen.
G) Drift	10	Keine Akkumulation von Fehlern in der Transformationsmatrix beobachtet.
H) Empfindlichkeit	1	Abhängig von Beleuchtung, Kamerawinkel, Sichtbarkeit der Marker.
I) Kalibrierung	2	Schwieriger Abgleich von realer Welt zu virtuellem Koordinatensystem.
J) Usability	2	Anspruchsvolle Handhabung der Marker, vorsichtige Bewegungen notwendig.

Persönliche Beurteilung der Marker-Technologie anhand des NyARToolkits.
Skala: 1 bis 10, mit 1 = schlecht, 10 = hervorragend.
Quelle: [dorner2014virtual], S. 100-104

Frage

- Zentrale Frage:
  - Reichen alltägliche Technologien zur Erreichung einfacher *Augmented Reality*?
- Antwort:
  - Ja, *Augmented Reality* ist mit geringem Hardware- und Softwareeinsatz realisierbar.
  - Jedoch abhängig von Technologie.
  - Einstiegshürden und Machbarkeitsgrenzen werden in Zukunft wahrscheinlich sinken, da „alltäglich“ weiter leistungsfähige Geräte umschließen wird (z.B. *Xbox Kinect*).

Empfehlungen

- Transformation beachten:
  - Ist eigentümlich.
  - Nur Ebene mit Z-Koordinate > 0 scheint sichtbar zu sein.
- NyARToolkit für präzise Anwendungen vermeiden.

Grobplanung

- Vollzieht sich in vier großen Phasen:
  1) Visuell-mathematische Einarbeitung:
     - Studium der Transformationsprinzipien
  2) Kamera-Marker-Interaktion:
     - Kalibrieren der Kamera
     - Test multipler Marker
  3) Spiellogik:
     - Implementierung der Spiellogik
  4) Grafik:
     - Gestaltung der Spielcharaktere

Meilensteine

Nr.	Kennzeichen	Datum	Beschreibung
1	MS-1	09.06.2015	Erster Prototyp: Erkennung eigener Marker
2	MS-2	23.06.2015	Zweiter Prototyp: multiple Marker mit individuellen Figuren
3	MS-3	07.07.2015	Dritter Prototyp: Kollisionserkennung anhand Grenzvolumina
4	MS-4-F	Anfang WiSe	Abgabe des Projekts

Zeitplanung - Übersicht

Woche	Planung	Realität (Problem ⇒ Verzögerung)
18.05.2015 – 24.05.2015	Einbindung verschiedener Modelle Studium der Transformationsprinzipien	Feinheiten bei der Transformation (z-Koordinate, x-Achse ist invertiert) wurden bei der Einarbeitung mangels Verwendungszweck übersehen.
25.05.2015 – 31.05.2015	Erkennung eigener Marken Anlegung einer Codeverwaltung	
01.06.2015 – 07.06.2015	Erkennung eigener Marker Tests multipler Marker	Die geringe Anzahl an Test-Markern schuf einen anfänglich sicheren, jedoch später als trügerisch identifizierten Eindruck.
08.06.2015 – 14.06.2015	Tests multipler Marker Erstellung verschiedener 3D-Objekte	Mit steigender Anzahl an Markern wurde die Erkennung schwierig.
15.06.2015 – 21.06.2015	Interpretation von Gesten als Angriff/Verteidigung	Auf Grund begrenzten Raumes musste das Prinzip eines Verteidigungsfeldes in einen beweglichen Marker umgewandelt werden
22.06.2015 – 28.06.2015	Kollisionserkennung	Die erkannte Distanz zwischen den Markern ist abhängig von der Kameraposition.
29.06.2015 – 30.08.2015	Implementierung der Spiellogik	Schwierigkeiten in der Kollisionserkennung wirkten sich nachteilig auf die davon abhängige Logik aus.
31.08.2015 – 06.09.2015	Generierung der Spielfiguren	Eigentümliches virtuelles Koordinatensystem erschwerte die Spielfigurenerzeugung.
07.09.2015 - Abgabe	Testphase	Probleme bei Erkennung von Markern und hierdurch notwendige Revisionen erschwerten die Testphase.

Zeitplanung - Ergebnis

- Obgleich zunächst oberflächlich als realistisch betrachtet, verursachten „Kleinigkeiten“ in den späteren Phasen Probleme.
- Die Skalierbarkeit der Anwendung, d.h., die Zunahme von Markern schuf ungeahnte Schwierigkeiten:
  - Marker-Verwechslung
  - Platzmangel für Interaktion.
- Allerdings ist fraglich, wie man diese Probleme hätte früher erkennen oder vermeiden können.

Quellenverzeichnis

[dorner2014virtual] Ralf Dörner, Wolfgang Broll, Paul Grimm, Bernhard Jung, Virtual und augmented reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität, 2014