SAP SPORTS ONE

HECTOR SEMINAR
FÖRDERUNG HOCHBEGABTER SCHÜLER

MINECRAFT

Minecraft Modifikation zur Spieldarstellung

Abschlussbericht der Kooperationsphase 2016/2017

Durchgeführt an der SAP Abteilung Sports One

in St. Leon-Rot

Betreuer: Andrew McCormick-Smith, Timo Wolf und Martin Rogge

Kolja Groß
Ziethenstraße 46
68259 Mannheim

Andre Hitter
Joseph-Bauer-Straße 6
68259 Mannheim

Inhaltsverzeichnis

1. Einleitung

Mit dem Ziel, die Leistungsfähigkeit von Spielern verschiedener Teams unterschiedlicher Sportarten zu fördern, hat SAP die Cloud-basierte Software Sports One entwickelt.
Um dieses Projekt voranzutreiben, werden engagierte und vielfältig qualifizierte Mitarbeiter benötigt. Neben Interesse am Sport sind eine Affinität zur Informationstechnologie und Innovationsgeist wichtige Voraussetzungen an zukünftige Mitarbeiter.
Um junge Mitarbeiter anzuwerben, muss eine auf diesem Gebiet tätige Software-Firma anschauliches Material bereithalten können, welches den Aufgaben- und Forschungsbereich klar erkenntlich und anschaulich darstelllt.
Ziel dieses Projektes war es, ausgehend von den Daten über ein Fußballspiel, die im XML-Format vorliegen, eine anschauliche, spielerische und damit motivierende Darstellung des Geschäftsfeldes von SAP Sports One zu realisieren.
Hierzu haben wir das Spiel „Minecraft“ verwendet, in welchem die Daten visuell dargestellt werden sollen.

2. Grundlagen

2.1. XML-Daten

Die Auszeichnungssprache XML (Extensible Markup Language) wird dazu verwendet, Daten als geordnete Text-Daten wiederzugeben.
Hierbei werden erhobene Daten in Elemente mit verschiedenen Attributen, sowie deren Wertzuweisungen unterteilt. Die Elemente können dabei mehrere Attribute zugewiesen bekommen, sowie keines, eines oder sogar mehrere Elemente – mit weiteren Attributen oder untergeordneten Elementen – beinhalten. Die daraus entstandene Datenstruktur lässt sich als Baumstruktur begreifen.
Opta[1] macht sich diese Datenstruktur zunutze. In drei verschiedenen XML-Dateien – Tracking.xml, Events.xml, Lineup.xml – werden alle relevanten Daten des Spiels gespeichert. Während die Datei Lineups.xml neben der Mannschaftsaufstellung hauptsächlich Umgebungsdaten wie Wetter und Uhrzeit beinhaltet, finden sich in Events.xml sämtliche relevanten Ereignisse wie Tacklings, Torschüsse oder auch Pässe, sowie alle an den Ereignissen beteiligten Spieler, der Gewinner und auch der Verlierer. In Tracking.xml findet sich die genaue Ortsbeschreibung jeden Spielers zu jedem Zeitpunkt. Diese Datei ist aufgrund der Aufzeichnung der Daten mit 25 FPS[2] die größte von allen (>300MB). Beide anderen Dateien befinden sich in einer Größenordnung kleiner als 1MB.

Während die Event- und Lineupdaten von Menschenhand erfasst werden, fertigt ChyronHego die Tracking-Daten mit Hilfe eines Computerprogramms an und gibt diese wiederum an Opta weiter. Dazu werden über besondere Kameras die Position des Balles, sowie die der einzelnen Spieler erfasst. Die Datenerhebung erfolgt live während des 90-minütigen Spiels und wird gespeichert und weitergegeben.
Kunden von Opta sind meist Firmen, welche diese Daten aufbereiten oder weiterverarbeiten. Zu diesen Firmen gehört unter anderem auch SAP.

1Gehört zur Sportmediengruppe „Perform“ und ist ein international richtungsweisender Anbieter von Live-Sportdaten. In jedem Bundesligaspiel werden von Ihnen bis zu 60 Millionen Events erfasst.
2Die Abkürzung „FPS“ steht für frames per second (Bilder pro Sekunde) und ist ein Maß für die Bildfrequenz.

In der Event-Datei sind alle Spielevents wie Zweikämpfe, Tore, Schüsse und Pässe enthalten. Diese Events werden der Reihe nach untereinander aufgelistet und mit dem Tag Event geordnet. Im Beispiel ist das Event ein Torschuss. Das Event enthält zusätzlich weitere Attribute, die zeigen von welchem Team das Event ausgeht, hier Hoffenheim. Ein weiteres Attribut gibt den beteiligten Spieler, in diesem Fall Sebastian Rudy, an und offenbart weitere Details, wie z.B. mit welchem Fuß auf das Tor geschossen wurde oder ob der Torschuss aus einem Freistoß hervorging. Insgesamt weist eine Event-Datei in der Regel über 800 Events auf.

```
<Event
    EventTime="2015-09-23T21:48:22.097+02:00"
    MatchId="Game 025"
    EventId="ID"
>
<ShotAtGoal
    Team="Hoffenheim"
    Player="Rudy"
    TypeOfShot="rightLeg"
    ShotOrigin="8"
    BallPossessionPhase="1827"
    AfterFreeKick="false"
    >
    <ShotWide
        Placing="far"
        PitchMarking="over"
    />
    </ShotAtGoal>
</Event>
```

Abbildung 1: Leicht verändertes Codebeispiel aus Events.xml

Die Tracking-Datei hingegen gibt die Position der einzelnen Spieler sowie die des Balles zu jeden Zeitpunkt des Fußballspiels an. Die Positionen werden alle 0,04 Sekunden, also 25-mal in der Sekunde gemessen und in der XML-Datei gespeichert. Hierbei ist jeder Eintrag nach dem gleichen Muster aufgebaut. Alle gesammelten Einträge eines Spielers werden untereinander aufgereiht und unter dem Tag FrameSet gelistet. In diesem Tag sind wiederum mehrere Attribute enthalten. MatchId weist die Einträge einem bestimmten Fußballspiel zu, GameSection zeigt die

```
<FrameSet
    GameSection="firstHalf"
    MatchId="DFL-MAT-0025EV"
    TeamId="DFL-CLU-000007"
    PersonId="DFL-OBJ-0001YS"
>
    <Frame
        N="10000"
        T="2015-09-23T20:01:07.000+02:00"
        X="-2.03"
        Y="-8.41"
        S="0.00"
        M="1"
    />
    <Frame
        N="10001"
        T="2015-09-23T20:01:07.040+02:00"
        X="-1.98"
        Y="-8.33"
        S="0.84"
        M="1"
    />
</FrameSet>
```

Abbildung 2: Codebeispiel aus Tracking.xml

Spielhälfte des Matchs, TeamId gibt das Team an und PersonId den Spieler. Unter dem Tag Frame sind die einzelnen Positionen des jeweiligen Spielers als Attribute abgespeichert. Diese Attribute sind: N ist ein Zähler für die Anzahl der Einträge, T weist die Ortszeit auf, M gibt die Minute des Spiels an, X zeigt an welche Stelle sich der Spieler in X-Richtung befindet und Y zeigt die Y-Koordinate, während S die aktuelle Geschwindigkeit des Spielers angibt. Eine Tracking-Datei enthält ungefähr 3 Millionen Einträge und benötigt entsprechend viel Speicherplatz.

2.2. Minecraft

Das Open-World-Spiel Minecraft wurde vom schwedischen Programmierer Markus Notch Persson erschaffen und erschien erstmals am 10.Mai 2009 für den PC. Seine Firma Mojang, welche im September 2014 für 2,5 Milliarden Dollar vom Microsoft-Konzern gekauft wurde, entwickelte das Spiel weiter und veröffentlichte zusätzliche Versionen für Android, iOS und den Raspberry Pi. Mit 120 Millionen Verkäufen ist es eines der erfolgreichsten Computerspiele weltweit.
Der Spieler kann in dem Spiel aus meist würfelförmigen Blöcken Konstruktionen in einer 3D-Welt errichten. Des Weiteren kann der Spieler Ressourcen sammeln, gegen Monster kämpfen, aus einfachen Blöcken Gegenstände erschaffen und die Welt erkunden. Dabei gibt es hauptsächlich zwei Spielmodi:
Zum einen gibt es den Kreativ-Modus, in dem der Spieler unbegrenzte Mengen an Ressourcen und Gegenstände verwenden kann. Außerdem ist es nicht möglich, Schaden zu erleiden beziehungsweise an Hunger zu sterben. Ebenso ist es möglich, zu fliegen und Blöcke sehr schnell abzubauen. So ist dieser Modus sehr gut geeignet, um große und komplexe Bauwerke zu errichten.
Im anderen Modus, dem sogenannten „Überlebensmodus“, gibt es eine Gesundheitsleiste, deren Punktestand durch Stürze, Ertrinken sowie Angriffe von Monstern reduziert werden kann. Eine Hungerleiste zwingt den Spieler, in regelmäßigen Abständen Nahrung zu sich zu nehmen, um am Leben zu bleiben. Außerdem muss der Spieler die benötigten Ressourcen selbst sammeln und sich durch den Bau eines Unterschlupfs vor den Ungeheuern in der Nacht schützen.
Das Spiel besitzt kein vorgegebenes Spielziel und wurde ursprünglich nur in der Programmiersprache Java entwickelt.

2.3. Arbeitsplan

Für unser Projekt wählten wir den Kreativ-Modus: so hatten wir die Möglichkeit, ein großes Objekt zu bauen, ohne Einschränkungen bei den Ressourcen befürchten zu müssen. Dadurch konnte die Mod[3] zwar nur an einem bestimmten Ort in einer bestimmten Welt funktionieren, doch sie funktioniert so stabiler und fehlerfrei. Auf große Flexibilität wurde an dieser Stelle verzichtet.

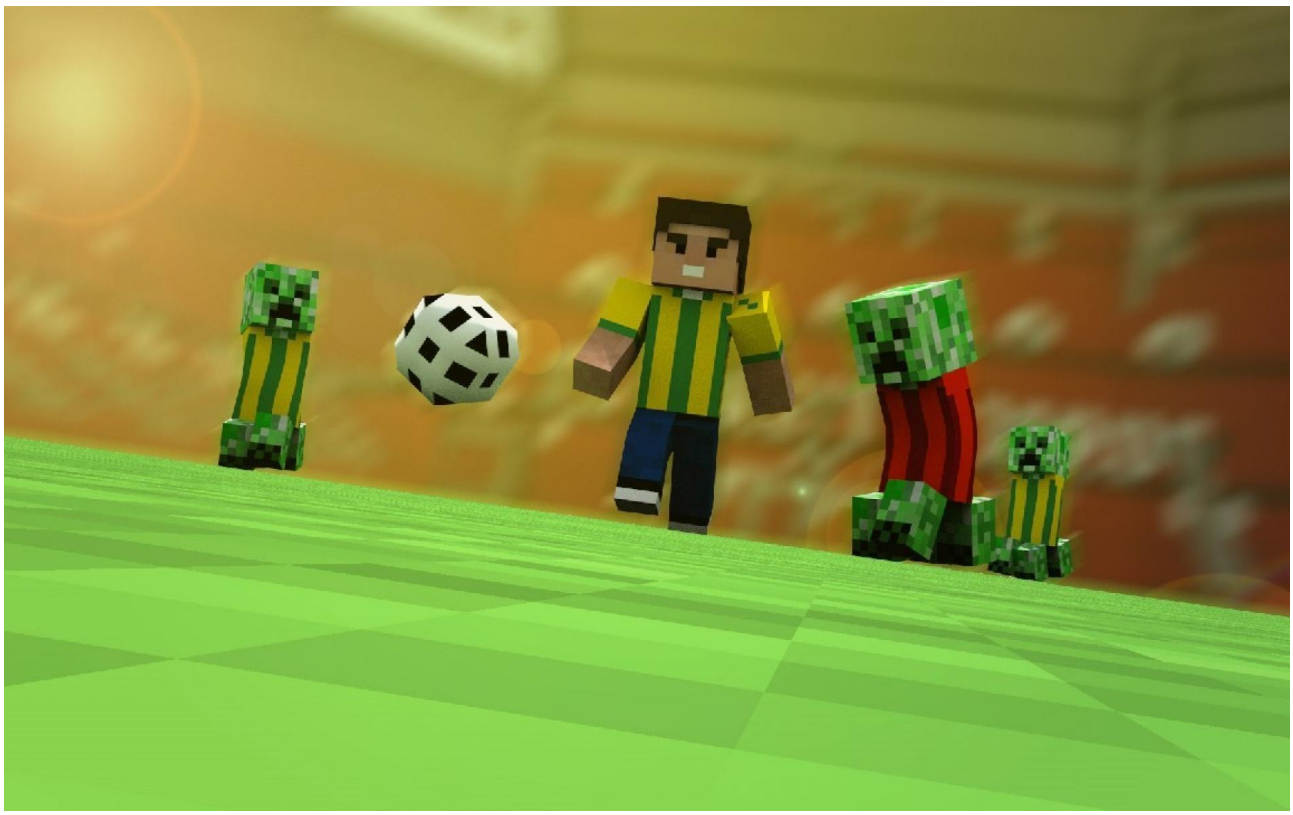

Abbildung 3: Künstlerische Interpretation der Verknüpfung von Minecraft und Fußball

Quelle: "https://www.pinterest.co.uk/pin/482800022533159246/"

3Mod, die (Abkürzung: Modifikation): Erweiterung oder Veränderung eines bereits veröffentlichten Computerspiels

3. Erstellung

3.1. Stadionbau

Das Stadion sollte möglichst ansprechend gestaltet werden. Für die Spielfeldgröße wählten wir sehr einfache Maße (50x25 Blöcke), um etwaige Schwierigkeiten bei der Umrechnung in das reale Spielmaß zu vermeiden.
Ansonsten sollte das Stadion möglichst viele „Fußballstadion-typische“ Eigenschaften besitzen. Als erstes legten wir einen grünen Rasen entsprechend der Maße und zeichneten mit weißen Blöcken die Spielfeldmarkierungen ein. Dann wurde auf den beiden kurzen Seiten mittig jeweils ein Tor aus Holzblöcken erschaffen und auf einer langen Seite zwei Wechselbänke. Im nächsten Schritt wurden zum Spielfeld weitere Elemente eines Fußballstadions hinzugefügt: Aus Holzblöcken wurden ringsum überdachte Tribünen in die Höhe gebaut. Auf den zwei Dächern wurde je eine Anzeigetafel errichtet, die das Spielergebnis anzeigen sollen. Damit das Stadion nicht leer blieb, wurden noch einige Zuschauer implementiert. Ein SAP-Logo durfte in der Stadioneinrichtung natürlich auch nicht fehlen, es wurde durch blaue Blöcke realisiert.

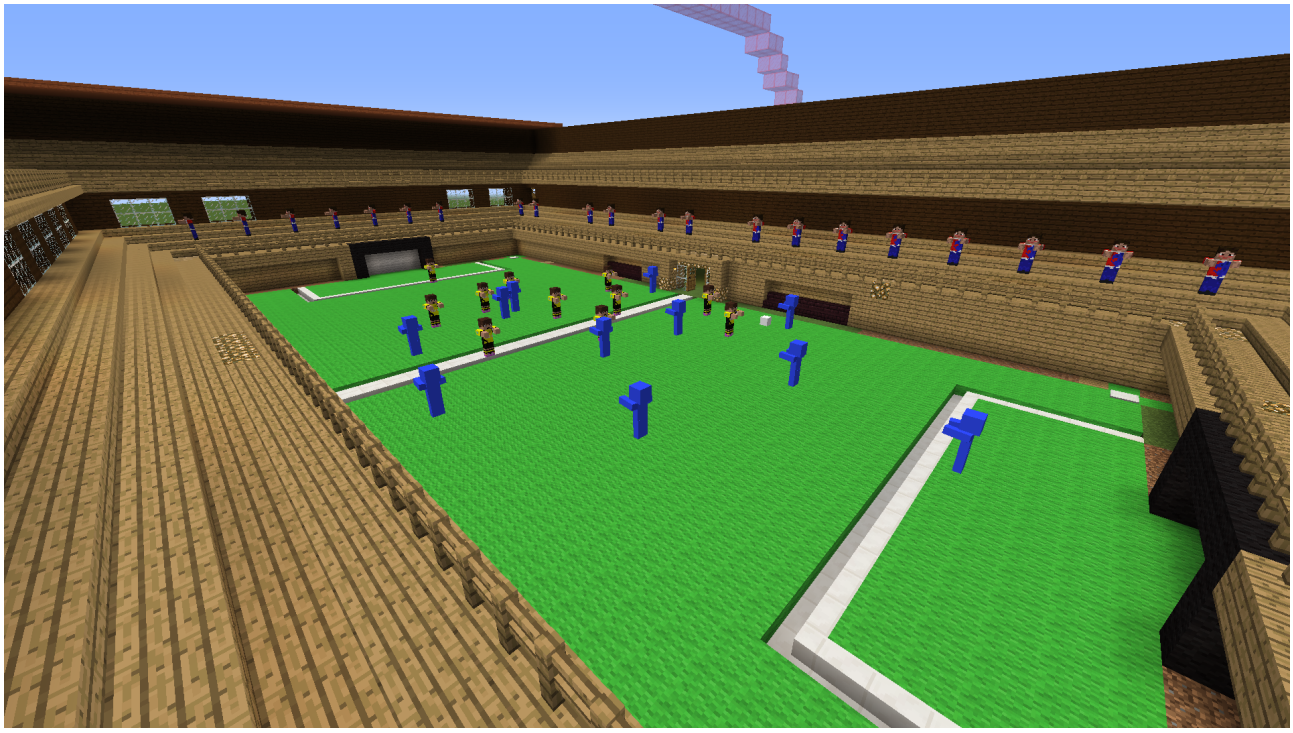

Abbildung 4: Das Stadion

3.2. Programmierung

Hier kann keine vollständige Beschreibung des Programms erfolgen. Ziel dieses Kapitels ist es, die wichtigsten Merkmale des Programms zu erläutern und seine wesentlichen Funktionsprinzipien zu verdeutlichen.

3.2.1. Voraussetzungen

Zuerst einmal ist zu sagen, dass Minecraft auf Java basiert. Ein Zusatz für dieses Spiel muss demnach also auch in Java programmiert sein.
Weitere Voraussetzungen sind nun eine dekompilierte Version von Minecraft. Diese muss außerdem bereits die Erweiterung „Forge Modloader“ enthalten, welche als Schnittstelle für Mods benutzt wird, da sie für Kompatibilität zwischen verschiedenen Mods sorgt und auch die Programmierung durch Hinzufügen einiger Methoden vereinfacht.

3.2.2. Grundgerüst

Zunächst muss ein Grundgerüst geschaffen werden. Der Forge Modloader kümmert sich, wie der Name schon sagt, darum, dass Modifikationen in Minecraft richtig geladen und ausgeführt werden.

Folgendes Codebeispiel zeigt den Grundaufbau der Hauptklasse einer Mod. Mit der Annotation *@Mod* wird die Klasse als Hauptklasse gekennzeichnet und einige Parameter, wie ID, Name und Version der Mod werden übergeben.

Des weiteren sind hier die drei *init()*[4]-Methoden zu finden, in welcher alle Initialisierungsschritte in der richtigen Reihenfolge stattfinden.

Hier werden also alle wichtigen, später benötigten Objekte initialisiert.

Codebeispiel @Mod-Annotation in der Hauptklasse

```
@Mod(modid=[id], name=[name], version=[version])
public class Football {

	@EventHandler
	public void preInit() {

	}

	@EventHandler
	public void init() {

	}

	@EventHandler
	public void postInit() {

	}
}
```

3.2.3. Spieler und Ball

Um das Spiel darzustellen benötigt man nun natürlich Spielerfiguren, welche die echten Fußballspieler darstellen. Dies lässt sich durch das Erstellen neuer NPCs oder auch *Entities*[5] umsetzen.

Es gibt Spieler aus zwei Teams, sowie einen Ball; Objekte, welche alle anhand der Trackingdaten bewegt werden. Diese Voraussetzung legt nahe, ein übergeordnetes Objekt zu erstellen, welches diese Eigenschaften besitzt, sodass den später anzulegenden Objekten Spieler und Ball diese Fähigkeit vererbt wird.

---

4 init als Abkürzung für initialization, was Initialisierung bedeutet.

5 Entity: engl. für Wesen oder Objekt. Gemeint ist hier das Spielerobjekt, nicht zu verwechseln mit dem Java-Objekt, welches im Englischen durch object beschrieben ist.

In diesem Basisobjekt (*GameObjectBase*) ist außerdem, neben dem Einlesen der Daten festgelegt,

- dass das Spielobjekt nicht *despawnen* (verschwinden) kann.
- wie das Objekt in den Speicher von Minecraft eingelesen wird.

Nun können drei weitere Objekte erstellt werden, eines für das Heim-Team, eines für das Auswärts-Team und eines für den Ball. Alle drei Objekte erben hierbei vom eben erstellten Objekt *GameObjectBase*.
Aufgrund der Funktionsweise des *Renderings*[6] in Minecraft wird, um das Aussehen der Spielfiguren festzulegen, jedem der drei Objekte eine Rendering-Klasse zugeordnet. In dieser wird das 3D-Modell, sowie der *Skin* festgelegt.
Der Skin, im deutschen Haut, beschreibt in Minecraft die Farbe und die Gestaltung eines Spielers.

3.2.4. Kommandos

3.2.4.1. Was sind Kommandos?

Die Spielfiguren wurden nun in Minecraft hinzugefügt, jedoch passiert mit diesen bisher noch nichts. Sie tauchen nicht einmal in der Welt auf. Das liegt daran, dass von diesen bisher keine Instanzen erzeugt wurden.
Um dies zum richtigen Zeitpunkt bewerkstelligen zu können, müssen wir uns die Minecraft-Kommandos zu Nutze machen.
Minecraft besitzt eine Eingabezeile, welche über den *ingame*[7]-Chat benutzt werden kann. Dort werden Kommandos durch ein vorangehendes „/“-Zeichen gekennzeichnet.

Einige Vanilla[8]-Minecraft eigene Kommandos wären zum Beispiel:

- /time set [Zeit] :setzt die Tageszeit auf die gewählte Uhrzeit
- /toggledownfall :wechselt den Niederschlag zwischen Regen/kein Regen
- /kill @e :alle Entities dieser Welt werden getötet (auch der Spieler)

6 Rendering: Der Prozess in welchem das letztendlich für den Benutzer sichtbare Bild auf dem Bildschirm errechnet und dargestellt wird.
7 ingame: engl. für im laufenden Spiel
8 Vanilla ist eine Bezeichnung für das unveränderte Minecraft, das heißt, eine Minecraftversion ganz ohne Modifikationen jeglicher Art.

Um ein neues Kommando in Minecraft hinzuzufügen, muss eine neue Klasse erstellt werden, welche das Minecraft-Interface *ICommand* aus dem Paket *net.minecraft.command* implementiert.
Da *ICommand* ein Interface ist, müssen nun einige Methoden überschrieben werden. Diese sind:

- public String *getCommandName*()
  Gibt den Namen des Kommandos als String zurück.

- public List *getCommandAliases*()
  Hier werden verschiedene Abkürzungen für das Kommando festgelegt.

- public void *processCommand*(ICommandSender *sender*, String[] *argString*)
  In dieser Methode wird festgelegt, welche Aktionen beim Aufruf des Kommandos durchgeführt werden sollen. Die Methode hat mehrere Variablen: Neben dem Sender des Kommandos, der in einem Objekt vom Typ IcommandSender gespeichert ist, werden die weiteren Argumente als Strings in einem Array übergeben.

- public boolean *canCommandSenderUseCommand*(ICommandSender var1)
  Diese Funktion benötigen wir nicht, da sie sich um Zugangsberechtigungen kümmert. Daher wird in unserem Fall immer der Wert true zurückgegeben.

Daneben gibt es noch einige weitere, welche hier aber nicht von Bedeutung sind und ohne Anpassung überschrieben werden.

3.2.4.2. Das Hauptkommando

Das wichtigste Kommando, welches durch die Mod hinzugefügt wird, ist /game. Mit ihm ist es zum einen möglich, die Spieldaten zu laden, sowie daraufhin automatisch auch das Spiel zu starten (durch das zusätzliche Argument *track*) und auch das Spielfeld zu leeren (durch das Zusatzargument *clear*).

Im Folgenden wird die Funktion des Arguments *track* weiter erläutert.

Nach dem Aufruf des Befehls wird zuerst eine Chat-Nachricht ausgegeben, um dem Spieler zu verdeutlichen, dass zuerst die Daten eingelesen werden müssen, bevor das Spiel startet.

Daraufhin werden nun die drei Parser[9] für die Tracking-Daten, die Event-Daten und die Lineup-Daten gestartet, jeweils umgeben von try/multi-catch-Blöcken, um eventuelle Fehler abzufangen.

Die Parser lesen alle drei Dateien gefiltert ein, dass heißt es werden die für uns wichtigen Daten (Koordinaten, Spielernamen, etc..) in Objekten gespeichert.

Nachdem dies abgeschlossen ist, wird *f* - die Variable, welche die Zahl der bisher gezeigten Bilder verfolgt - auf 0 gesetzt.

Sie wird mit jedem *ServerTickEvent* (nach Spielstart) erhöht. Das *ServerTickEvent* wird zu jedem *ServerTick* aufgerufen. Das Event findet (normalerweise) genau 20 mal in der Sekunde statt.

Nun werden in zwei for-Schleifen die Spieler in der Welt gespawnt[10]. Die Spieler der Teams sind für einen guten Zugriff in Arrays des Typs Entity[] gespeichert.

9 Parser: Programmteil,welcher für das Einlesen von Daten aus externen Dateien zuständig ist.
10 spawnen: eingedeutscht von engl. erschaffen. Bdt. im Zsmnh.: Zuweisen einer Position und Einfügen an dieser in der Spielwelt

Der Prozess des Spawnens läuft folgendermaßen ab:

1. Dem jeweiligen Element des Arrays wird ein Objekt vom Typ Spieler des jeweiligen Teams zugewiesen.

2. Der Spieler bekommt eine Position in der Minecraft-Welt. Hierbei ist zu bemerken, dass jeder Spieler die gleiche Position zugewiesen bekommt. Der Grund hierfür ist, dass man so sicher gehen kann, dass jeder Spieler mindestens einmal mit irgendeinem anderen Objekt der Welt kollidiert. Das ist nötig, um später die zugewiesenen Rotationen der Spieler zuverlässig darstellen zu können. Die tieferen Gründe für diese etwas umständliche Eigenheit der Minecraft-Engine konnten wir bisher nicht herausfinden.

3. Der jeweilige Spieler wird nun in die Welt gespawnt.

Nach den beiden Teams wird nun auch der Ball gespawnt, welcher schon vorher initialisiert wurde. Nun wird eine weitere Chat-Nachricht an den Spieler gesendet, um ihm mitzuteilen, dass alle Daten eingelesen wurden und das Spiel nun beginnt.
Außerdem wird der Zeitstempel jedes gefundenen Torschuss-Events auch über den Chat ausgegeben, sodass der Benutzer direkt zu diesen Events springen kann.
Zuletzt wird der globalen Variable *started* der Wert *true* zugewiesen, sodass zum Beispiel im bereits genannten *ServerTickEvent* der Code ausgeführt werden kann. Diese Variable wird im nächsten Kapitel noch einmal näher erklärt.

Codebeispiel Track-Kommando:

```
sender.addChatMessage(new ChatComponentText("Getting Data to Start the
                                                  Game.."));

      try {
            football.parser.Tracking.parser();
      } catch (ParserConfigurationException | SAXException | IOException e) {
            e.printStackTrace();
      }
      try {
            football.parser.Events.parser();
      } catch (ParserConfigurationException | SAXException | IOException e) {
            e.printStackTrace();
      }
      try {
            football.parser.Lineups.parser();
      } catch (ParserConfigurationException | AXException | IOException e) {
            e.printStackTrace();
      }

f = 0;
for(int i=0; i<11; i++) {
      tA[i] = new EntityGameObjectPlayer(world);
      tA[i].setPositionAndRotation(3, 4, 3, 90, 0);
      world.spawnEntityInWorld(tA[i]);
}
for(int i=0; i<11; i++) {
      tB[i] = new EntityGameObjectPlayerRed(world);
      tB[i].setPositionAndRotation(3, 4, 3, -90, 0);
      world.spawnEntityInWorld(tB[i]);
}
world.spawnEntityInWorld(ball);
sender.addChatMessage(new ChatComponentText("Data Collected, the Game is
                                                    starting now.."));
for(int i=0; i<football.parser.EventsHandler.events.size(); i++) {
      sender.addChatMessage(new ChatComponentText("ShotAtGoalEvent at " +
                                    football.parser.EventsHandler.events.get
                                    (i).getEventTime()));
}
started = true;
```

3.2.4.3. Weitere Kommandos

Neben dem Hauptkommando gibt es noch weitere neu implementierte Kommandos.
Mit der Chateingabe /jump kann man zu einem Zeitpunkt (in Sekunden) springen.
Mögliche Gründe der Nutzung hierfür ergeben sich, wenn man direkt zu den Torschüssen springen möchte, welche am Anfang im Chat ausgegeben werden.
Des Weiteren gibt den Befehl /fastforward. Dieser nimmt als weiteres Argument auch eine Zahl an, mit welcher die Geschwindigkeit dann multipliziert wird.
Gibt man /spec ein, so fügt man dem Stadion Zuschauer zu. Dieser Befehl ist also rein zur graphischen Verbesserung.

3.2.5. Das ServerTickEvent

Das *ServerTickEvent* wird jeden *ServerTick* zweimal aufgerufen, einmal in der *START*-Phase und einmal in der *END*-Phase. Das bedeutet eigentlich, dass es 40 mal pro Sekunde aufgrerufen wird, aus Stabilitätsgründen sind normalerweise aber nur 20 Ticks pro Sekunde entscheidend, da entweder ausschließlich die erste oder die zweite Phase des Ticks benutzt wird.
In diesem Fall wird nur die Event-Phase *START* benutzt.
Wenn die globale Variable *started* den Wert *true* besitzt, wird jeden Tick, also 20 mal in der Sekunde die Position der Spieler angepasst und auf den Wert des jeweiligen Frames gesetzt. Danach wird die Variable *f* um den Wert von *speed* erhöht. Der Wert des Integers *speed* ist eins, sofern er nicht durch das Kommando /fastforward geändert wird.
Nun wird überprüft ob an dem Zeitstempel, an welchem man sich befindet, ein Torschuss-Event stattfindet und wenn das wahr ist, wird ein Applaus-Ton abgespielt.

Daraufhin kommt folgender Code:

```
if(out == 4) {
            CommandGame.f = CommandGame.f + 1;
            out = 0;
}
out++;
```

Dieser beschreibt, dass jedes vierte Frame übersprungen wird, damit die ursprünglich 25 Frames pro Sekunde, welche unsere Tracking-Daten aufweisen mit den 20 Ticks pro Sekunde übereinstimmen.
Es wird also ähnlich der Next-Neighbour[i] Methode aus der Film-Konvertierung und ungleich der Motion-Interpolation[ii] einfach ein Teil der Frames verworfen.

4. Probleme und Lösungen

Während der Entwicklung standen wir natürlich auch einigen Problemen gegenüber.
Eine große Hürde war es, das Spiel wirklich in Echtzeit abzuspielen, da, wie bereits erwähnt, die Frequenzen des TickEvents und der Tracking-Daten nicht übereinstimmen. Dieses Problem konnten wir nach einigen Überlegungen durch Überspringen der überflüssigen Frames lösen.
Ein weiteres großes Problem hatten wir mit den Ladezeiten der Daten. Eine 300MB große Datei braucht ihre Zeit um eingelesen zu werden, also mussten wir uns überlegen, welche Daten wir denn nun wirklich benötigen und welche zu verwerfen sind. Durch geschickte Konfiguration konnten wir hier die Zeit von 5 Minuten auf teils weniger als 10 Sekunden verkürzen.

5. Fazit zum Projektpartner

Neben einem sehr interessanten Einblick in die Arbeitsstruktur der SAP konnten wir auch viele spezifische und lehrreiche Dinge mitnehmen.
Allein schon die praktische Erfahrung im Programmieren mit Java hat uns sehr weitergeholfen. Hier hinzu kam die ständige Hilfe von Mitarbeitern, welche wir jederzeit zu Rate ziehen konnten.
In den Meetings am Ende der beiden Arbeitswochen, an welchen wir unser Projekt vorgestellt haben, konnten die Mitarbeiter des Sports One Teams einen Einblick in unsere Arbeit erhalten, aber auch weiterhelfen. Hierbei war sehr schön zu spüren, dass sich viele für unsere Arbeit interessierten und sich freuten helfen zu können.
Daher kann man eine Kooperation mit der SAP und speziell mit der Abteilung „Sports and Entertainment“ von unserer Seite aus nur empfehlen.

6. Danksagungen

Wir bedanken uns zuerst sehr bei Herr Dr. Andrew McCormick-Smith, welcher unsere Kooperation organisiert hat und sehr bemüht war, uns einen Einblick in die Arbeit des Sports One Teams zu geben.

Des Weiteren möchten wir uns bei Herr Timo Wolf bedanken, welcher sich mit um die Organisation kümmerte und uns bei der Ideenfindung und Umsetzung geholfen hat.

Außerdem möchten wir uns bei Herr Martin Rogge bedanken, welcher sich während der Kooperation tatkräftig bemüht hat, uns zu unterstützen und uns Tipps und Hilfen zur weiteren Projektumsetzung zu geben.

Herrn Christoph Gölz danken wir für die Betreuung des Projekts und sein Interesse an einer interessanten und schönen Zusammenarbeit.

Ebenfalls möchten wir unseren langjährigen Kursleitern Frau Monika Krämer und Herr Matthias Taulien danken, welche sich nun bereits seit 6 Jahren um uns kümmern.

Zu unseren wichtigsten Unterstützern gehören natürlich auch unsere Eltern, welche uns auf diesem jahrelangen Weg begleitet haben.

Ein ganz besonderes Dankeschön gilt Frau Josephine und Herr Dr. Hans-Werner Hector, Stifter des Hector-Seminars, welches uns eine langjährige Förderung ermöglicht hat.

7. Quellen

www.stackoverflow.com

www.wikipedia.org

- /Minecraft (30.08.2017)
- /extensiblemarkuplanguage (30.08.2017)

www.optasports.com (28.08.2017)

www.gamepedia.de

www.sap.com

8. Selbständigkeitserklärung

Hiermit versichern wir, dass wir diese Arbeit unter der Beratung von Herrn Gölz selbstständig verfasst haben und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, sowie Zitate kenntlich gemacht haben.

Ort, Datum Unterschrift

Anhang

i Next-Neighbour Methode: Um die Anzahl der Bilder pro Sekunde eines Films anzupassen wird bei dieser Methode ein Teil der Frames verworfen. Soll also ein Film mit ursprünglich 25 Bildern pro Sekunde nun mit 20 pro Sekunde angezeigt werden, so wird von 20teln auf 25teln gerechnet und entstandene Bruchzahlen werden auf die nächste Ganzzahl (next neighbour) gerundet. Dadurch wird in diesem Beispiel jedes fünfte Bild einfach verworfen.

ii Motion-Interpolation: Anstatt einfach Bilder zu verwerfen, werden hier Zwischenbilder errechnet, wofür einfach die angrenzenden Bilder miteinander verrechnet, also interpoliert, werden. Diese Art der Interpolation ist zwar realitätsnäher, aber alles andere als Ressourcen schonend.