Mancala Projekt Dokumentation

Team "XML DTD" - XML
Praktikum WS 2016/2017

Ton That, Khiem

Trinh, Phi Long

Tran Trung, An

Goldberg, Viktor

Mancala Projekt Dokumentation: Team "XML DTD" - XML Praktikum WS 2016/2017

Ton That, Khiem
Trinh, Phi Long
Tran Trung, An
Goldberg, Viktor

Publication date 27.03.2017

Table of Contents

1. Einleitung 1
2. Design und Architektur 2
    - Mancala Spielregeln 2
        - Mulden und Häuser 2
        - Aussähung der Steine 2
        - Das Haus 2
        - Freier Zug 2
        - Ende 2
        - Sieg 2
    - Konzept und Entwurf 2
        - Diagramm und Architektur 3
        - Graphical User Interface (GUI) 3
        - Client/Server Modell 5
3. Implementierung 7
    - Namespaces 7
    - Single Node Updates mittels XQuery 7
    - GUI mit SVG, CSS, HTML 8
4. Zusätzliche Features 9
    - Deployment und Inbetriebnahme 9
    - Continuous Integration 9

List of Figures

2.1. UML-Klassendiagramm 3
2.2. Alpha Version des Spielbretts 4
2.3. Produktive Version des Spielbretts 4
2.4. Spielbrett mit überliegender Logik-Ebene | Spieler 1 am Zug 5
2.5. Spielbrett mit überliegender Logik-Ebene | Spieler 2 am Zug 5
2.6. Client/Server Architektur 6
3.1. Screenshot des implementierten Spiels im Webbrowser CLIQZ 8

Chapter 1. Einleitung

Mancala ist ein antikes Brettspiel, welches in verschiedenen Varianten gespielt wird. Dieses "Zähl-und-Fang" Spiel wird von zwei Spielern gespielt, wobei das Ziel ist, die meisten Steine im eigenen Haus zu sammeln. Im Rahmen des Praktikums XML-Technologien im WS 2016/17 entwickelt das Team "XML-DTD" dieses Spiel unter Einsatz des XML Stacks. Dieses DocBook repräsentiert die Dokumentation für die Vorgehensweise als auch Umsetzung und Inbetriebnahme des Online-Spiels.

Hinweis: Die Darstellung der einzelnen Abbildungen in dieser Dokumentation wurde speziell für das PDF-Format optimiert.

Chapter 2. Design und Architektur

Mancala Spielregeln

In dieser Implementierung wird das Augenmerk auf das sogenannte zweistufige Mancala - besser bekannt als "Kalah" - gelegt. Hierfür werden initial 48 Steine auf 12 Mulden gleichverteilt.

Mulden und Häuser

Die Mulden eines jeweiligen Spielers befinden sich unten bzw. auf der rechten Seite. Das Haus des Nutzers ist das große Element auf der rechten Seite bzw. in der oberen Ecke.

Aussähung der Steine

Der Spieler wählt eine seiner Mulden aus, aus der die Steine ausgesäht werden sollen. Die Aussähung findet gegen den Uhrzeigersinn statt, wobei in jede Mulde ein Stein gelegt wird, bis alle Steine ausgesäht sind.

Das Haus

Wird bei der Aussähung das eigene Haus passiert, so wird ein Stein abgelegt. Wird beim Durchlauf das Haus des Gegners passiert, überspringt man es.

Freier Zug

Fällt der letzte Stein in das eigene Haus, so hat der Spieler einen freien Zug.

Ende

Das Spiel ist beendet, wenn alle sechs Mulden auf der Seite eines Spielers leer sind. Der Spieler, der dann immer noch Steine auf seiner Seite liegen hat, fängt diese.

Sieg

Die Steine in beiden Häusern werden gezählt. Der Spieler mit der größeren Anzahl an Steinen gewinnt.

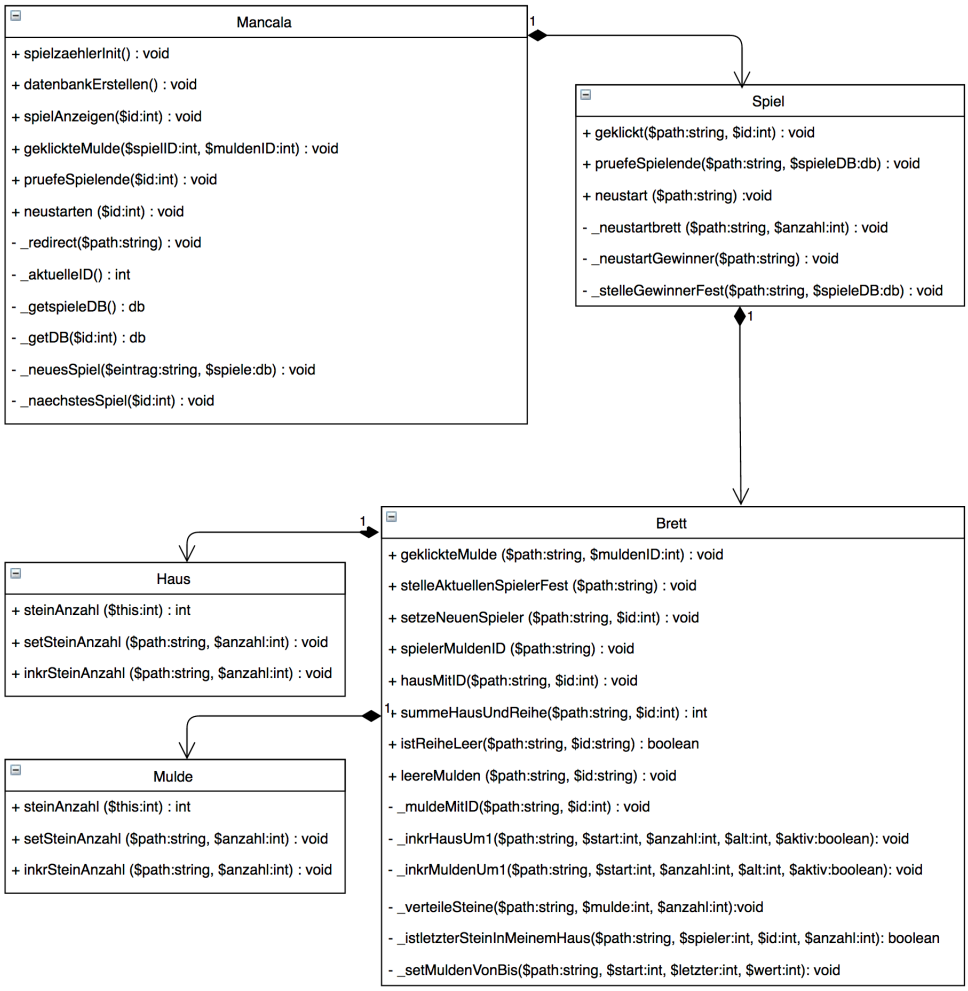
Konzept und Entwurf

In diesem Praktikum werden eine Reihe von XML-Technologien eingesetzt, u.a. SVG, XSLT, XQuery und BaseX.

- **SVG (Scalable Vector Graphics)** wird zur Erstellung von zweidimensionalen Web-Graphiken, die vorallem eine gewisse Interaktivität und Animation mit sich bringen verwendet.
- **XSL und XSLT (Extensible Stylesheet Language)** bildet die Style-Sprache in XML (ähnlich zu CSS bei HTML). XSLT (Transformation) hilft bei der Umwandlung von XML Dokumenten in andere Formate, wie z.B. HTML und PDF.
- **XQuery** wird als Sprache zum Auslesen und Beabeiten von XML Tabellen eingesetzt, analog zur SQL-Sprache für Datenbanken.
- **BaseX** ist eine skalierbare und hoch-performante XML Datenbank, die das Durchsuchen und Abspeichern von XML Daten erlaubt.

Diagramm und Architektur

Figure 2.1. UML-Klassendiagramm



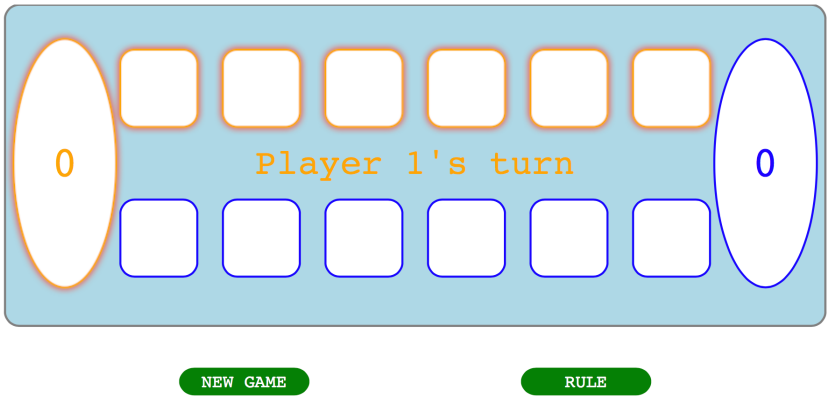
Das UML-Klassendiagramm zeigt die Integration der einzelnen Klassen. Die folgende Auflistung beschreibt die implementierten Klassen mit Fokus auf Abhängigkeiten und die gegenseitige Interaktion. Die Klassen stellen jeweils die XQM-Dateien dar, mit ihren jeweiligen Methoden.

- **Mancala**: Verwaltet die gesamte Seite, die angezeigt wird, als auch die RestXQ Abhängigkeiten, für die unterliegende Client/Server Kommunikation.
- **Spiel**: Enthält einige Methoden zur tatsächlichen Spiellogik, die zur besseren Übersichtlichkeit noch einmal die Module Brett, Haus und Mulde unterteilt wurden.
  - **Brett**: Sämtliche Abhängigkeiten, die mit dem Brett in Verbindung stehen, kommen hier zusammen. Vor allem die Interaktion des Spielers wird hier abgebildet.
  - **Haus**: Verwaltung der Steine, die in einem Haus liegen.
  - **Mulde**: Verwaltung der Steine, die in einer Mulde liegen.

Graphical User Interface (GUI)

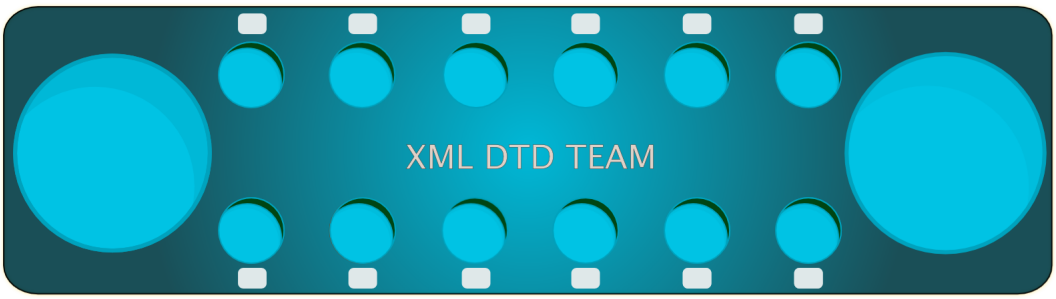
Die HTML Seite des Projekts bildet die graphische Schnittstelle. Diese wurde mittels einer SVG Datei generiert und anschließend mit XSLT für den Webbrower transformiert. Der gesamte Code liegt in folgender XML Datei: “webapp/static/mancala.xml”.

Figure 2.2. Alpha Version des Spielbretts



Wie in Abbildung 2.2 dargestellt, wurde das Spielbrett zunächst mittels einer eigens-verfassten SVG Datei erstellt. Somit hat sich das Team mit der SVG und XSLT Transformierungstechnologie auseinandergesetzt. Anschließend wurde wie in Abbildung 2.3 das unterliegende und statische Brett mittels Photoshop entworfen und in das SVG Format automatisch übersetzt.

Figure 2.3. Produktive Version des Spielbretts

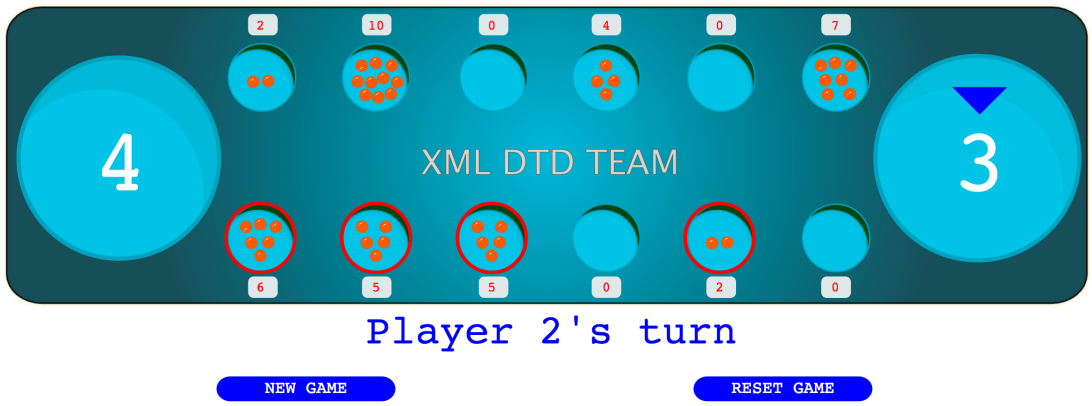


Sämtliche Funktionalität wurde anschließend auf das generierte statische Brett als überliegendes Layer implementiert. Die Funktionen bzw. Komponenten beinhalten unter anderem:

- **Steine**: Hierfür wurden 12 Graphiken erstellt und angelgt, damit diese je nach Zählerstand in die jeweilge Mulde gelegt werden können.
- **Schaltflächen**: Zwei Schaltflächen (Buttons) zum Anlegen eines neuen Spiels "NEW GAME" und für das Zurücksetzen eines laufeneden Spiels wurden mittig unter das statische Spielbrett eingefügt.
- **Markierungen**: Die Mulden des Spielers, der gerade am Zug ist, werden mit einer Markierung (organge) umrandet. Dies wurde mit 12 transparenten und zugleich klickbaren Kreisen implementiert. Ein besonderes Features ist, dass die XSLT den aktuellen Spielzustand ausließt und den Kreis nur umrandet und klickbar macht, wenn auch Steine darin vorhanden sind und außerdem der zugehörige Spieler am Zug ist.
- **Texte**: Unter anderem die Anzahl der Steine im jeweiligen Haus und die einzelnen Zähler unter den Mulden. Zudem wird über bzw. unter dem Brett angezeigt, welcher Spieler gerade am Zug ist oder gewonnen hat.
- **Pfeile**: Über dem Haus wird jeweils angezeigt, welcher Spieler gerade am Zug ist.

Figure 2.4. Spielbrett mit überliegender Logik-Ebene | Spieler 1 am Zug

Figure 2.5. Spielbrett mit überliegender Logik-Ebene | Spieler 2 am Zug



Die beiden Abbildungen 2.4 und 2.5 zeigen sowohl die Ansicht wenn Spieler 1 als auch Spieler 2 am Zug sind. Die überliegende Logik des Brettspiels ist hier gut zu erkennen. Pro Mulde werden aus Darstellungsgründen maximal 12 Steine angezeigt. Sollte während eines Spiels die Anzahl der Steine pro Mulde 12 übersteigen, so wird nur noch die tatsächliche Anzahl im Feld darüber/darunter modifiziert, jedoch weiterhin die Graphik für 12 Steine eingeblendet. Dieses Phänomen konnte jedoch bei zahlreichen Spielversuchen nur in Ausnahmefällen nachgestellt und beobachtet werden.

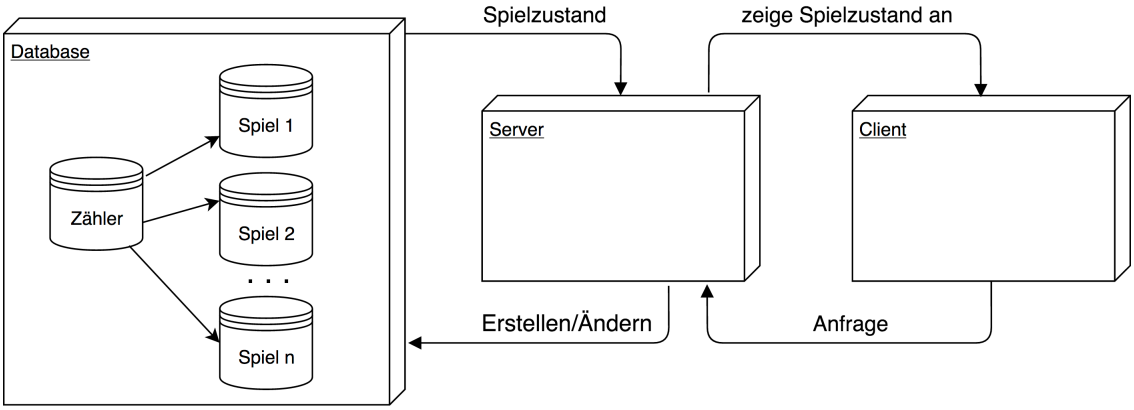
Client/Server Modell

Die Architektur wurde so gewählt, dass die Umgebung dem klassischen Client/Server Modell mit den folgenden Komponenten entspricht:

- Datenbank
- Client
- Server

Dieses Modell wird detailliert in nachfolgender Abbildung dargestellt.

Figure 2.6. Client/Server Architektur



Chapter 3. Implementierung

Für die Implementierung der vorhergehenden Architektur und des Designs wurde eine Dateistruktur festegelegt, die wie folgt aussieht:

- *spielstart.xml*: Der Spielstart enthält die initialen Werte für die Generierung der **Steine**, **Mulden** und **Häuser**, als auch eine **Spiel-ID** zur Identifizierung des Spiels in der Datenbank und zwei weitere Attribute:
  - *istAmZug*: Defeniert, welcher Spieler (Spieler 1 oder Spieler 2) gerade am Zug ist
  - *gewonnen*: Bestimmt, ob das Spiel bereits gewonnen wurde und wenn ja, von wem.
- *spielzaehler.xml*: Enthält ein einziges Attribut, das als Zähler dient, um zu bestimmen, wie viele Spiele bereits in der Datenbank vorhanden sind.
- *mancala.xml*: Generiert die Oberfläche. Es enthält außerdem alle notwendigen Elemente des Spielbretts. Die Daten mit zugehöriger Spiel-ID werden aus der Datenbank ausgelesen und anschließend in diese XML-Datei übertragen und darüber präsentiert. Die Präsentation funktioniert durch Konvertierung der XML-Datei in eine SVG mit Hilfe von XSLT.
- *mancala.xqm*: Diese Datei enhält alle Methoden, die im Klassendiagramm aus Sektion 2.2.1 der Architektur beschrieben wird.

Namespaces

Zur objekt-orientierten Umsetzung des Programms und vor allem der Implementierung, wie im Klassendiagramm dargestellt, wird das Konzept der Namespaces angewandt. Dadurch lässt sich hauptsächlich eine gewisse Klassenorientierung (Class/Scope) simulieren. Die Namespaces, die implementiert wurden, umfassen:

- *seite*: Zuständig für die Kategorisierung aller Methoden, die mit dem Seitenaufbau bzw. der Präsentation in Verbindung gesetzt werden.
- *spiel*: Verwaltet alle aufgelisteten Methoden, die zur tatsächlich logischen Komponente des Spiels gehören.

Um den Namepsace *spiel* noch einmal zu unterteilen, wurde zur Übersichtlichkeit

- *brett*
- *mulde*
- *haus*

eingeführt.

Single Node Updates mittels XQuery

Wird ein XQuery Skript aufgerufen, so wird es erstmal vollständig geparsed und anschließend zum Ende des Skripts erst ausgeführt. Dadurch ergibt sich die folgende Herausforderung:

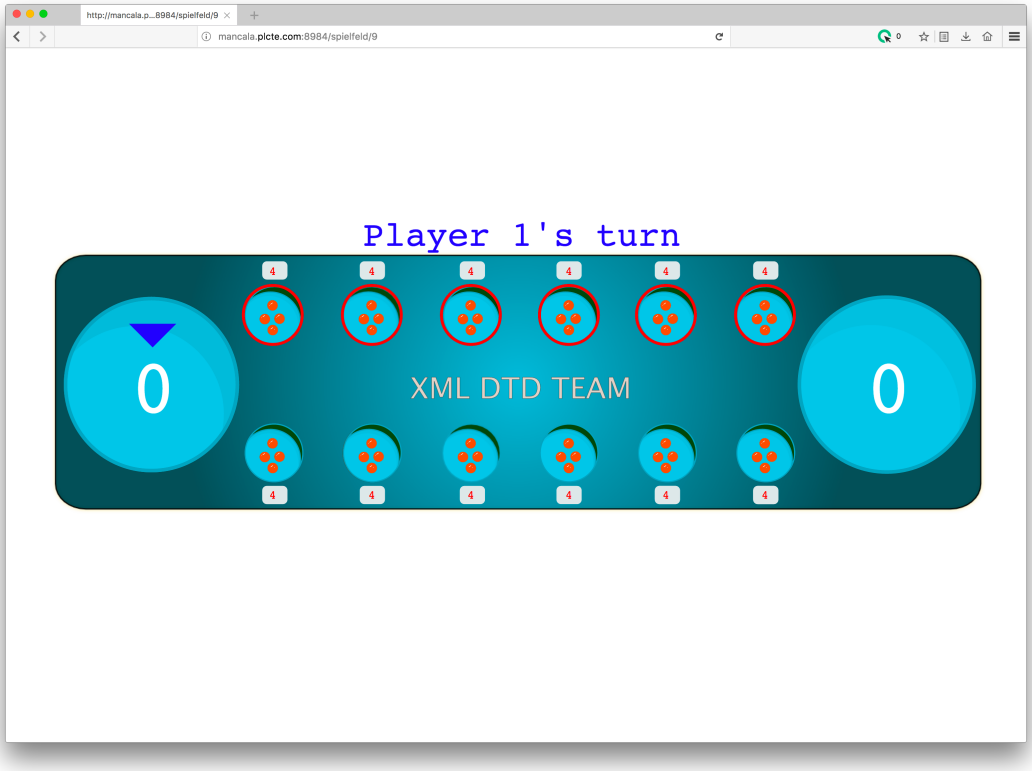
Ein Knoten kann nicht zwei mal zur gleichen Zeit/Abfrage aktualisiert werden.

Das Problem wird umgangen, indem die Programmlogik entsprechend den Möglichkeiten von XML angepasst wurde.

GUI mit SVG, CSS, HTML

Die GUI wird vollständig als SVG generiert und mit Hilfe von XSLT in das HTML Format übersetzt. Dediziertes HTML und CSS kommt in der vorliegenden Implementierung nicht zum Einsatz. Das Ergebnis ist in Abbildung 3.1 zu sehen.

Figure 3.1. Screenshot des implementierten Spiels im Webbrowser CLIQZ



Chapter 4. Zusätzliche Features

Das DTD-Team hat sich dazu entschieden noch eine Reihe an zusätzlichen Funktionen und Features einzubinden, die vor allem für die Weiterentwicklung hilfreich sind und die allgemeine "User Experience" erheblich steigern.

Deployment und Inbetriebnahme

Die Applikation wird auf einem weltweit erreichbaren Server für die Dauer des Praktikums zur Verfügung gestellt. Dieser Server wird in der Microsoft Azure Cloud gehostet und über den Public-DNS-Hostname "mancala.plcte.com" erreichbar gemacht.

Bei dem Server handelt es sich um eine virtuelle Maschine mit folgenden Spezifikationen:

- Debian/GNU Linux 8 Jessie build 3.16.39-1
- 64-bit Architektur
- 1 GB RAM
- 15 GB Festplatten-/Hauptspeicher

Zur Inbetriebnahme des mit basex ausgestatteten Jettie Servers wurde außerdem das Paket "basex" über das Aptitude Source Verzeichnis mit folgendem Befehl heruntergeladen: "apt-get install basex".

Nachdem das basexhttp Skript aus dem "bin/"-Ordner gestartet und in den Hintergrund (daemonized) verlargert wird, kann auf das Spiel weltweit über jeden verfügbaren Webbrowser mittels **http://mancala.plcte.com:8984** zugegriffen werden.

Die Browser, die im Rahmen des Praktikum getestet wurden, sind:

- Apple Safari
- Google Chrome
- Mozilla Firefox
- Microsoft Internet Explorer / Edge
- CLIQZ

Continuous Integration

Zur übersichtlicheren und nachhaltigeren Weiterentwicklung wurde auf die bewerte Verfahrensweise "Continuous Integration" gesetzt. Dies erlaubt es den aktuellen Code in einem Versionierungssystem (in unserem Fall **git**) zu pflegen und den Head (bzw. die Master-Branch) direkt - aus der Entwicklungsumgebung heraus - produktiv zu *committen*. Der Server wird anschließend bei jedem Commit durch eine sogenannte *Webhook* vollautomatisch neugestartet und die App somit provisioniert.

Das Entwicklungsteam kann somit an einem neuen Feature im sog. *Feature-Branch* arbeiten und bei Abschluss und zugehörigem Funktionstest die neue Funktion direkt online schalten, ohne dass die Berührung des produktiv laufenden Codes erforderlich ist.

Continuous Integration ermöglicht somit einen reibungslosen Entwicklungsablauf und schränkt während der Entwicklung die Erreichbarkeit der Applikation nicht ein. Somit wird auch die Verfügbarkeit des Spiels gewährleistet.