Semesterprojekt: Architekturdokumentation Mensch ärgere Dich nicht!

Martin Puse
853001

Marcel Pillich
863121

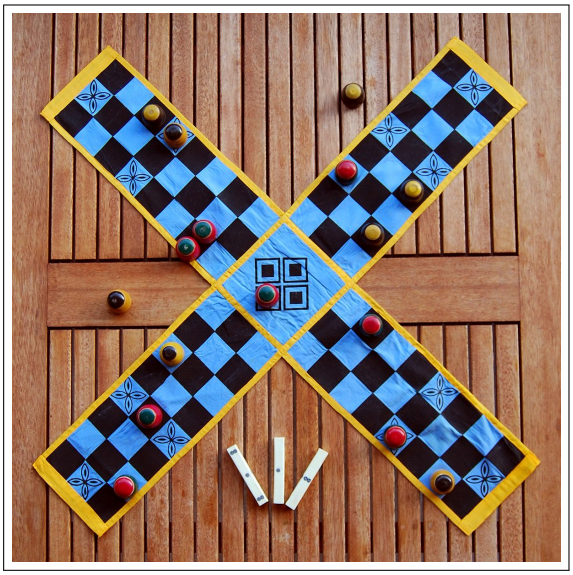

Fig. 1. Pachisi

***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, daßsie 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 Spielspaßfür 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 mühelose 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

- os unabhngig lauffhig - trotz development dauerhaft lauffhiger zustand (fr designer) - keine programmierkenntnisse, um neue spielfelder zu erzeugen und zu testen - erweiterbarkeit der regeln - erweiterbarkeit fr knstliche spieler - einfachste bedienung (nur maus + linke maustaste)

usage fr designer: entwickele spielfeld, starte generator, starte applikation, vorschlge an den programmierer zur weiterentwicklung usage fr programmierer: entwickele view erweitere regeln (controller: businesslogic, model: neue datenfelder) erweitere generator/grammatik nach implementierung designer neue version zur verfgung stellen

III. Implementierung

parallele entwicklung (2er team)

tech stack: java antlr beliebiger texteditor fr csv

projekte: Applikation (MenschAergerDichNicht) Generator (MADPlayfieldGenerator)

einfache und konfigurationslose Verknpfung (generierte Spielfelddatei: fester platz und name in projektstruktur (model GENPlayfieldCreator.java)

Klassen:

Controller:

Game (HauptController) hauptschleife

PlayerController (UnterController) kapselung fr player objekte

View: GUI -verantwortlich fr den Rahmen, fenstergre usw PlayfieldPanel - rendering des spielfeldes (tiles, pieces)

Model: Playfield - alle tiles Tile - knoteninformationen - referenzinformation ber pieces Player - enthlt seine pieces Ableitungen: KIPlayer/HumanPlayer

implementierungsvorteile durch knotenbasierte spielfeldelemente

-erzeugt eleganten regel code durch tiletypen und lookahead function -einfache erweiterung neuer tiletypen und regeln - natrlich 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

fr die abbildung eines brettspiels in digitaler form bietet sich das klassische mvc pattern an.

dabei ist zu beachten, das die dreickbeziehung zwischen mvc nicht symmetrisch (fhrt zu spagetticode, unklare verantwortlichkeiten) sondern hierarchisch implementiert wird. konkret: controller kennt model und view dh controller darf vernderungen im model vornehmen und bekommt events der view mitgeteilt view kennt model, darf aber nur lesen, um eigenen zusatnd upzudaten model kennt niemanden, stellt aber api fr lese/schreibzugriffe bereit

das spielfeld und der aktuelle spielzustand sind im model festgehalten, die view prsentiert das spielfeld und erwartet interaktion vom benutzer, der controller verarbeitet die eingaben des nutzers oder steuert die knstlichen spieler.

B. Singleton

Die Applikation soll stets ein lauffhiges Menschrgeredichnicht Spiel reprsentieren, welche ohne komplizierte Laufzeutinstellungen auskommt oder mit commandlineparametern zu starten sein mu (designer requirement). Es ist also nicht erwnscht, das mehrer Instanzen der Hauptklassen erzeugbar sind, da dies zu kompliziertem Initialisierungscode in den toplevelschichten fhren wrde (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 bersichtlichen Mainklasse, in der nur noch die Singletons instanziiert und miteinander verknpft werden. Smtliche Weiterentwicklung des Programms kann innerhalb der MVC-Komponenten erfolgen, ohne das die Main angepasst werden mu. 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 positionsunabhngig - spieleranzahl - ein startfeld pro spieler - gleiche anzahl an home und goal feldern pro spieler - hhe, breite der spielfeldmatrix - daten werden als getter im genierten code bereitgestellt

semantic parser - liest die einzelnen felder - verknpft die knoten - generiert den erzeugungscode des spielfeldes

VI. Fazit

knoten ansatz sehr gut geeignet fr erweiterung/generalisierung der spielfeldgenerierung einsatz passender pattern an den richtigen stellen fr weiterentwicklung der software (ki subclass, spielregeln)

grammatik zeilen/spalten gebunden (keine planar frei plazierbare tiles ohne x, y beschrnkung) aber mglich, tiles nur knotengebunden, dh zustzlicher editor fr graphisches design von csv-vorlagen mglich verschieben von graphischen elementen hat keinen einflu auf knotenverknpfungen

Acknowledgment

The authors would like to thank...

References

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