

Semesterprojekt: Architekturdokumentation

Mensch ärgere Dich nicht!

Martin Puse
853001

Marcel Pillich
863121

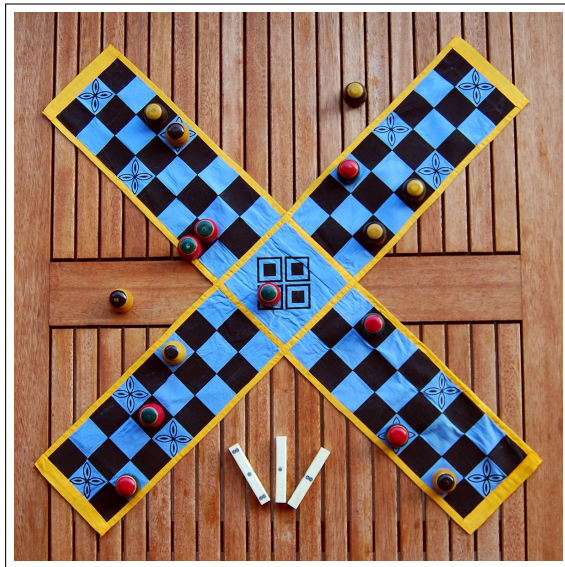


Fig. 1. Pachisi

Abstract—Die vorliegende Arbeit enthält eine detaillierte Beschreibung einer Java-Applikation für des Spiels "Mensch ärgere Dich nicht", welche für Spieleprogrammierer und Spieledesigner gleichermaßen geeignet ist. Ziel war es, dass der Programmierer sich um Erscheinungsbild und Regeln des Spiels kümmern kann, ohne selber Spielfelder designen zu müssen, wohingegen der Designer keine Programmierkenntnisse benötigt, um neue Spielfelder hinzuzufügen und zu testen. Zu diesem Zweck wurden eine Grammatik für die Spielfelder und ein Generator, welcher die Grammatiken verarbeitet und automatisiert Code erzeugt, erstellt. Die dahinterliegende Idee, die Spielfelder nicht auf Positions-, sondern Knoteninformationen zu basieren, wird ausführlich dargestellt. Schlussendlich werden Weiterentwicklungsmöglichkeiten des Prototyps präsentiert.

I. EINLEITUNG

"Mensch ärgere Dich nicht" ist ein Brettspiel für bis zu 4 Personen. Es zählt zu den Klassikern im deutschsprachigen Raum und wurde 1907/1908 von Josef Friedrich Schmidt in Anlehnung an das englische Spiel Ludo erfunden, die Ursprünge finden sich jedoch im indischen Spiel Pachisi. Erstmals 1910 erschienen, ging es 1904 in Serie. Mittlerweile existieren auch viele Abkömmlinge und Varianten mit veränderten Regeln oder Spielfeldern. Allen Varianten ist jedoch gemein, daß sie ein symmetrisches Spielfeld besitzen und analog entwickelt wurden. Man kann vermuten, dass mit

asymmetrischen Spielfeldern experimentiert, dies jedoch nicht weiter verfolgt wurde, da es ungleich schwieriger ist, eine asymmetrische Konfiguration zu finden bei der die Chancengleichheit und somit der ausgewogene Spielspaß für alle Spieler gegeben ist. Dennoch ist der Reiz dieses Szenarios nicht zu verachten, da die Freiheit bei der Gestaltung asymmetrische Spielfelder äußerst fruchtbar für die Entwicklung interessanter Spielvarianten, welche sich nicht in das relativ starre klassische Korsett einpassen müssen, ist. Die größte Hürde in diesem Bereich ist also die mühsame Erstellung eines experimentellen Spielfeldes, um dieses unmittelbar auf seine Spielspaßeigenschaften zu testen.

Zu diesem Zweck wurde eine Java-Applikation entwickelt, welche einen Spielfeldgenerator implementiert, der mittels einer DSL variable Spielfelder erzeugen kann. Im Verlauf der Architekturdokumentation wird sowohl auf die Besonderheiten der Implementierung, als auch auf die verwendeten Design-Patterns und den Code-Generierungsansatz eingegangen. Abschließend wird ein Fazit formuliert, das die Erfahrungen und Ergebnisse zusammenfasst.

II. ANFORDERUNGEN AN DIE APPLIKATION

Grundlegend an die Anforderungen ist die Tatsache, dass das Programm für Programmierer und Designer gleichermaßen benutzbar sein muß. Während der Designer sich vollständig auf die Entwicklung interessanter Spielfelder konzentrieren können muß, d.h. man kann und darf keine Programmierkenntnisse vom ihm verlangen, darf die den Programmierer nicht daran hindern, die Applikation weiter zu entwickeln, also in den Bereichen grafisches Erscheinungsbild, Regeln, Künstliche Intelligenz und Spielfeld Veränderungen vorzunehmen. Aus diesen Gedanken heraus haben sich die grundlegenden Anforderungen gebildet. Die Applikation sollte möglichst unabhängig vom Betriebssystem lauffähig sein. Da eine starke Eigenschaft von Java die Portabilität und Lauffähigkeit auf vielen unterschiedlichen Geräten ist, wurde diese als Entwicklungsprogrammiersprache gewählt. Dadurch lassen sich auch Abkömmlinge des Prototypen (exemplarisch MobileApp) gut planen und umsetzen. Die verbindende Technologie zwischen Programmierer und Designer ist das ausgereifte Codegenerierungs-Tool Antlr, welches das .csv Dateiformat von Haus aus beherrscht und für die programmierkenntnislose Erstellung von Spielfeldern geeignet ist. Diese Aufstellung des Technologiestacks erlaubt einen wechselseitigen Workflow ohne große Unterbrechungen. Der Designer kann mit

der aktuellen Version neue Spielfelder erzeugen oder bereits designte testen und Vorschläge zum Regelwerk machen. Der Programmierer kann solche Feature-Requests umsetzen und einspielen, woraus der Designer wieder iterieren kann. Beide kommunizieren also vorrangig über die Grammatik des Spiels und sind nicht primär an technische Beschränkungen gebunden. Weiter kann der Programmierer unabhängig vom Designer an der graphischen Erscheinung der View und dem Verhalten der künstlichen Spieler arbeiten. Für den Designer ist eine einfache Bedienung wichtig, d.h. es wird nur von ihm verlangt mit der View zu kommunizieren und neue Spielfelder zu erzeugen.

- unabhängig lauffähig - trotz development dauerhaft lauffähiger Zustand (für Designer) - keine Programmierkenntnisse, um neue Spielfelder zu erzeugen und zu testen - Erweiterbarkeit der Regeln - Erweiterbarkeit für künstliche Spieler - einfachste Bedienung (nur Maus + linke Maustaste)

Usage für Designer: entwickle Spielfeld, starte Generator, starte Applikation, vorschläge an den Programmierer zur Weiterentwicklung
Usage für Programmierer: entwickle View, erweitere Regeln (Controller: Businesslogic, Model: neue Datenfelder)
erweitere Generator/Grammatik nach Implementierung Designer
neue Version zur Verfügung stellen

III. IMPLEMENTIERUNG

parallele Entwicklung (2er Team)

Tech Stack: Java Antlr beliebiger Texteditor für CSV

Projekte: Applikation (MenschAergerDichNicht) Generator (MADPlayfieldGenerator)

einfache und konfigurationslose Verknüpfung (generierte Spielfelddatei: fester Platz und Name in Projektstruktur (Model GENPlayfieldCreator.java))

Klassen:

Controller:

Game (HauptController) Hauptschleife

PlayerController (UnterController) Kapselung für Player Objekte

View: GUI -verantwortlich für den Rahmen, Fenstergröße usw
PlayfieldPanel - Rendering des Spielfeldes (tiles, pieces)

Model: Playfield - alle tiles Tile - Knoteninformationen - Referenzinformation über pieces Player - enthält seine pieces
Ableitungen: KIPlayer/HumanPlayer

Implementierungsvorteile durch Knotenbasierte Spielfeldelemente

-erzeugt eleganten Regelcode durch Tiletypen und lookahead function -einfache Erweiterung neuer Tiletypen und Regeln - natürlich Abbildung der Regeln in Code

Typendefinitionen: TileType

Beispiele zeigen: if (tile.TILETYPE == START && tile.hasPiece())) // cant move to start, its not free

IV. GENUTZTE DESIGN PATTERN

A. MVC

Für die Abbildung eines Brettspiels in digitaler Form bietet sich das klassische MVC Pattern an.

dabei ist zu beachten, dass die Dreiecksbeziehung zwischen MVC nicht symmetrisch (führt zu spaghetti code, unklare Verantwortlichkeiten) sondern hierarchisch implementiert wird. Konkret: Controller kennt Model und View d.h. Controller darf Veränderungen im Model vornehmen und bekommt Events der View mitgeteilt. View kennt Model, darf aber nur lesen, um eigenen Zustand upzudaten. Model kennt niemanden, stellt aber API für Lese/Schreibzugriffe bereit.

Das Spielfeld und der aktuelle Spielzustand sind im Model festgehalten, die View präsentiert das Spielfeld und erwartet Interaktion vom Benutzer, der Controller verarbeitet die Eingaben des Nutzers oder steuert die künstlichen Spieler.

B. Singleton

Die Applikation soll stets ein lauffähiges Mensch-gegen-Maschine Spiel repräsentieren, welche ohne komplizierte Laufzeiteinstellungen auskommt oder mit Commandlineparametern zu starten sein muss (Designer Requirement). Es ist also nicht erwünscht, dass mehrere Instanzen der Hauptklassen erzeugbar sind, da dies zu kompliziertem Initialisierungscode in den Toplevelschichten führen würde (Main). Im Code wird dies dadurch reflektiert, dass die Hauptklassen von Model/View/Controller als Singletons implementiert sind. Die Eleganz dieser Variante zeigt sich in der übersichtlichen Mainklasse, in der nur noch die Singletons instanziiert und miteinander verknüpft werden. Sämtliche Weiterentwicklung des Programms kann innerhalb der MVC-Komponenten erfolgen, ohne dass die Main angepasst werden muss. Dadurch ist ein Initialisierungsprozess der App festgehalten.

V. CODE-GENERIERUNG

2-stufiger Parser

Antlr Setup, Usage

Meta Parser: liest Spielfelddatei und ermittelt Metadaten, also Daten das ganze Spielfeld betreffend und positionsunabhängig
- Spieleranzahl - ein Startfeld pro Spieler - gleiche Anzahl an Home und Goal Feldern pro Spieler - Höhe, Breite der Spielfeldmatrix - Daten werden als Getter im generierten Code bereitgestellt
Semantic Parser - liest die einzelnen Felder - verknüpft die Knoten - generiert den Erzeugungscode des Spielfeldes

VI. FAZIT

Knoten Ansatz sehr gut geeignet für Erweiterung/generalisierung der Spielfeldgenerierung
Einsatz passender Pattern an den richtigen Stellen für Weiterentwicklung der Software (KI Subclass, Spielregeln)

Grammatik zeilen/spalten gebunden (keine planar frei platzierbare Tiles ohne x, y Beschränkung) aber möglich, Tiles nur Knotengebunden, d.h. zusätzlicher Editor für graphisches Design von CSV-Vorlagen möglich
Verschieben von graphischen Elementen hat keinen Einfluss auf Knotenverknüpfungen

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

- [1] H. Kopka and P. W. Daly, *A Guide to L^AT_EX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.