Fachbereich Informatik und Medien

Masterarbeit

Interaktive Evolution zur Assistenz bei der Einrichtungsplanung

vorgelegt von

Stephan Dreyer

an der Fachhochschule Brandenburg
im Fachbereich Informatik und Medien

Zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

Erstgutachter: Dipl.-Inform. Ingo Boersch
Zweitgutachter: Prof. Dr.-Ing. habil. Michael Syrjakow

Brandenburg, 9. August 2012

Zusammenfassung

Die Planung der Inneneinrichtung von Räumen ist oft aufwändig und zeitintensiv, da ein weites Spektrum verschiedenster Einrichtungsgegenstände verfügbar ist und diese fast beliebig kombiniert werden können. Diese Arbeit schlägt interaktive evolutionäre Algorithmen zur Assistenz bei der Einrichtungsplanung vor. Dazu werden zunächst die Bestandteile der interaktiven Evolution genauer definiert und Arten der interaktiven Selektion vorgestellt. Anschließend wird die Konzeption und Entwicklung des Softwaresystems dokumentiert. Zur Verringerung der Benutzerermüdung wird die Methode der Sparse Fitness Evaluation umgesetzt. Es werden Versuche mit der entwickelten Software durchgeführt und aufgetretene Probleme diskutiert. Abschließend gibt die Arbeit einen Ausblick für eine weitere Entwicklung.

Abstract

Planning interior spaces is often expensive and time-consuming, since a wide range of furnitures is available which can be combined almost arbitrarily. This thesis proposes interactive evolutionary computation for assistance in interior layout planning. For that, the components of interactive evolution are defined in detail and types of interactive selection are introduced. Following this, the design and development of the software system are documented. The method of sparse fitness evaluation is implemented to reduce user fatigue. Experiments with the developed software are carried out and problems encountered are discussed. In conclusion, the thesis gives prospects for further development.

Inhaltsverzeichnis

1. Einleitung

„Jedes Problem, das ich löste, wurde zu einer Regel, die später dazu diente, andere Probleme zu lösen.“

René Descartes (1596 - 1650)
französischer Philosoph, Mathematiker,
Naturwissenschaftler und Begründer des Rationalismus

1.1. Motivation

Für die Planung von Inneneinrichtungen werden derzeit zahlreiche Systeme angeboten. Diese funktionieren typischerweise wie eine Art Baukastensystem. Der Benutzer kann aus einer Menge von Einrichtungsgegenständen auswählen und diese frei im Raum platzieren. Dieser lösungsorientierte Ansatz zur Optimierung erfordert, dass der Benutzer sich bereits ein Ziel gesetzt hat. Er kann seine Kreativität frei entfalten und viele Randbedingungen selbst kontrollieren. Dies bedingt ein hohes Maß an Eigenleistung. Wenn Personen sich jedoch weniger mit den Einzelheiten der Inneneinrichtung beschäftigen möchten oder nicht die Zeit dazu haben, können sie gegen ein Entgelt einen Experten mit der Einrichtung der Wohnung beauftragen, typischerweise einen Innenarchitekten. Dieser findet mit Hilfe seiner eigenen Kreativität und Erfahrung eine Problemlösung, die die Wünsche des Kunden mit einbezieht. Wenn die Personen die durch einen Dienstleister verursachten zusätzlichen Kosten einsparen möchten, bleibt ihnen noch die Möglichkeit, sich selbst in Zeitschriften, Katalogen, Fachgeschäften oder im Internet über mögliche Kombinationen von Einrichtungsgegenständen zu informieren.

Für einen problemorientierten Ansatz zur Lösung von Optimierungsproblemen, bei dem subjektive Eindrücke wie Vorlieben, Intuition und Emotionen mit einbezogen werden, schlägt die Fachliteratur oft *evolutionäre Algorithmen* mit *interaktiver Fitness* vor. (vgl. [Tak01], [LKC01], [GG07], [GYM08], [KZTA05], [HT00]) Diese ermöglichen es dem Benutzer, zufriedenstellende Lösungen zu finden, indem Lösungskandidaten bewertet werden. Der evolutionäre Algorithmus führt die Optimierung gemeinsam mit dem Benutzer durch. Denkbar wäre, einen genetischen Algorithmus zur Optimierung und Hilfestellung bei der Einrichtung einer Wohnung zu verwenden.

Als Motivation wurde im Vorfeld eine Befragung im Internet durchgeführt. Es sollte ermittelt werden, wie die Akzeptanz einer Software zur Inneneinrichtungsplanung ist und was die häufigsten Gegenargumente zur Benutzung einer solchen Software sind.

Die Ergebnisse der Befragung in Anhang A lassen sich wie folgt interpretieren:

Benutzerschnittstelle
Eine einfache Bedienung wird gefordert. Viele Befragte gaben an, dass ihnen die Bedienung einer Planungssoftware zu kompliziert ist. Eine realitätsgetreue Darstellung ist ebenfalls eine häufig genannte Anforderung.

Kreativität
Das häufigste Argument gegen die Benutzung einer Inneneinrichtungsplanungssoftware war, dass die Befragten schon vorher konkrete Vorstellungen von ihrer Inneneinrichtung hatten. Möglicherweise kann jedoch eine problemorientierte Software die Kreativität dieser Benutzergruppe unterstützen.

Preis
Das wichtigste Kriterium bei der Auswahl der Möbel war der Preis. Die Befragten haben sich bei diesem Kriterium jedoch nur selten von anderen Personen beraten lassen. Bei der Benutzung einer Software zur Inneneinrichtungsplanung wurden nur selten preisliche Vorstellungen umgesetzt. Hier könnte eine problemorientierte Software verschiedene Vorschläge machen, wobei eine Preisgrenze berücksichtigt wird.

1.2. Ziel der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung einer problemorientierten Anwendung zur Inneneinrichtungsplanung. Zur Zielgruppe der Anwendung gehören Benutzer, die sich bei einem möglichst geringen zeitlichen Aufwand durch Einrichtungsvorschläge bei der Zusammenstellung einer Inneneinrichtung helfen lassen möchten, ohne Geld für einen Experten auszugeben. Die Anwendung soll Einrichtungsgegenstände aus einer Datenbank zu Einrichtungsplänen zusammenstellen. Der Benutzer kann durch eine Bewertung der Einrichtungspläne dem System seine subjektiven Eindrücke mitteilen. Die Art, Anzahl, Position und Ausrichtung der Einrichtungsgegenstände sollen nach den Präferenzen des Benutzers optimiert werden. Dazu sollen interaktive evolutionäre Algorithmen verwendet werden. Der Benutzer soll in die Evolution mit einbezogen, jedoch nicht überfordert werden. Aus diesem Grund ist zu untersuchen, wie der Benutzer in seinen Aufgaben möglichst wenig belastet wird. Dazu gehört auch ein geringer zeitlicher Aufwand. Außerdem müssen die Freiheitsgrade des evolutionären Algorithmus festgelegt werden. Es ist zu untersuchen, welche Einschränkungen der Benutzer festlegen darf und welche durch den evolutionären Algorithmus evolviert werden können.

1.2.1. Grafische Darstellung

Die Darstellung von Individuen sollte sich auf das mögliche Minimum beschränken. Sie sollte jedoch alle Informationen beinhalten, die für den Benutzer relevant sind. In diesem konkreten Anwendungsfall sollen Zusammenstellungen von Einrichtungsgegenständen angezeigt werden. Daher bietet sich eine dreidimensionale Darstellung an, die der natürlichen visuellen Wahrnehmung des Benutzers ähnlich ist. Die Perspektive sollte vom Benutzer verändert werden können.

1.2.2. Eingabe

Die Eingabeschnittstelle ist so zu gestalten, dass die Bewertung mit möglichst wenigen Interaktionen durchgeführt werden kann. Es soll eine Eingabe per Maus oder Touchscreen möglich sein. Die Schnittstelle soll auch einen Wechsel der Perspektive erlauben.

1.3. Aufbau der Arbeit

Kapitel 2 bietet eine Einführung in die theoretischen Grundlagen evolutionärer und genetischer Algorithmen. Die interaktive Evolution, deren Problemstellungen und Anwendungsgebiete sowie die Methode der *Sparse Fitness Evaluation* werden vorgestellt. In Kapitel 3 werden Ziele und Anforderungen des zu entwickelnden evolutionären Algorithmus definiert. Anschließend wird ein Konzept für den evolutionären Algorithmus und dessen Benutzerschnittstelle unter Berücksichtigung der Anforderungen erarbeitet und Möglichkeiten zur Optimierung erläutert. In Kapitel 4 ist die Realisierung der Architektur und Implementierung des gesamten Systems dokumentiert. Dort sind auch einfache Beispiele zur Umsetzung eines evolutionären Algorithmus zu finden. Anschließend werden Hilfsprogramme und die Benutzeroberfläche vorgestellt. Die Versuche und der Funktionsnachweis zur Clusterbildung sowie zur automatischen und interaktiven Evaluation werden in Kapitel 5 erläutert und ausgewertet. Zuletzt werden die Ergebnisse der Arbeit in Kapitel 6 zusammengefasst und ein Ausblick für weiterführende Arbeiten gegeben.

2. Theoretische Grundlagen

In diesem Kapitel werden evolutionäre Algorithmen vorgestellt, wobei die interaktive Bewertung von Lösungen durch den Benutzer im Vordergrund steht. Die damit verbundenen Probleme werden näher benannt, Anwendungsgebiete aufgezeigt und die Methode der *Sparse Fitness Evaluation* vorgestellt.

2.1. Evolutionäre Algorithmen

Die Evolution ist ein Konzept, welches in der Natur bei allen Lebewesen vorzufinden ist. Sie optimiert vererbbare Merkmale, zu denen der physische Aufbau und das Verhalten zählen, durch zufällige genetische Veränderungen. Diese Veränderungen beeinflussen die Reproduktionswahrscheinlichkeit der jeweiligen Lösung. Wenn diese begünstigt wird, so hat dieses Lebewesen eine höhere evolutionäre Fitness als andere Lebewesen, die sich mit einer geringeren Rate reproduzieren. Diese Eigenschaften betreffen meist die Anpassung an die Umwelt, die Fähigkeit sich ernähren zu können und von Feinden nicht gefressen zu werden, bevor die Fortpflanzung stattfindet. Manchmal erschließen sich durch zufällige genetische Veränderungen auch neue Nahrungsressourcen, die dem Lebewesen einen Vorteil gegenüber anderen geben. Charles Darwin beschrieb dies mit dem Ausdruck „Survival of the Fittest“[1] (vgl. [Dar09]). Nur ausreichend angepasste Lebewesen überleben. Der Evolution liegt kein Determinismus zu Grunde und es gibt kein Ziel. Nach der Evolutionstheorie von Darwin sind folgende Bedingungen für das Auftreten von Evolution erforderlich:

- Individuen einer Population können sich reproduzieren
- Es treten Veränderungen auf, die die Reproduktionsrate beeinflussen
- Die Veränderungen können vererbt werden

[1] Der Ausdruck „Survival of the Fittest“ stammt ursprünglich vom Philosophen und Soziologen Herbert Spencer und wurde von Charles Darwin übernommen.

- Die Ressourcen sind begrenzt

Evolutionäre Algorithmen sind eine Klasse von Optimierungsverfahren, die das evolutionäre Prinzip in abstrakter Form auf technische Systeme übertragen. Sie stellen eine probabilistische Suche im Lösungsraum dar. Vier Unterarten der evolutionären Algorithmen haben sich durchgesetzt:

- *Evolutionäres Programmieren* (EP)
- *Evolutionsstrategien* (ES)
- *Genetische Algorithmen* (GA)
- *Genetisches Programmieren* (GP)

Detaillierte Informationen über diese vier Verfahren sind in [BHS97] zu finden.

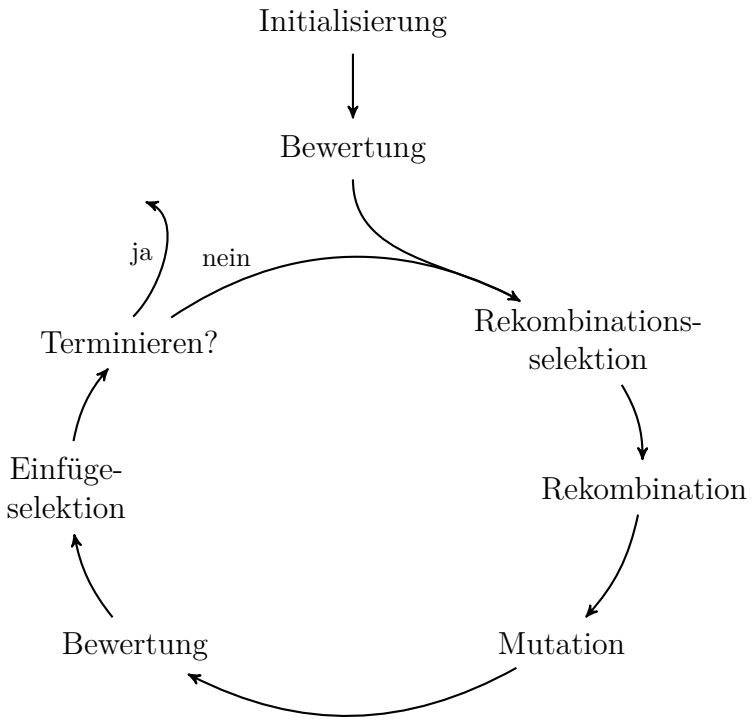


Abbildung 2.1.: Evolutionärer Zyklus (nach [Wei02] S. 43)

Abbildung 2.1 zeigt den Zyklus bei evolutionären Algorithmen. Bei der Initialisierung wird die Population zunächst mit Individuen gefüllt. Diese können zufällig erzeugt werden oder durch das Problem bereits vorgegeben sein. Anschließend wird eine erste Bewertung aller Individuen durchgeführt. Nun

werden Individuen zur Reproduktion auf Basis der Bewertungen selektiert, rekombiniert, mutiert, bewertet und wieder in die Population eingefügt. Zum Einfügen in eine bestehende Population kann es erforderlich sein, Individuen zu selektieren, die ersetzt werden sollen. Ist die Bedingung zum Terminieren nicht erfüllt und die Optimierung noch nicht beendet, startet ein weiterer evolutionärer Zyklus.

2.2. Genetische Algorithmen

Genetische Algorithmen wurden 1975 vom US-amerikanischen Informatiker John Holland entwickelt [Hol75]. Die folgenden verwendeten Begriffe wurden der biologischen Evolutionstheorie entlehnt, haben jedoch eine eigene Bedeutung. Eine Problemlösung wird durch ein *Individuum* repräsentiert. Dieses besteht aus *Genotyp* und *Phänotyp*. Der Genotyp ist die Gesamtheit aller vererbbaren Informationen, also die kodierten Informationen des Individuums. Die Ausprägung dieser genetischen Eigenschaften wird als Phänotyp bezeichnet. Dieser wird durch eine Dekodierung der Erbinformationen erzeugt. Die Kodierung der genetischen Informationen wird als *Chromosom* bezeichnet. Die kleinste Einheit der genetischen Information ist das *Gen*. Die verschiedenen möglichen Werte eines Gens heißen *Allele*. (vgl. [Wei02] S. 27)

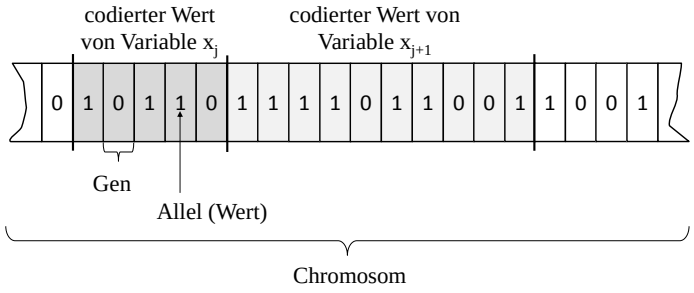


Abbildung 2.2.: Chromosom bei genetischen Algorithmen (nach [Nis97])

Abbildung 2.2 verdeutlicht diese Zusammenhänge am Beispiel eines Chromosoms in Form eines Bitstrings, der in klassischen genetischen Algorithmen verwendet wird. Dieser hat eine feste Länge. Die Gene sind die jeweiligen Bits, die Allele für jedes Gen sind die Werte 0 und 1. Je nach Anforderungen können

jedoch auch beispielsweise reelle oder ganzzahlige Datentypen zur Kodierung verwendet werden.

Zur Rekombination und Variation sind *genetische Operatoren* erforderlich. Dabei wird zwischen zwei Arten unterschieden. Die Mutation ist ein Operator, der ein oder mehrere Gene um einen Zufallswert verändert. Die maximale Veränderung der Mutation heißt *Schrittweite*. Die Mutation erfolgt mit einer bestimmten Wahrscheinlichkeit p_{mut}. Das Crossover ist ein Operator, der aus zwei Elternindividuen ein oder mehrere Kinder erzeugt. Dabei werden die Eigenschaften beider Eltern kombiniert. (vgl. [GKK04] S. 40f)

Die *Fitness* eines Individuums ist ein Hilfsmittel zur Selektion und spiegelt die Güte der darin kodierten Problemlösung wider. Individuen mit einer guten Fitness dürfen sich mit einer hohen Wahrscheinlichkeit reproduzieren. Die Fitness wird oft durch eine positive reelle Zahl oder einen Rang in der Population beschrieben. Die Fitnessfunktion in Formel 2.1 ist eine Abbildung des phänotypischen Raums S_{phaeno} auf einen reellen Fitnesswert. Dazu wird der Phänotyp des jeweiligen Individuums einer Bewertung unterzogen.

$$fit : S_{phaeno} \rightarrow [0, \infty) \tag{2.1}$$

Durch die Existenz einer Fitnessfunktion können zwei Individuen algorithmisch verglichen werden. So repräsentiert das Chromosom c_1 genau dann eine bessere Lösung als das Chromosom c_2, wenn gilt $fit(c_1) > fit(c_2)$ (vgl. [GKK04] S. 37f).

Eine weitere Form der Fitness ist die implizite Fitness. Wenn Individuen beispielsweise in einem künstlichen Ökosystem konkurrieren, kann die Fitness nicht immer durch eine skalare Größe beschrieben werden (vgl. [BHS07] S. 71). In manchen Fällen kann auch eine Dominanzordnung bestehen, ohne dass konkrete Fitnesswerte verglichen werden. In diesem Fall ist c_1 genau dann eine bessere Lösung als c_2, wenn gilt $f_{dom}(c1, c2) > 0$.

Die Auswahl von Individuen zur Reproduktion wird als *Selektion* bezeichnet. Sie bedeutet eine Verschiebung der Häufigkeit von Individuen anhand ihrer Fitness. Die Selektion kann deterministisch oder probabilistisch erfolgen (vgl. [Wei02] S. 66f).

Die enge Definition von genetischen Algorithmen sieht als Genotyp einen Bitstring fester Länge vor. John Holland beschrieb diese erstmalig mit dem *kanonischen genetischen Algorithmus*, kurz *kGA*. Aufgrund der begrenzten Rechen-

geschwindigkeit der damaligen Computer wurde auf einen geringen Speicherbedarf und eine schnelle Verarbeitung durch Bitoperationen Wert gelegt. Moderne genetischen Algorithmen unterliegen kaum noch solchen Beschränkungen, da heutige Computer Gleitkommaoperationen sehr schnell ausführen können und einen ausreichend großen Arbeitsspeicher haben. Die weite Definition von genetischen Algorithmen ermöglicht auch eine Kodierung mit reellen Zahlen. Die Anzahl der Gene kann variabel sein.

2.3. Interaktive Evolution

Einige Optimierungsaufgaben erfordern eine Evaluation durch den Benutzer, wenn keine mathematische Funktion zur Bewertung der Problemlösungen definiert werden kann und die Güte der Lösungen auf subjektiven Eindrücken wie Vorlieben, Emotionen und der Intuition des Menschen beruht (vgl. [Tak01]). Des Weiteren kann in einigen Fällen die Konvergenz eines evolutionären Algorithmus erhöht werden, indem der Benutzer sein Fachwissen in die Bewertung mit einbringt (vgl. [KZTA05]). Zur Abgrenzung der interaktiven Evolution wurden zwei Definitionen entwickelt. (vgl. [Tak01])

Enge Definition
„Ein interaktiver evolutionärer Algorithmus ist eine Methode, bei der ein evolutionärer Algorithmus auf Basis von Fitnesswerten, die durch einen Menschen zugewiesen wurden, ein Zielsystem optimiert.“ (vgl. [Tak01])

Weite Definition
„Ein interaktiver evolutionärer Algorithmus ist eine Methode, bei der ein evolutionärer Algorithmus mit einer Mensch-Maschinen-Schnittstelle ein Ziel-System optimiert.“ (vgl. [Tak01])

Nach der engen Definition ersetzt der Benutzer lediglich die sonst berechenbare Fitnessfunktion, indem er Lösungskandidaten bewertet und diesen einen Fitnesswert zuordnet. Sie kann zusammengefasst als „evolutionäre Optimierung mit interaktiver Fitness“ beschrieben werden. Die weite Definition hingegen erlaubt dem Benutzer, an jedem Teil des evolutionären Optimierungsprozesses mitzuwirken.

Eigene Definition
In dieser Arbeit soll ein abgegrenzter Bereich der interaktiven Evolution be-

trachtet werden. Der Benutzer soll an der Selektion von Individuen aktiv teilnehmen. Die Art der Selektion ist nicht weiter eingegrenzt und kann problemangepasst erfolgen. Dazu wird eine enge Definition eingeführt, bei der die Selektion allgemein betrachtet wird: „Ein interaktiver evolutionärer Algorithmus ist eine Methode, bei der ein evolutionärer Algorithmus ein Ziel-System optimiert, dessen Selektion durch einen Menschen unter Verwendung einer Mensch-Maschinen-Schnittstelle durchgeführt wird.“ Der weitere Verlauf dieser Arbeit bezieht sich auf diese Definition. Abbildung 2.3 bietet eine Übersicht der verschiedenen Definitionen.

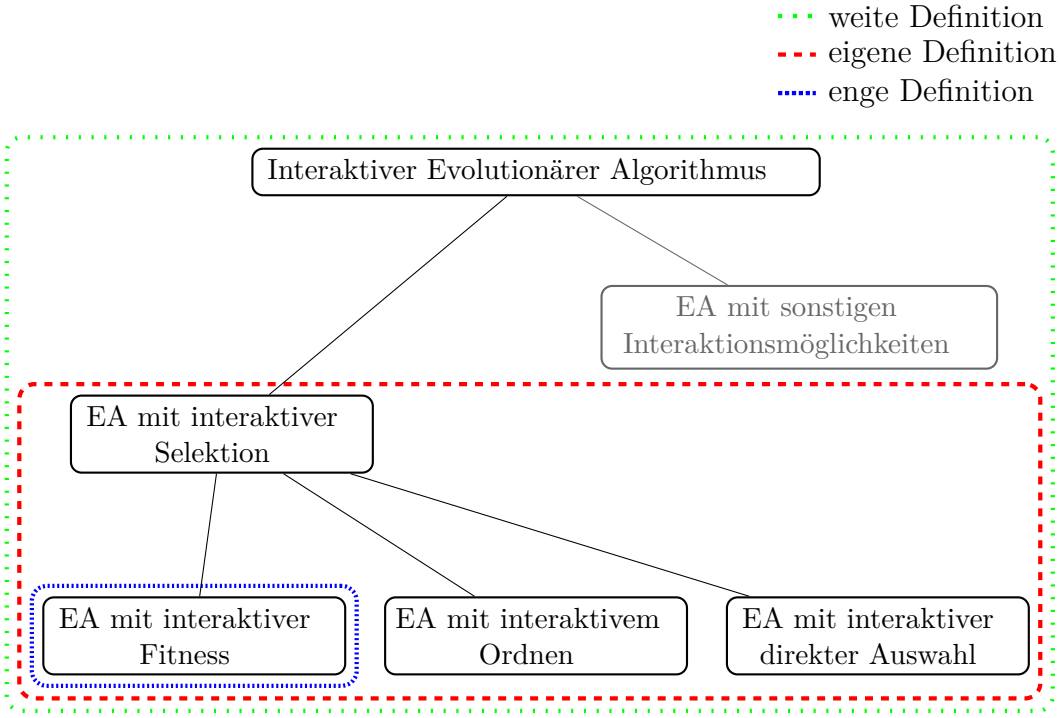


Abbildung 2.3.: Definitionen interaktiver Evolution

Der Benutzer muss die für die Optimierungsaufgabe relevanten Informationen wahrnehmen können, wozu eine optische oder akustische Darstellung der Phänotypen erforderlich ist. Ebenso muss der Benutzer dem System seine subjektiven Eindrücke mitteilen können. Dazu ist eine Schnittstelle zur Eingabe notwendig.

Im weiteren Verlauf dieser Arbeit kann hinsichtlich der Selektion zwischen zwei Arten evolutionärer Algorithmen unterschieden werden (vgl. [Dre11] S. 6):

- Ein *evolutionärer Algorithmus* (im Folgenden mit dem Akronym *EA* bezeichnet) führt die Selektion mit Hilfe einer maschinell berechenbaren Fitnessfunktion durch.
- Ein *interaktiver evolutionärer Algorithmus* (im Folgenden mit dem Akronym *IEA* bezeichnet) erfordert eine Selektion durch den Benutzer.

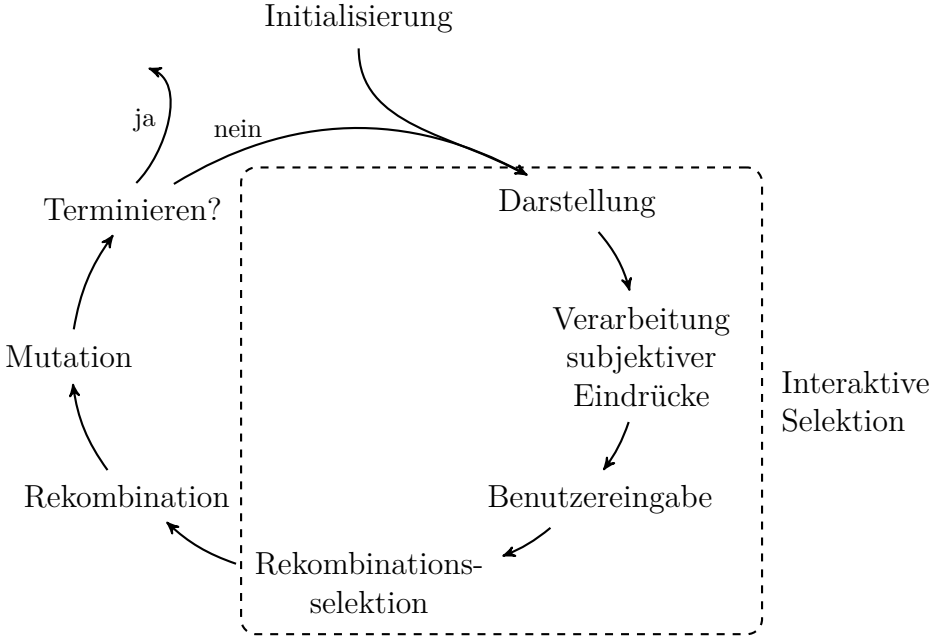


Abbildung 2.4.: Interaktiver evolutionärer Zyklus

Der evolutionäre Zyklus von interaktiven evolutionären Algorithmen ist in Abbildung 2.4 dargestellt. Dieser wurde gegenüber dem evolutionären Zyklus aus [Wei02] in Abbildung 2.1 vereinfacht. Er enthält keine Einfügeselektion, da diese nicht bei allen evolutionären Algorithmen notwendig ist. Die Bewertung wird beim interaktiven evolutionären Algorithmus durch die interaktive Selektion ersetzt. Diese beinhaltet die Darstellung, die Verarbeitung subjektiver Eindrücke durch den Benutzer und die anschließende Eingabe der Präferenzen.

2.3.1. Interaktive Selektion

Während bei den automatisch ablaufenden evolutionären Algorithmen eine Fitnessfunktion zur Berechnung der Güte von Individuen verwendet wird, gibt es bei der interaktiven Evolution mehrere Verfahren zur Selektion von Individuen. Abbildung 2.5 bietet eine Übersicht, wie aus einer Individuenmenge I^m die Individuenmenge I^n bestimmt werden kann. I^m ist der Teil der Population, der zu dem jeweiligen Zeitpunkt dargestellt wird. I^n ist eine Teilmenge von I^m und umfasst die Individuen aus I^m, die sich reproduzieren dürfen. Es existieren folgende Arten der interaktiven Selektion:

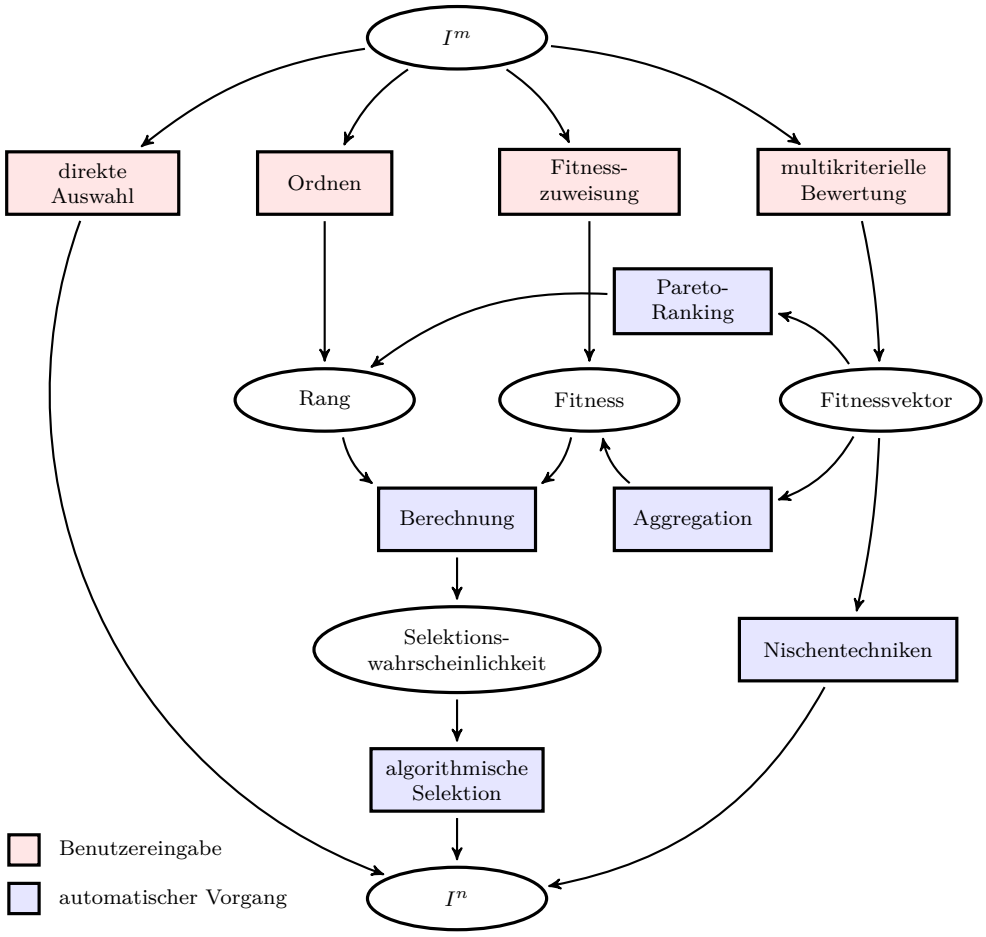


Abbildung 2.5.: Arten der interaktiven Selektion

- Bei der **direkten Auswahl** wählt der Benutzer die Individuen aus, die er als gute Lösungen erachtet. Die Reproduktionswahrscheinlichkeit der gewählten Lösungskandidaten hat den Wert eins, während sie bei den nicht ausgewählten Individuen den Wert null hat. Dies ist die minimale Anzahl möglicher Bewertungsabstufungen. Das Verfahren wird auch

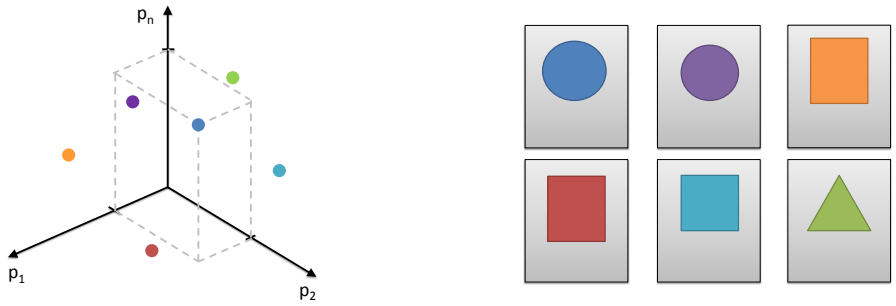
als „simulated breeding“, auf Deutsch „simulierte Züchtung“ bezeichnet (vgl. [Tak01] S. 1276). Eine anschließende algorithmische Rekombinationsselektion wie in Abbildung 2.4 ist hier nicht erforderlich.

- Bei der **Fitnesszuweisung** ersetzt der Benutzer die Fitnessfunktion und weist jedem Lösungskandidaten eine reelle Zahl als Fitnesswert zu. Die Selektionswahrscheinlichkeiten werden aus den Fitnesswerten berechnet. Anschließend erfolgt eine algorithmische Selektion.
- Beim **Ordnen** werden keine reellen Fitnesswerte verwendet. Der Benutzer ordnet die Individuen anhand seiner subjektiven Eindrücke und erstellt so eine Rangfolge. Anschließend wird eine rangbasierte Selektion durchgeführt. Die Selektionswahrscheinlichkeit jedes Individuums ist abhängig von dessen Rang in der Population.
- Bei der **multikriteriellen Bewertung** evaluiert der Benutzer einzelne Kriterien des Individuums. Das Resultat ist ein Fitnessvektor, der für jedes Kriterium einen Fitnesswert enthält. Zur Selektion auf Basis dieses Vektors schlägt die Literatur verschiedene Techniken vor. Durch eine *Aggregation* kann mittels einer Aggregationsfunktion die Gesamtfitness des Individuums berechnet werden. Alternativ kann auch ein *Pareto-basiertes Ranking* durchgeführt werden, wodurch eine Rangfolge entsteht. Zur Mehrzieloptimierung werden auch oft *Nischentechniken* zur Selektion von Individuen eingesetzt. (vgl. [FF94]) Neben diesen drei Techniken gibt es noch viele weitere Ansätze, die an dieser Stelle nicht erschöpfend behandelt werden können.

2.3.2. Darstellung

Die Darstellung der Phänotypen ermöglicht dem Benutzer, die spezifischen Eigenschaften der Individuen wahrzunehmen. Je nach Populationsgröße und Art der Benutzerschnittstelle kann die gesamte Population oder nur ein Teil davon optisch dargestellt werden. Dies beeinflusst die subjektiven Eindrücke des Benutzers. Er bewertet ein Individuum oft relativ zu dem, was er gerade sieht, oder kurz zuvor gesehen und im Gedächtnis behalten hat.

Abbildung 2.6(a) zeigt die Individuen im Parameterraum. Die Anzahl der Dimensionen dieses Raums entspricht der Anzahl der durch die Individuen ko-



(a) Individuen im Parameterraum S_{para}

(b) Darstellung einer Menge von Phänotypen

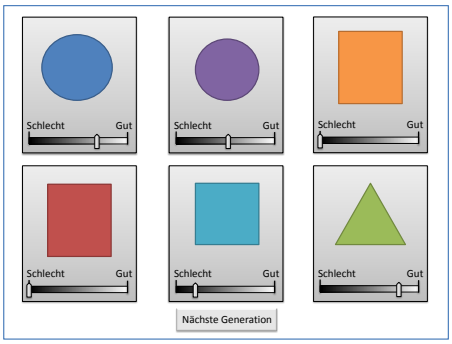
Abbildung 2.6.: Genotypen sowie Darstellung der Phänotypen (nach [Dre11] S. 5)

dierten Eigenschaften. Zur Darstellung (Abbildung 2.6(b)) müssen aus den Genotypen zunächst mittels einer Dekodierungsfunktion $f_{dek} : S_{para} \rightarrow S_{phaeno}$ Phänotypen erzeugt werden.

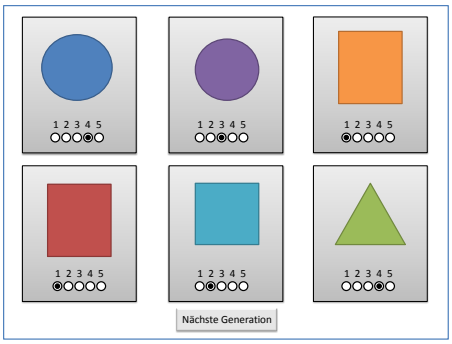
Die Darstellung von Individuen sollte auf das Problem angepasst sein. Einige Probleme erfordern eine dreidimensionale graphische Darstellung, beispielsweise bei der architektonischen Raumplanung (vgl. [IT09]) oder der Beleuchtungsabstimmung in der Computergrafik (vgl. [HT00]). Bei anderen Problemen ist eine zweidimensionale grafische Darstellung angebracht, beispielsweise bei der Bildverarbeitung in der Medizin (vgl. [PC97]) oder der Generierung von Phantombildern (vgl. [Tak00]). Die Darstellung von Individuen muss jedoch nicht zwangsweise grafisch erfolgen. Eine akustische Darstellung, zum Beispiel bei der Sprachsynthese oder Hörgeräteanpassung (vgl. [Sat02], [TO07]) ist ebenfalls möglich. Des Weiteren ist auch eine textuelle Darstellung denkbar, beispielsweise wenn Texte evolutionär erzeugt werden sollen.

2.3.3. Benutzerschnittstelle

Zur interaktiven Selektion muss dem Benutzer eine Schnittstelle zur Verfügung gestellt werden, mit der er seine subjektiven Eindrücke dem System mitteilen kann. Meistens werden interaktive evolutionäre Algorithmen so umgesetzt, dass der Benutzer seine Eingaben per Maus oder Tastatur durchführen kann.



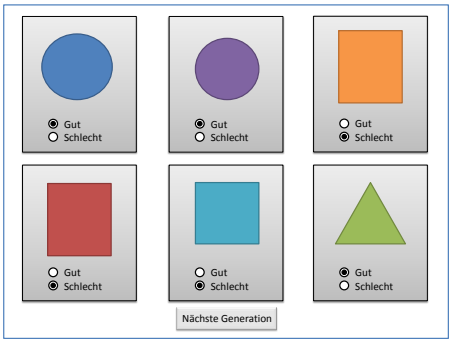
(a) Eingabe reeller Fitnesswerte per Schieberegler



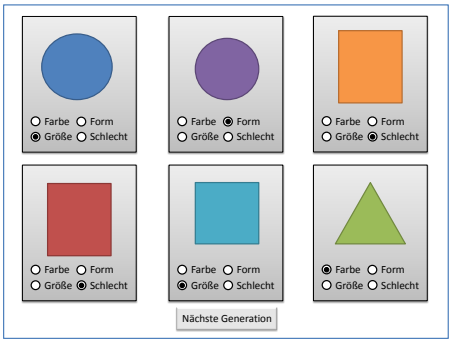
(b) Eingabe integraler Fitnesswerte

Abbildung 2.7.: Explizite Zuweisung von Fitnesswerten mit dem Kachelmodell (aus [Dre11])

Abbildung 2.7 zeigt zwei Schnittstellen zur Fitnesszuweisung. Beide sind Varianten des Kachelmodells (vgl. [Dre11] S. 23f). Zur Eingabe reeller Werte bieten sich beispielsweise Schieberegler wie in Abbildung 2.7(a) an. Der Benutzer kann mit diesen beinahe stufenlos Fitnesswerte eingeben. Die Abstufung der Bewertungsschritte ist nur durch die Pixelgröße des Schiebereglers oder programmatische Beschränkungen begrenzt. Die Eingabe ganzzahliger Fitnesswerte kann mit einer Schnittstelle wie in Abbildung 2.7(b) erfolgen. Hier ist die Anzahl der Bewertungsschritte vom Programm vorgegeben. In der Abbildung sind fünf Schritte zu sehen, wodurch für jedes Individuum Fitnesswerte von eins bis fünf vergeben werden können.



(a) Eingabe von Gefallen oder Abneigung

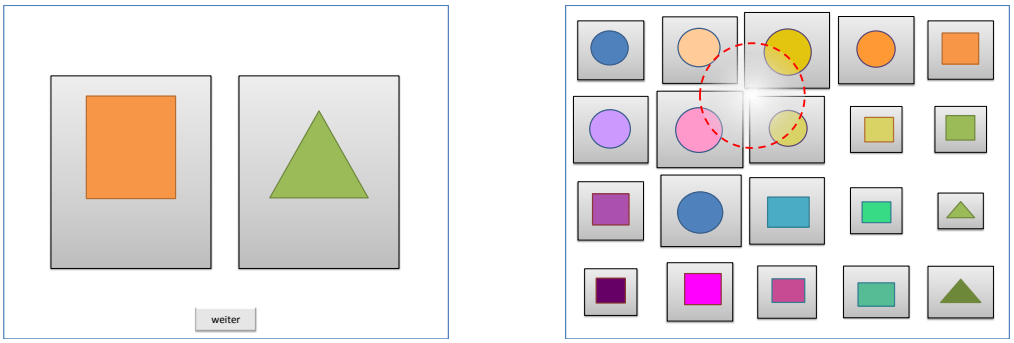


(b) Auswahl des besten Kriteriums

Abbildung 2.8.: Selektion ohne explizite Fitnesszuweisung (aus [Dre11])

Die Eingabeschnittstelle für Gefallen und Abneigung in Abbildung 2.8(a) kann für die direkte Auswahl von Individuen verwendet werden. In diesem Fall würden nur die mit „Gut“ bewerteten Individuen zur Rekombination verwendet

werden. Es ist jedoch auch möglich, diese Individuen zweistufig zu bewerten und anschließend eine algorithmische Selektion durchzuführen. Die mit „Schlecht“ bewerteten Individuen hätten dadurch ebenfalls eine, wenn auch geringere Chance zur Reproduktion. Mit der Benutzerschnittstelle aus Abbildung 2.8(b) kann der Benutzer das Kriterium angeben, welches seiner Meinung nach den anderen Kriterien überlegen ist. Ist der subjektive Eindruck des Benutzers beispielsweise „Die Farbe dieses Kreises gefällt mir sehr gut“, so kann er das Attribut „Farbe“ auswählen. Wenn ihm kein Kriterium gefällt, kann er das gesamte Individuum mit „Schlecht“ bewerten. Nach der Eingabe muss eine algorithmische Selektion auf Basis der Kriterienbewertungen erfolgen.



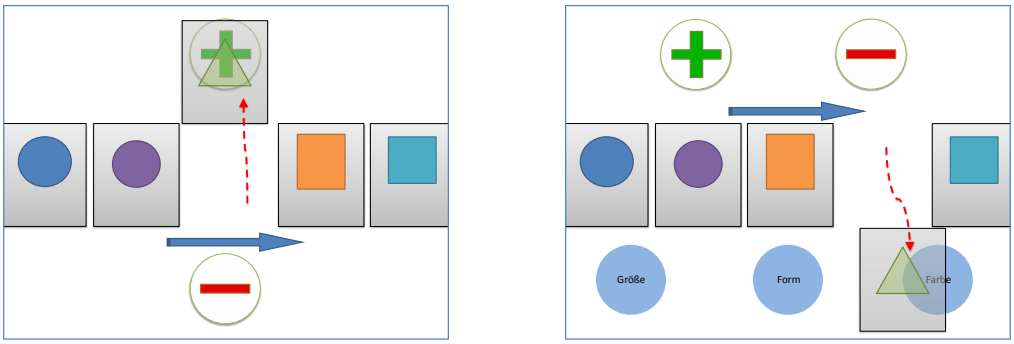
(a) Auswahl des besseren Individuums

(b) Indirekte Bewertung beim Schauermodell

Abbildung 2.9.: Selektion ohne explizite Fitnesszuweisung (aus [Dre11])

In Abbildung 2.9(a) ist eine Eingabeschnittstelle dargestellt, die eine Auswahl des besseren Individuums ermöglicht. So können die Individuen vom Benutzer geordnet werden. Das ausgewählte Individuum bekommt einen höheren Rang als jenes, welches nicht ausgewählt wurde. Die Benutzerschnittstelle eignet sich aber auch zur direkten Auswahl. In diesem Fall darf sich nur das ausgewählte Individuum reproduzieren. Das in Abbildung 2.9(b) dargestellte *Schauermodell* aus [Fis08] ermöglicht eine indirekte Fitnessbewertung. Der Benutzer kann hier bestimmte Bereiche bevorzugen, indem er den Schauer über diese verschiebt. Die Individuen besitzen zwar einen Fitnesswert, jedoch wird dieser nicht direkt vom Benutzer eingegeben, sondern in Abhängigkeit des Abstands zum Schauer zugewiesen. Als Schauer wird hier ein lokaler Bereich mit einer hohen Fitness bezeichnet, ähnlich einem Regenschauer, der lokal das Wachstum von Pflanzen fördert.

Die beiden Benutzerschnittstellen mit dem Strom-Modell in Abbildung 2.10 wurden ebenfalls in [Fis08] entwickelt. Die Bilder der Individuen fahren mit



(a) Dreistufige Bewertung (b) Bewertung nach Kriterien

Abbildung 2.10.: Varianten des Strom-Modells (aus [Dre11])

einer definierten Geschwindigkeit wie auf einem Laufband von links nach rechts über den Bildschirm. Der Benutzer kann diese bewerten, indem er sie auf das entsprechende Feld zieht. Beim Drei-Klassen-Modell in Abbildung 2.10(a) kann ein Bild auf der Pluszone zur positiven Bewertung abgelegt werden. Die Fitness wird damit erhöht. Ebenso kann ein Bild auf die Minuszone gezogen werden, wodurch die Fitness verringert wird. Verlässt ein Bild rechts den Bildschirm, ohne dass der Benutzer eingegriffen hat, so wird das Individuum neutral bewertet. Abbildung 2.10(b) zeigt das Strom-Modell mit Attributzonen. Hier kann der Benutzer zusätzlich einzelne Attribute positiv bewerten, indem das Bild auf der entsprechenden Zone abgelegt wird. (vgl. [Dre11] S. 26f)

2.4. Probleme bei der interaktiven Selektion

Die Teilnahme des Benutzers am evolutionären Prozess ist bei *IEA* notwendig. Ohne den Benutzer findet keine Bewertung und damit keine Evolution statt. Die Anzahl der gleichzeitig präsentierten Individuen sollte nicht zu hoch sein. Bei zeitkontinuierlichen Medien wie Audio oder Video ist die Populationsgröße durch die Erinnerungskapazität des Menschen begrenzt. Die Anzahl der durch den Benutzer zu evaluierenden Individuen sollte ebenfalls gering gehalten werden, da der Benutzer sonst zu schnell ermüdet und die Bewertung weniger gewissenhaft durchführen würde. Untersuchungen aus [Tak01] haben ergeben, dass eine Generationsanzahl zwischen 10 und 20 sinnvoll ist, wenn jeweils eine Generation dem Benutzer in Form von Kacheln präsentiert wird. Die Zeit, die der Benutzer mit der Evaluation verbringt und die Anzahl der notwendigen Interaktionen sollte ebenfalls gering gehalten werden. Laut den Angaben von [GYY09] ist eine Evaluationsdauer von unter zehn Minuten akzeptabel. Versuche in [LSA+06] haben ergeben, dass die Benutzer bei einer Dauer von über einer Stunde frustriert und erschöpft waren (vgl. [Dre11] S. 10). Ein guter interaktiver evolutionärer Algorithmus ist also ein Kompromiss aus einer geringen Benutzerermüdung, einer großen evolutionären Vielfalt und der damit verbundenen Leistung des Algorithmus.

In [Dre11] wurde untersucht, wie ein interaktiver evolutionärer Algorithmus entworfen werden kann, um einer potenziellen Benutzerermüdung entgegen zu wirken. Im Folgenden ein Auszug der entwickelten Lösungskonzepte:

- In [Tak01] wird eine einfach zu bedienende Benutzerschnittstelle gefordert, die den Benutzer nicht mit Einstellungsmöglichkeiten überfordert. Es sollten fünf bis sieben Stufen zur Bewertung von Individuen angeboten werden. Bei mehr Stufen kann ein Mensch die Bewertung kaum noch differenziert durchführen. Dadurch entsteht ein sogenanntes Quantisierungsrauschen (vgl. [Tak01]). Der interaktive Evolutionsprozess soll vom Benutzer nicht als unangenehme Aufgabe, sondern als spannende, kurzweilige, gemeinsame Entwicklung guter Lösungen empfunden werden.
- Durch den Einsatz eines Cluster-Verfahrens kann die Anzahl der dem Benutzer präsentierten Individuen reduziert werden. Als Umkehrschluss kann die Populationsgröße erhöht werden, während die Anzahl der zu bewertenden Individuen gleich bleibt. Eine größere Population bedeutet

eine erhöhte *Diversität* und damit eine höhere Wahrscheinlichkeit, dass die gesuchte Lösung oder deren Allele in der Population enthalten sind.

- Zur Optimierung der Parameter eines interaktiven evolutionären Algorithmus könnte wiederum ein genetischer Algorithmus eingesetzt werden. Dadurch kann die *Konvergenz* des *IEA* durch Erfahrungen bisheriger Evaluationsdurchgänge erhöht werden.

2.4.1. Populationsgröße

Die Größe der Population beeinflusst die Konvergenz des Algorithmus maßgeblich. Bei einer geringen Populationsgröße besteht eine kleinere Wahrscheinlichkeit, dass zu einem beliebigen Zeitpunkt eine gute Lösung bereits in der Population enthalten ist. Die Diversität ist gering und ein vorzeitiges Konvergieren wahrscheinlich. Dabei werden Allele einzelner Gene zu früh aus der Population entfernt und eventuell bessere Lösungen können nicht mehr allein durch Crossover entstehen. Dieser Effekt wird auch als *Gendrift* bezeichnet (vgl. [Wei02] S. 31). Vorteilhaft ist an einer kleinen Population, dass der Benutzer weniger Zeit zur Evaluation aller Individuen benötigt, als es bei einer großen Population der Fall ist.

2.5. Anwendungsgebiete interaktiver evolutionärer Algorithmen

Interaktive evolutionäre Algorithmen wurden in vielen Gebieten wie Design, Wissenschaft, Kunst und Technik eingesetzt. Tabelle 2.1 fasst die Anwendungsbeispiele aus [Dre11] zusammen.

Domäne	**Anwendungsfall**	**Ziel**	**Art des Algorithmus**
Design	Modedesign	Zusammenstellung von Kleidungsstücken	A
	Redaktionelles Design	Entwurf von Postern unter Beeinflussung von Layout, Schriftgröße und Farbe	A
	Mikroelektronische Systeme	Entwurf von Inertialsensoren und RF-Filtern	B
	Raumplanung in der Gebäudearchitektur	Entwurf von Grundrissen von Räumen unter Einbeziehung von Benutzervorgaben und Regeln	A
Kunst	Graphische Kunst und Computeranimation	Generierung von Fraktalen	A
	Beleuchtung in der Computergrafik	Optimierung der Art, Intensität und Position von Lichtquellen im dreidimensionalen Raum	A
	Filter für die Bildverarbeitung	Generierung von Bildverarbeitungsfiltern mittels einer Filtersprache	A
	Musikkomposition	Erzeugung kurzer rhythmischer Muster mit einem GA und anschließende zeitliche Anordnung dieser mittels GP	A
Technik	Gesichtsbilder	Generierung von Gesichtsbildern unter Beeinflussung der Parameter Gesichtsform, Haare, Augenbrauen, Augen, Nase, Mund	A
	Sprachverarbeitung und Sprachsynthese	Generierung von Audiofiltern zur Verbesserung von Sprachaufnahmen	A
	Hörgeräteanpassung	Optimierung der Parameter von Hörgeräten	A
	Abfrage von Mediendatenbanken	Suche von Bildern in Datenbanken anhand textueller Beschreibungen	A
	Bildverarbeitung in der Medizin	Optimierung grafischer Filter zur Verbesserung von MRT-Bildern	A
	Steuerung und Robotik	Generierung von Bewegungsprogrammen für einen Roboterarm	A
	Lebensmittelindustrie	Optimierung des Mischverhältnisses verschiedener Kaffeesorten durch geschmacklichen Abgleich mit einem Referenzkaffee	A
Wissenschaft	Geophysik	Optimierung der Startparameter eines Simulationsmodells der Mantelkonvektion der Erde	A
	Bildung	Generierung von Sätzen aus Bildern zur Assistenz beim Verfassen von Kindergeschichten	A
	Soziale Systeme	Optimierung der Zusammensetzung von Studententeams zur Verbesserung der Teamleistung	A
	Data Mining	Generierung von Funktionen zur Klassifizierung von Datensätzen	B

A - Interaktiver evolutionärer Algorithmus nach der weiten Definition

B - Automatischer evolutionärer Algorithmus mit interaktiver Bewertung alle n Generationen zur Konvergenzbeschleunigung

Tabelle 2.1.: Anwendungsgebiete interaktiver evolutionärer Algorithmen (nach [Dre11] S. 7ff)

2.6. Sparse Fitness Evaluation

Eine Methode, bei der eine hohe Populationsgröße gewählt werden kann, während die Anzahl der zu evaluierenden Individuen sich nicht maßgeblich erhöht, ist die in [LC99] vorgeschlagene *Sparse Fitness Evaluation*. Der Grundgedanke dieser Methode ist, ähnlichen Individuen ähnliche Fitnesswerte zuzuweisen. Dazu sind zwei Schritte nötig: Clusterbildung und Fitnesszuweisung.

2.6.1. Clusterbildung

Als *Clusterbildung* wird eine Gruppierung von Objekten nach ihren Eigenschaften bezeichnet. Dies hat zum Ziel, dass Objekte innerhalb eines Clusters eine hohe Ähnlichkeit aufweisen und dadurch auch ähnlich behandelt werden können. Bei evolutionären Algorithmen mit Clusterbildung sollten Individuen innerhalb eines Clusters Lösungen mit ähnlicher Güte darstellen. Bei interaktiven evolutionären Algorithmen bedeutet dies, dass sich die Phänotypen in der subjektiven Wahrnehmung des Benutzers ähneln.

Die Population wird in eine Menge von Clustern zerlegt, wobei jeder Cluster hier eine Menge von phänotypisch ähnlichen Individuen ist. Nach der Erzeugung der Cluster sind die Schwerpunkte jedes Clusters bekannt. Es werden nur die Individuen, die den Schwerpunkten am nächsten sind, zur Evaluation dem Benutzer präsentiert. (vgl. [LC99])

2.6.2. Fitnesszuweisung

Die Fitnesszuweisung erfolgt bei den restlichen Individuen im Cluster in Abhängigkeit des euklidischen Abstands (Formel 2.2) der Genotypen.

$$dist(x) = \sqrt{\sum_{i=1}^{n} (c_i - x_i)^2} \tag{2.2}$$

Alle Fitnesswerte, die nicht durch den Benutzer zugewiesen wurden, sind prognostizierte Werte. [LC99] führt ein Beispiel an, bei dem der *k-Means*-Algorithmus zur Erzeugung der Cluster verwendet wurde. Durch Versuche wurde bestätigt, dass diese Methode bei einer geringeren Anzahl von Generationen zum Ziel geführt hat (vgl. [LC99]). Die Fitness von Individuen ist proportional zum

euklidischen Abstand vom Zentrum des Clusters. Dadurch entstehen in der Fitnesslandschaft Spitzen an den Clustergrenzen, wie in Abbildung 2.11 zu sehen ist. Die gestrichelte Linie stellt hier die tatsächliche Fitness dar, die durchgezogene Linie ist die prognostizierte Fitness. Zur Vereinfachung wird hier nur von einem Gen ausgegangen. Auf der x-Achse befinden sich die Allele dieses Gens, beispielsweise reelle Zahlen. Die Fitnessprognosen an den Randgebieten der Cluster können sehr ungenau sein. Die Unsicherheit dieser Fitnesswerte wird bei dieser Methode nicht mit einbezogen.

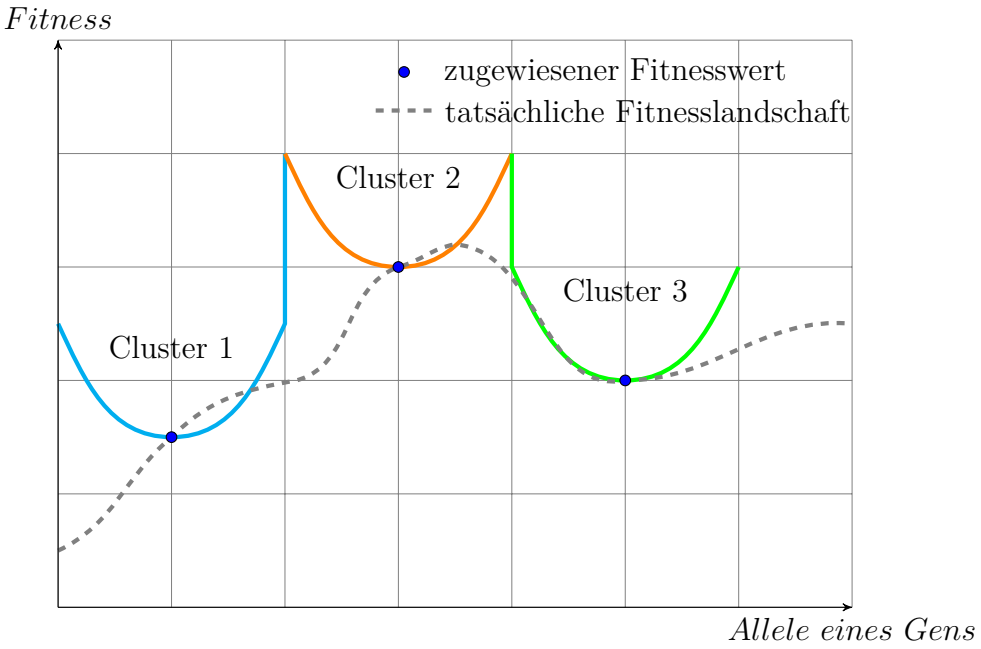


Abbildung 2.11.: Beispiel einer Fitnesslandschaft bei der in [LC99] vorgestellten Methode

2.6.3. IGA-LPS aus [GYM08]

In [GYM08] wird ein Cluster-Algorithmus vorgeschlagen, der nicht auf klassischen Algorithmen wie *k-Means* basiert. Dieser hat den Namen *IGA-LPS*[2]. Es wird vorausgesetzt, dass ein Genotyp in eine Menge von unabhängigen Segmenten unterteilt werden kann. Ein solches Segment wird dort als *Gene Meaning Unit* bezeichnet. Eine *Gene Meaning Unit* ist eine Menge von Genen, die eine phänotypische Eigenschaft beschreibt. Der erste Cluster wird erzeugt, indem zufällig ein Individuum aus der Population als Zentrum des Clusters gewählt

[2] *IGA-LPS* ist ein Akronym für *interactive genetic algorithm with large population size*

wird. Von diesem Zentrum wird zufällig eine *Gene Meaning Unit m* gewählt. Danach werden alle Individuen dem Cluster zugewiesen, bei denen die Ähnlichkeit α_m der gewählten *Gene Meaning Unit m* eins ist und die Ähnlichkeit α des Chromosoms unter dem Grenzwert α_0 liegt. α_0 ist hier eine vordefinierte Konstante, α und α_m lassen sich wie folgt berechnen:

$$\begin{aligned} \alpha(x_i, x_j) &= \frac{1}{N_g} \sum_{m=1}^{N_g} \alpha_m(x_i, x_j) \\ \alpha_m(x_i, x_j) &= \begin{cases} 1 & \text{wenn } x_{im} = x_{jm} \\ 0 & \text{wenn } x_{im} \neq x_{jm} \end{cases} \end{aligned} \tag{2.3}$$

Danach werden weitere Cluster mit demselben Verfahren erzeugt. Wenn die maximale Clusteranzahl N_{maxc} erreicht ist und noch Individuen existieren, die keinem Cluster zugeordnet sind, so werden diese unabhängig von α_m dem letzten Cluster zugeordnet. Dieser Algorithmus hat das Ziel, dass die Individuen innerhalb eines Clusters (abgesehen vom letzten) mindestens in einer Eigenschaft gleich sind. Im Gegensatz zur Methode aus [LC99] wird beim IGA-LPS die Unsicherheit der Fitnessprognosen berücksichtigt. Allgemein kann nicht definiert werden, ob die Fitnesswerte außerhalb des Zentrums eines Clusters kleiner oder größer als der Fitnesswert des Zentrums sind. In [LC99] werden diese als größer definiert. Bei der Methode aus [GYM08] hingegen ist die Fitness ein Intervall, mit dem die Unsicherheit des prognostizierten Fitnesswertes ausgedrückt wird. Dem Zentrum eines Clusters wird die Fitness durch den Benutzer zugewiesen, und die Intervallbreite ist null. Dies wird als Punktintervall bezeichnet. Den restlichen Individuen wird ein Fitnessintervall zugewiesen, dessen Mittelpunkt der Fitness des Clusterzentrums entspricht. Die Intervallbreite ist abhängig vom Abstand des Individuums zum Zentrum des Clusters (vgl. [GYM08]). Abbildung 2.12 zeigt als Beispiel drei Cluster mit den zugehörigen Fitnessintervallen. Die gestrichelte Linie stellt die tatsächliche Fitness dar. Die dreieckigen Flächen sind die Fitnessintervalle. Auch hier wird wieder von nur einem Gen im Genotyp ausgegangen.

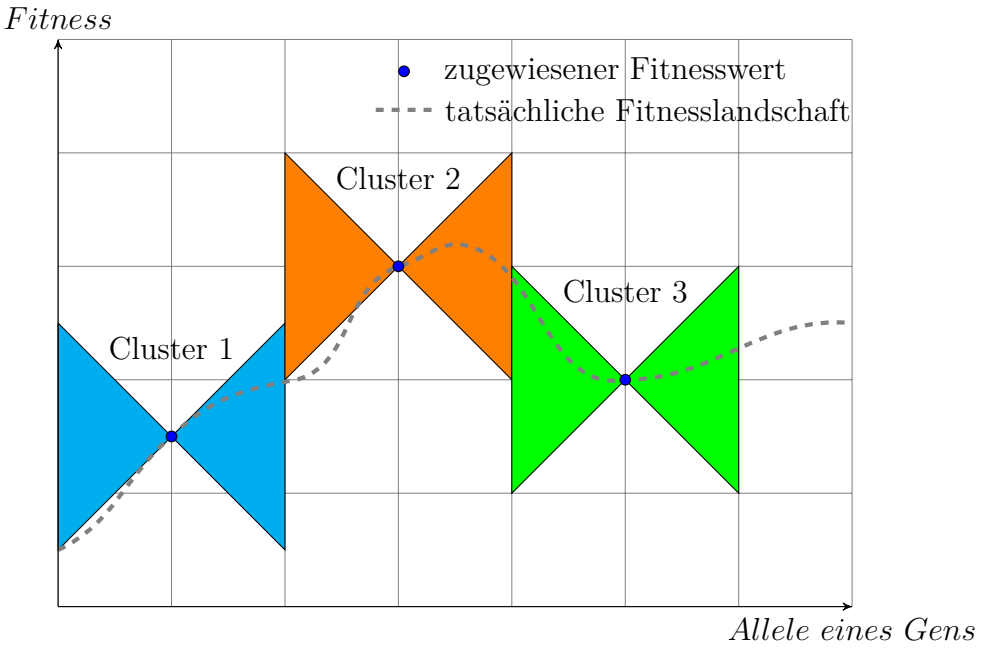
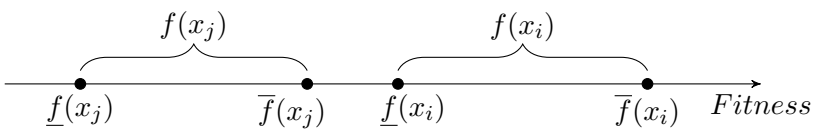


Abbildung 2.12.: Beispiel einer Fitnesslandschaft bei der Intervall-Fitness aus [GYM08]

2.6.4. Vergleich von Fitnessintervallen

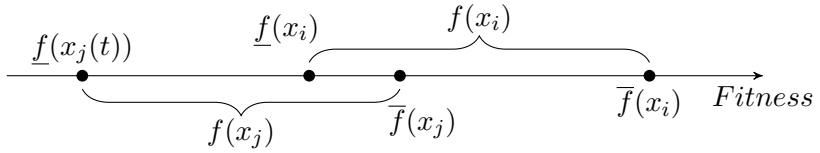
Anders als Fitnesswerte in Form von reellen Zahlen können Fitnessintervalle oft nicht direkt miteinander verglichen werden. Dennoch ist es notwendig, eine Dominanzbeziehung von Individuen mit Fitnessintervallen zu bestimmen. Der Algorithmus in [GYM08] bedient sich des in [GG07] vorgestellten Ansatzes der probabilistischen Dominanz. Sollen die Individuen x_i und x_j verglichen werden, deren Fitnessintervalle $f(x_i) = [\underline{f}(x_i), \overline{f}(x_i)]$ und $f(x_j) = [\underline{f}(x_j), \overline{f}(x_j)]$ sind, so müssen verschiedene Fälle betrachtet werden.

- Fall 1: $\overline{f}(x_i) > \overline{f}(x_j)$ und $\underline{f}(x_i) \geq \overline{f}(x_j)$.



Hier dominiert x_i x_j mit einer Wahrscheinlichkeit von eins.

- Fall 2: $\overline{f}(x_i) > \overline{f}(x_j)$ und $\underline{f}(x_i) < \overline{f}(x_j)$.

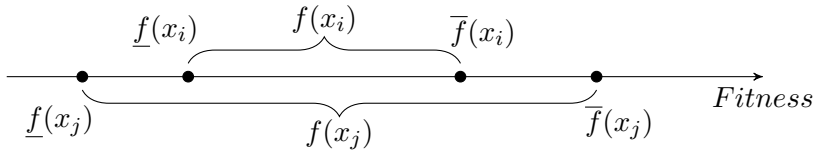


  Die Wahrscheinlichkeit p_{ij}, mit der x_i x_j dominiert, wird wie folgt berechnet:

$$\begin{aligned} p_{ij} =& \frac{\overline{f}(x_i) - \overline{f}(x_j)}{w(f(x_i))} + \frac{\overline{f}(x_j) - \underline{f}(x_i)}{w(f(x_i))} \\ & \cdot \left(0.5 \cdot \frac{\overline{f}(x_j) - \underline{f}(x_i)}{w(f(x_j))} + \frac{\underline{f}(x_i) - \underline{f}(x_j)}{w(f(x_j))} \right) \end{aligned} \tag{2.4}$$

  $w(f(x_n))$ ist die Breite des Fitnessintervalls.

- Fall 3: $\overline{f}(x_i) \leq \overline{f}(x_j)$ und $\underline{f}(x_i) \geq \underline{f}(x_j)$.



  Die Wahrscheinlichkeit p_{ij} wird wie folgt berechnet:

$$p_{ij} = 0.5 \cdot \frac{w(f(x_i))}{w(f(x_j))} + \frac{\underline{f}(x_i) - \underline{f}(x_j)}{w(f(x_j))} \tag{2.5}$$

Es gibt drei weitere Fälle, die inversen Relationen. Diese können ebenfalls mit den angegebenen Formeln behandelt werden. Nachdem die Dominanzwahrscheinlichkeit p_{ij} berechnet ist, wird eine Zufallszahl $r \in [0, 1]$ erzeugt. Wenn $r \leq p_{ij}$, ist x_i das dominierende Individuum, andernfalls x_j. Die Bestimmung der Dominanz zwischen zwei Individuen erfolgt also nicht deterministisch. Eine fitnessproportionale Selektion ist nicht möglich, da nur eine Dominanzbeziehung bestimmt werden kann. Es bietet sich die Turnierselektion an (vgl. [GG07]).

3. Analyse und Entwurf

In diesem Kapitel wird ein Konzept zur Umsetzung der Zielstellung entwickelt. Dazu werden die Anforderungen zunächst konkretisiert, ein evolutionärer Algorithmus und eine Benutzerschnittstelle entworfen und anschließend eine Methode zur Verringerung der Benutzerermüdung gewählt.

3.1. Ziele der Optimierung

Im Folgenden ist die Zielstellung dieser Arbeit in die Taxonomie für Ziele evolutionärer Algorithmen zur Mehrzieloptimierung aus [BRT05] eingeordnet. Der **Anwendungsfall** ist die Planung der Innenrichtung. Dieser lässt sich der **Domäne** der Innenarchitektur zuordnen. Ein **eindimensionales qualitatives Ziel** ist die Begehbarkeit. Sie lässt sich algorithmisch berechnen, aber der Zielwert liegt im Ermessen des Benutzers. Ästhetische Faktoren wie Anordnung, Art und Anzahl der Einrichtungsgegenstände, Stil, Kombination der Farben und die verwendeten Materialien sind **mehrdimensionale qualitative Ziele**. Diese liegen in der subjektiven Wahrnehmung des Benutzers und können nicht berechnet werden. Als **quantitatives Ziel** können die Anschaffungskosten bestimmt werden. Diese sollen meist in einem bestimmten Intervall, dem Budget des Benutzers liegen. Die Kosten eines Individuums lassen sich berechnen, sofern für jeden Einrichtungsgegenstand der Preis bekannt ist.

3.2. Anforderungen

Im Folgenden werden funktionale und nichtfunktionale Anforderungen genauer definiert. Diese ergeben sich aus der Zielstellung in Kapitel 1.2, den Umfrageergebnissen in Anhang A und den in Kapitel 2.4 beschriebenen Problemen bei der interaktiven Selektion.

3.2.1. Funktionen des genetischen Algorithmus

/F0010/
Das Programm erzeugt eine Startpopulation der Größe n und initialisiert diese mit Zufallswerten. Die Einrichtungsgegenstände werden zufällig aus einem Katalog von Gegenständen ausgewählt.

/F0020/
Die Software soll anhand der Bewertungen Individuen auswählen, die reproduziert werden. Dazu soll ein Selektionsoperator implementiert werden.

/F0030/
Während der Reproduktion sollen mit einer bestimmten Wahrscheinlichkeit die Eigenschaften von zwei Individuen kombiniert werden. Gemeinsame Eigenschaften sollen dabei erhalten bleiben. Dazu ist ein geeigneter Crossover-Operator zu implementieren.

/F0040/
Reproduzierte Individuen sollen mit einer bestimmten Wahrscheinlichkeit zufällig verändert werden. Dazu soll ein Mutations-Operator implementiert werden.

/F0050/
Es soll ein evolutionärer Zyklus implementiert werden, der nacheinander die Darstellung, Warten auf Benutzereingaben, Selektion, Crossover, und Mutation durchführt. Die Anzahl der Durchläufe ist durch einen Parameter zu begrenzen.

/F0060/
Es soll die Methode der Sparse Fitness Evaluation aus Kapitel 2.6 mit Fitnessintervallen (siehe Kapitel 2.6.3) implementiert werden.

/F0070/
Es soll ein Regelsystem zur Umsetzung von Randbedingungen implementiert werden. Mit Hilfe dieser Regeln sollen erzeugte Individuen validiert werden. Überschreiten die Regelverletzungen eines Individuums einen Grenzwert, so soll dieses nicht valide Individuum verworfen und neu erzeugt werden.

/F0080/
Die Zeit, die zur Suche einer zufriedenstellenden Lösung benötigt wird, soll

möglichst gering gehalten werden. Eine durchschnittliche Evaluationsdauer unter zehn Minuten soll angestrebt werden.

3.2.2. Funktionen der Benutzeroberfläche

/F0090/
Das Programm dekodiert die Individuen der Population und stellt jeweils ein Individuum grafisch dar. Die Darstellung erfolgt als dreidimensionaler Raum, in dem die jeweiligen Einrichtungsgegenstände platziert sind.

/F0100/
Die Bewertung eines Individuums durch den Benutzer soll mit möglichst wenigen Eingaben erfolgen.

/F0110/
Die Benutzeroberfläche soll Schaltflächen zur Bewertung von Individuen bieten. Die Bewertung soll in 5 Stufen möglich sein.

/F0120/
Die Benutzeroberfläche soll vier Schaltflächen bieten, mit denen die Perspektive verändert werden kann: Kamera horizontal nach links rotieren, Kamera horizontal nach rechts rotieren, Kamera vertikal nach oben rotieren, Kamera vertikal nach unten rotieren. Die vertikale Rotation soll durch einen minimalen und einen maximalen Winkel begrenzt sein. Die Kamera soll jederzeit auf den Raum ausgerichtet sein.

/F0130/
Erweiterung zu /F0120/: Die Rotation der Kamera soll auch durch ein Klicken und Ziehen mit der Maus möglich sein.

/F0140/
Die Gesamtkosten einer Inneneinrichtung sollen dem Benutzer jederzeit angezeigt werden.

/F0150/
Der Benutzer soll jederzeit die Möglichkeit haben, den Evolutionsprozess abzubrechen.

3.2.3. Konfiguration

/F0160/
Die Software soll sich über eine XML-Datei konfigurieren lassen. Beim Start soll diese Datei eingelesen und daraus alle Komponenten erzeugt werden.

/F0170/
Typ und Parameter der Population sollen über die XML-Datei konfiguriert werden können.

/F0180/
Typ und Parameter der Selektion sollen über die XML-Datei konfiguriert werden können.

/F0190/
Typ und Parameter der genetischen Operatoren sollen über die XML-Datei konfiguriert werden können.

/F0200/
Typ und Parameter der Regeln zur Validierung sollen über die XML-Datei konfiguriert werden können.

3.2.4. Administration

/F0210/
Zur Administration des Möbelkatalogs soll eine grafische Benutzeroberfläche zur Verfügung stehen.

/F0220/
Die Administrationsoberfläche soll das Einpflegen neuer Einrichtungsgegenstände ermöglichen. Dazu sollen 3D-Modelle in einem geeigneten Format importiert werden können.

/F0230/
Über die Administrationsoberfläche sollen bestehende Einrichtungsgegenstände angezeigt und gesucht werden können. Das 3D-Modell ausgewählter Einrichtungsgegenstände soll dargestellt werden.

/F0240/
Über die Administrationsoberfläche sollen die Metadaten der Einrichtungsgegenstände bearbeitet werden können. Dazu gehören Name, Preis, Artitelnummer, Kategorie, Verwendungszweck, Farbe, Stil, Kollektion und Hersteller.

/F0250/
Über die Administrationsoberfläche sollen Einrichtungsgegenstände gelöscht werden können.

3.2.5. Weitere Anforderungen

/F0260/
Das System soll plattformunabhängig implementiert werden und nicht auf systemspezifische Funktionen angewiesen sein. Dazu bietet sich die Programmiersprache Java an.

/F0270/
Das System soll modular aufgebaut sein. Das Modell, also der genetische Algorithmus, ist von der Präsentationslogik zu trennen. Für die Funktionen des genetischen Algorithmus (Selektion, genetische Operatoren, Validierung, Populationsverwaltung) sind geeignete Module zu entwerfen.

/F0280/
Das System soll nach dem Prinzip der losen Kopplung implementiert werden. Dazu sollen die Module über möglichst schmale Schnittstellen kommunizieren und austauschbar sein.

/F0290/
Das System soll möglichst generisch entwickelt werden. Einzelne Module sollen so weit wie möglich mit unterschiedlichen Datentypen funktionieren.

/F0300/
Das System soll leistungsfähig genug sein, um einen flüssigen Arbeitsablauf auf einem durchschnittlich ausgestattetem PC (aktueller PC unter 800 Euro) zu ermöglichen. Die Reaktionszeiten der grafischen Benutzeroberfläche sollen unter zwei Sekunden liegen.

3.3. Entwurf des evolutionären Algorithmus

3.3.1. Kodierung des Genotyps

Für die Umsetzung eines evolutionären Algorithmus ist es erforderlich, eine geeignete Kodierung für Problemlösungen zu finden. Für die hier vorliegende Zielstellung sind zwei Arten der Kodierung denkbar:

- Ein Genotyp beschreibt eine Regel, wie ein Einrichtungsplan konstruiert werden soll. Diese Regel wird durch eine Baumstruktur kodiert. Somit wäre der Algorithmus dem genetischen Programmieren zuzuordnen.
- Ein Genotyp beschreibt, welche Einrichtungsgegenstände ein Einrichtungsplan enthält und wie diese positioniert werden sollen. Er kann als Liste von reellen Zahlen kodiert werden. Dafür bietet sich ein genetischer Algorithmus an.

Beide Kodierungen eignen sich für die Anwendung, jedoch können Baumstrukturen im späteren Verlauf des Algorithmus sehr komplex werden, wenn sich die Regeln immer weiter verfeinern oder Regeln ohne Nutzen hinzukommen, so dass sich die durchschnittliche Fitness nicht verbessert. Dies tritt oft bei Strukturen variabler Größe auf und wird als *Bloat-Effekt* bezeichnet (vgl. [BHS07] S. 86). Die Kodierung als Liste von Einrichtungsgegenständen stellt ein transparenteres Modell dar. Der *Bloat-Effekt* wird implizit verhindert, wenn der Benutzer Einrichtungsplänen mit zu vielen Gegenständen eine schlechtere Bewertung zuweist. Somit fällt die Wahl auf eine Kodierung als Liste reeller Zahlen mit einer variablen Länge. Der Algorithmus kann als genetischer Algorithmus nach der weiten Definition eingeordnet werden (siehe Kapitel 2.2).

Für eine gute Konvergenz des Algorithmus sollte die Anzahl der Gene gering gehalten werden, daher sollte jedes Gen im Genotyp auch eine zusätzliche Information liefern (vgl. [Wei02] S. 99). Es wird festgelegt, dass die Einrichtungsgegenstände nur auf der zweidimensionalen Ebene positioniert werden können.

Für das Chromosom c bietet sich folgende Codierung an:

$$c = \{(ID_1, x_1, y_1, \theta_1), \ldots, (ID_n, x_n, y_n, \theta_n)\} \tag{3.1}$$

Die Funktion der einzelnen Gene des Chromosoms sei wie folgt erklärt:

- $ID_i \in \mathbb{N}$ kodiert die Identifikationsnummer (kurz: ID) des Einrichtungsgegenstandes
- $x_i \in \mathbb{R}$ kodiert die x-Koordinate im euklidischen Raum
- $y_i \in \mathbb{R}$ kodiert die y-Koordinate im euklidischen Raum
- $\theta_i \in \mathbb{R}$ kodiert den Winkel in Grad, mit dem der Einrichtungsgegenstand gedreht ist

$n \in \mathbb{N}$ ist die Anzahl der einzelnen Gensegmente, also der Einrichtungsgegenstände, die in dem Chromosom kodiert werden.

Der sich daraus ergebende Suchraum S_{such} wäre unendlich groß, daher müssen Einschränkungen getroffen werden. Die physischen Grenzen x_{min}, x_{max}, y_{min} und y_{max} des Raums, in dem der Einrichtungsgegenstand platziert werden soll, dürfen nicht überschritten werden. Eine Positionierung in Abständen von weniger als einem Zentimeter wird ausgeschlossen. Daher ist x_i eine ganze Zahl mit $x_{min} \leq x_i \leq x_{max}$ und y_i eine ganze Zahl mit $y_{min} \leq y_i \leq y_{max}$.

Da Räume oftmals eine rechteckige Form haben und Einrichtungsgegenstände oft parallel zur Wand ausgerichtet werden, wird der Freiheitsgrad der Drehung ebenfalls eingeschränkt. Es wird festgelegt, dass statt dem Winkel θ_i die Anzahl der Drehungen um 90° kodiert werden. Also gilt $\theta_i \in \{0, 1, 2, 3\}$.

Die in einem Chromosom kodierten Einrichtungsgegenstände entstammen einem vorgefertigtem Katalog. Sie sind also Elemente einer endlichen Menge von Einrichtungsgegenständen.

Ein Teil eines Chromosoms, der einen einzelnen Einrichtungsgegenstand und dessen Position und Drehung kodiert, wird im weiteren Verlauf dieser Arbeit als Segment bezeichnet:

$$seg = (ID, x, y, \theta) \tag{3.2}$$

Mit dieser Festlegung kann das Chromosom nun als eine Menge von Segmenten beschrieben werden:

$$c = \{seg_1, \ldots, seg_n\} \tag{3.3}$$

3.3.2. Suchraum

Der Suchraum ist die Menge aller möglichen Chromosomen. Allgemein lässt sich ohne Kenntnisse über die im Katalog enthaltenen Einrichtungsgegenstände die Größe des Suchraums nur auf ein Intervall festlegen, da die Einrichtungsgegenstände eine hier nicht bekannte Länge und Breite haben. Die Positionierung ist auf zwei Arten eingeschränkt:

- Zwei Einrichtungsgegenstände können sich niemals überschneiden.
- Ein Einrichtungsgegenstand kann sich nicht mit einer Wand des Raums überschneiden.

Die Formel für das Intervall des Suchraums lautet wie folgt:

$$0 \leq |S_{such}| < \sum_{i=n_{min}}^{n_{max}} (n_{furn} \cdot (x_{max} - x_{min}) \cdot (y_{max} - y_{min}) \cdot 4)^i \tag{3.4}$$

Bei der oberen Grenze des Intervalls wird der Einrichtungsgegenstand als dimensionslos angenommen. Da dies in der Realität niemals der Fall sein kann, ist das Intervall rechtsoffen. Die untere Grenze ist null, denn möglicherweise passt keiner der zur Verfügung stehenden Einrichtungsgegenstände in den Raum. Eine Beispielrechnung soll die mögliche Größe des Suchraums genauer zeigen:

Angenommen, es stehen 50 verschiedene Einrichtungsgegenstände zur Verfügung und es soll ein Raum ausgestattet werden, der fünf Meter breit und sechs Meter lang ist. Der Raum soll in diesem Beispiel mit mindestens einem und höchstens fünf Einrichtungsgegenständen ausgestattet werden. Der Suchraum hätte dann eine Größe von

$$0 \leq |S_{such}| < \sum_{i=1}^{5} (50 \cdot 500 \cdot 600 \cdot 4)^i = 7.78 \cdot 10^{38} \tag{3.5}$$

3.3.3. Genotypischer Abstand

In einigen Programmteilen des evolutionären Algorithmus ist es erforderlich, einen Abstand zwischen zwei Individuen zu bestimmen. Wird ein Gen eines Individuums mutiert, so soll dadurch ein ähnliches Individuum erzeugt werden. Beispielsweise kann eine Koordinate im euklidischen Raum durch eine andere

Koordinate mit einem geringen euklidischen Abstand ersetzt werden. Eine geringe Variation der Anzahl der Drehungen eines Individuums hat ebenfalls nur eine geringe Änderung des Phänotyps zur Folge. Bei der ID der Einrichtungsgegenstände ist das nicht der Fall. ID und $ID+1$ können völlig unterschiedliche Einrichtungsgegenstände sein.

Menschen sind in der Lage, durch ihr subjektives Empfinden die Ähnlichkeit zwischen zwei Einrichtungsgegenständen zu beurteilen. Um eine Abstandsfunktion zu entwerfen ist es erforderlich, diese Ähnlichkeit zu beschreiben. Dabei muss analysiert werden, wieso für einen Menschen beispielsweise zwei Stühle A und B ähnlich sind. Eine Gemeinsamkeit ist trivial. Beides sind Stühle. Folglich kann ein Parameter eingeführt werden, der die Einrichtungsgegenstände nach ihrer Art klassifiziert. Angenommen, es existiert noch ein Sessel C. Dieser Sessel ist kein Stuhl, kann jedoch ebenfalls zum Sitzen verwendet werden. Der Zweck des Einrichtungsgegenstandes kann folglich als weiterer Parameter eingeführt werden. Weitere Parameter sind die Farbe, das Material und der Stil. Auch der Hersteller und die Kollektion, der der Einrichtungsgegenstand angehört, könnten für den Menschen relevante Parameter sein. Um zu verhindern, dass bei der Einrichtung eines Wohnzimmers eine Couch mit Schlaffunktion durch ein Bett ersetzt wird, ist die Klassifizierung nach dem Zweck nicht ausreichend. Ein Bett steht meistens in einem Schlafzimmer. Also wird ein Parameter eingeführt, der den Raum definiert. Den einzelnen Parametern muss eine Wichtung zugewiesen werden, denn eine Abweichung in der Farbe beispielsweise sollte in einem geringeren Abstand resultieren als eine Abweichung im Zweck.

In Tabelle 3.1 werden mögliche Parameter von Einrichtungsgegenständen aufgelistet. Die Wichtung ist dort textuell beschrieben, sollte bei der Realisierung jedoch durch numerische Werte repräsentiert werden.

Jedem Allel für das Gen ID wird zur Erhöhung des Informationsgehalts eine endliche Menge von *Annotationen* zugeordnet. Jede Annotation ist ein Tupel aus Schlüssel und Wert. Das Beispiel in Formel 3.6 zeigt die Annotationsfunktion die dem Allel 42 für das Gen ID eine Menge von Annotationen zuweist.

$$\begin{aligned} f_{anno}(42) = \{&(Raum, Wohnzimmer), (Art, Stuhl), (Zweck, Sitzen), \\ &(Farbe, Schwarz), (Material, Leder)\} \end{aligned} \tag{3.6}$$

Parameterschlüssel	**Beispiele für Werte**	**Wichtung**
Raum	Wohnzimmer, Schlafzimmer, Büro, Kinderzimmer	sehr hoch
Art	Tisch, Bett, Stuhl, Sessel, Couch, Schrank, Kommode	hoch
Zweck	Sitzen, Schlafen, Entspannen, Arbeiten, Essen, Unterbringung von Kleidungsstücken, Unterbau für TV	hoch
Orientierung	zur Wand, in den Raum, in Richtung eines Arbeitsmobiliars, in Richtung eines Essmobiliars	hoch
Positionierung	Frei im Raum, an einer Wand	hoch
Farbe	schwarz, beige, braun, weiß	mittel
Stil	klassisch, modern, pragmatisch, rustikal, asiatisch	mittel
Kollektion	Metropolis, Jersey	mittel
Hersteller	Ikea, ACME	gering
Material	Leder, Birke, Buche, Ahorn, Nussbaum	gering

Tabelle 3.1.: Beispiele für Parameter von Einrichtungsgegenständen

Für die Berechnung des genotypischen Abstands von zwei Individuen muss der Abstand *dist* zwischen den jeweiligen Chromosomen berechnet werden. Die Abstandsfunktion ist kommutativ, das heißt Chromosom c_1 hat zum Chromosom c_2 den selben Abstand wie c_2 zu c_1 (siehe Formel 3.7).

$$dist(c_1, c_2) = dist(c_2, c_1) \tag{3.7}$$

Weiterhin muss für jedes Chromosom gelten:

$$dist(c_n, c_n) = 0 \tag{3.8}$$

Es wird festgelegt, dass der Abstand normiert ist, daher gilt $dist \in [0, 1]$. Listing 3.1 zeigt im Pseudocode, wie der Abstand von Chromosomen berechnet wird. Die Funktion `dist` stellt die Kommutativität sicher, indem der Abstand von c_1 zu c_2 zum Abstand c_2 zu c_1 addiert und die Summe halbiert wird. Die Funktion `segmentAbstandVonLinks` sucht jeden einzelnen Einrichtungsgegenstand eines Segments des ersten Chromosoms im zweiten Chromosom. Wird dieses gefunden, so kann der Abstand beider Segmente mit der Formel 3.9 berechnet werden. Wird dieses jedoch nicht gefunden, so berechnet die Funktion `geringsterAbstand` den Segmentabstand aus dem Minimum der Abstände aller Segmente zum Ausgangssegment.

```
1  dist( Chromosom c1, Chromosom c2 ) {
2    // berechne Abstand von c1 zu c2
3    distanz1 := segmentAbstandVonLinks( c1, c2 )
4
5    // berechne Abstand von c2 zu c1
6    distanz2 := segmentAbstandVonLinks( c2, c1 )
7
8    // addieren und normalisieren
9    return ( distanz1 + distanz2 ) / 2
10 }
11
12 segmentAbstandVonLinks( Chromosom c1, Chromosom c2 ) {
13   if ( leer( c1 ) && leer( c2 ) ) {
14     return 0
15   } else if ( leer( c1 ) != leer( c2 ) ) {
16     return 1
17   }
18
19   distanz := 0
20
21   for ( Segment seg : c1 ) {
22     seg[] gemSeg := finde alle Segmente mit dem gleichen Möbelstück wie in fs
23
24     if ( nichtLeer( gemSeg ) ) {
25       distanz := distanz + geringsterAbstand( seg, gemSeg )
26     } else {
27       distanz := distanz + geringsterAbstand( seg, c2 )
28     }
29   }
30
31   return distanz / laenge( c1 )
32 }
33
34 geringsterAbstand( Segment seg1, Chromosom andere ) {
35   distanz := 1
36
37   for ( Segment seg2 : andere ) {
38     distanz := min( distanz, distSeg( seg1,seg2 ) )
39   }
40
41   return distanz
42 }
```

Listing 3.1: Pseudocode für die Abstandsfunktion

Formel 3.9 zeigt den Abstand zweier Segmente. Die Konstanten w_{trans}, w_{rot}, w_{skal} und w_{anno} sind Wichtungen für die einzelnen Abstände.

$$distSeg(seg_i, seg_j) = \Big(\frac{w_{trans} \cdot dist_{trans}(seg_i, seg_j) + w_{rot} \cdot dist_{rot}(seg_i, seg_j)}{w_{trans} \cdot w_{rot} \cdot w_{skal} \cdot w_{anno}} + \frac{w_{skal} \cdot dist_{skal}(seg_i, seg_j) + w_{anno} \cdot dist_{anno}(seg_i, seg_j)}{w_{trans} \cdot w_{rot} \cdot w_{skal} \cdot w_{anno}} \Big) \tag{3.9}$$

Die Distanz der Translationen $dist_{trans}$ ist der auf die Raumgröße normierte euklidische Abstand der Koordinaten zweier Einrichtungsgegenstände:

$$dist_{trans}(seg_1, seg_2) = \frac{\sqrt{(x(seg_1) - x(seg_2))^2 + (y(seg_1) - y(seg_2))^2}}{\sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}} \tag{3.10}$$

Weiterhin ist $dist_{rot}$ der normierte Abstand der Rotationen zweier Einrichtungsgegenstände:

$$dist_{rot}(seg_1, seg_2) = \frac{\left| \left(\left(\frac{rot(seg_1) - rot(seg_2) + 2}{4} - \left\lfloor \frac{rot(seg_1) - rot(seg_2) + 2}{4} \right\rfloor \right) \cdot 4 \right) - 2 \right|}{2} \tag{3.11}$$

Die Distanz der Skalierungen $dist_{skal}$ ist die normierte Manhattan-Distanz von Länge und Breite zweier Einrichtungsgegenstände:

$$dist_{skal}(seg_1, seg_2) = \frac{|laenge(seg_1) - laenge(seg_2)| + |breite(seg_1) - breite(seg_2)|}{\max(laenge(seg_1), laenge(seg_2)) + \max(breite(seg_1), breite(seg_2))} \tag{3.12}$$

$laenge(g)$ und $breite(g)$ sind die Funktionen, die dem Gen die Länge und Breite des Einrichtungsgegenstandes zuweisen.

Der Abstand der Annotationen $dist_{anno}$ wird wie folgt berechnet:

$$dist_{anno}(seg_1, seg_2) = 1 - \frac{sum_{wichtung}(f_{anno}(seg_1) \cap (f_{anno}(seg_2))}{sum_{wichtung}(f_{anno}(seg_1) \cup f_{anno}(seg_2))} \tag{3.13}$$

Die Summe der Wichtungen $sum_{wichtung}$ aus einer Menge von Annotationen A berechnet sich wie folgt:

$$sum_{wichtung}(A) = \sum_{q=1}^{|A|} wichtung(anno_q(A)) \tag{3.14}$$

Als Beispiel soll angenommen werden, die Segmente seg_1 und seg_2 haben folgende Annotationen:

$$f_{anno}(seg_1) = \{(Raum, Wohnzimmer), (Art, Stuhl), (Zweck, Sitzen)\}$$

$$f_{anno}(seg_2) = \{(Raum, Wohnzimmer), (Art, Couch), (Zweck, Sitzen)\}$$

Weiterhin werden folgende Wichtungen angenommen:

$$wichtung(Raum) = 20$$

$$wichtung(Art) = 10$$

$$wichtung(Zweck) = 8$$

Dann ergibt sich der Abstand der Annotationen von seg_1 und seg_2 wie folgt:

$$dist_{anno}(seg_1, seg_2) = 1 - \frac{sum_{wichtung}(f_{anno}(seg_1) \cap (f_{anno}(seg_2))}{sum_{wichtung}(f_{anno}(seg_1) \cup f_{anno}(seg_2))}$$

$$= 1 - \frac{sum_{wichtung}(\{(Raum, Wohnzimmer), (Zweck, Sitzen)\})}{sum_{wichtung}(\{(Raum, Wohnzimmer), (Art, Stuhl), (Art, Couch), (Zweck, Sitzen)\})}$$

$$= 1 - \frac{28}{48} = 0,42$$

3.3.4. Randbedingungen

Der Suchraum kann je nach Größe des einzurichtenden Raums und Anzahl der verfügbaren Möbelstücke sehr groß werden (siehe Kapitel 3.3.2). Es gibt jedoch auch Bereiche im Suchraum, die beispielsweise durch physikalische Gesetzmäßigkeiten keine gültigen Lösungen enthalten können. Unter Einbeziehung des Problemwissens kann der Suchraum reduziert werden. In der Realität kann sich beispielsweise ein Einrichtungsgegenstand niemals mit einem anderen Einrichtungsgegenstand oder einer Wand überschneiden. Um diese Einschränkungen algorithmisch umzusetzen, müssen Randbedingungen formuliert werden. Diese sollen physikalische Grenzen berücksichtigen und Benutzerpräferenzen widerspiegeln. Dazu werden zwei Arten von Randbedingungen definiert. Harte Randbedingungen müssen bei jedem Individuum zwingend erfüllt werden. Die

Erfüllung weicher Randbedingungen ist nicht zwingend gefordert, wird jedoch angestrebt (vgl. [Wei02] S. 172). Werden weiche Randbedingungen bei einem Individuum verletzt, können dessen Nachkommen dennoch bessere Lösungen darstellen. Die zulässige Gesamtheit der verletzten Bedingungen sollte jedoch beschränkt werden.

Für den zu entwickelnden evolutionären Algorithmus wurde eine Liste denkbarer Randbedingungen erstellt. Bei den weichen Randbedingungen liegen einige im subjektiven Empfinden des Benutzers und variieren möglicherweise zwischen einzelnen Benutzern:

Harte Randbedingungen

- Ein Einrichtungsgegenstand darf sich niemals mit einem anderen überschneiden
- Ein Einrichtungsgegenstand darf sich niemals mit einer Wand überschneiden

Weiche Randbedingungen

- Ein Schrank sollte in Richtung des Raums gedreht sein. Er ist beispielsweise wenig sinnvoll, wenn er direkt an einer Wand steht und sich nicht öffnen lässt.
- Hohe Einrichtungsgegenstände sollten kein Fenster verdecken.
- Ein bestimmter Gesamtpreis der Einrichtung sollte nicht überschritten werden.
- Ein bestimmter Einrichtungsgegenstand, beispielsweise ein Tisch, sollte weiter weg von der Wand stehen.
- Ein Schrank sollte an einer Wand stehen. Einige Benutzer könnten jedoch auch wünschen, dass er als Raumtrenner fungiert.
- Die Anzahl der Einrichtungsgegenstände eines bestimmten Typs sollte ein Minimum und ein Maximum haben. Beispielsweise ist in einem Schlafzimmer nur ein Bett vorgesehen, aber der Platz könnte auch für zwei kleine Betten ausreichend sein.
- Ein Einrichtungsgegenstand sollte in Relation zu einem anderen stehen. Beispielsweise sollte ein Stuhl an einem Tisch stehen. Oder ein Fernseher sollte eine in einer bestimmten Entfernung von einer Couch stehen.

- Die Begehbarkeit des Raumes sollte mit einbezogen werden. Dies ist besonders der Fall, wenn es sich um ein Durchgangszimmer handelt.
- Einrichtungsgegenstände gleicher Farbe sollten bevorzugt werden.
- Einrichtungsgegenstände, die einer gemeinsamen Kollektion angehören, sollten bevorzugt werden.

Der evolutionäre Algorithmus benötigt ein Regelsystem zur Validierung, welches die Randbedingungen umsetzen soll. Wie bereits beschrieben, ist es dafür notwendig, dass Regeln mit verschiedenen Prioritäten möglich sind. Bei der algorithmischen Umsetzung der Einhaltung dieser Regeln schlägt die Literatur folgende Methoden vor (vgl. [Wei02] S. 174ff):

- **Kindstod**. Es werden zufällig oder durch genetische Operatoren Individuen erzeugt und mit Hilfe der Regeln auf ihre Eignung getestet. Überschreitet ein Individuum einen bestimmten Schwellwert an Regelverletzungen, so wird es verworfen und ein neues erzeugt.
- **genetisches Reparieren**. Es werden zufällig oder durch genetische Operatoren Individuen erzeugt und mit Hilfe der Regeln auf ihre Eignung getestet. Wenn dabei ein bestimmter Schwellwert an Regelverletzungen überschritten wird, werden die Individuen so lange modifiziert, bis sie eine gültige Lösung repräsentieren.
- **legale Dekodierung**. Individuen werden generiert und so dekodiert, dass ausschließlich legale Phänotypen erzeugt und bewertet werden. Der Genotyp bleibt dabei unverändert. Hier kann von Vorteil sein, dass auch ungültige Genotypen in der Population bleiben und den Suchraum erforschen können.
- **legale Individuen**. Hier wird die Anfangspopulation ausschließlich mit gültigen Individuen initialisiert. Die genetischen Operatoren müssen so entworfen werden, dass sie ausschließlich gültige Individuen erzeugen können. Diese Methode erfordert einen höheren Aufwand bei der Implementierung.
- **Straffunktionen**. Ungültige Individuen werden in der Population belassen und mit einem angemessenen Strafwert belegt, der die Reproduktionswahrscheinlichkeit verringert.

3.3.5. Genetische Operatoren

Für die Funktion eines genetischen Algorithmus ist es erforderlich, dass mindestens ein genetischer Operator verwendet wird. Dabei werden zwei grundlegende Ziele verfolgt: die Erforschung (abgeleitet vom englischen Begriff *exploration*) des Suchraums und die Feinabstimmung (abgeleitet vom englischen Begriff *exploitation*) der potentiellen Lösungen.

In [Wei02] wurden Entwurfsprinzipien für genetische Operatoren definiert. Diese gehen von einer festen Rollenverteilung aus: das Crossover sollte nur zur Kombination der Eigenschaften von zwei Individuen dienen, die Mutation soll die **Erreichbarkeit aller Punkte** im Suchraum gewährleisten. Das bedeutet, dass mit dem Mutations-Operator theoretisch jede beliebige Lösung gefunden werden kann.

Eine Anforderung an den Crossover-Operator ist, dass er eine **Verschmelzung** durchführen kann. Das bedeutet, dass sich unterschiedliche Eigenschaften der Elternindividuen kombinieren lassen.

Weiterhin sollen **gemeinsame Eigenschaften der Eltern erhalten bleiben** und auf die Kinder übertragen werden. Dies hat zum Ziel, dass gemeinsame Eigenschaften guter Individuen weitervererbt werden können und durch das Crossover nicht wieder zerstört werden.

Ein weiteres Prinzip ist die **Übertragung von Genen** oder phänotypischen Allelen. Es besagt, dass bei jedem Kind jede Eigenschaft auf mindestens ein Elternteil zurückgeführt werden kann. Dieses Prinzip ist nicht zwingend erforderlich und wird beispielsweise bei binären Lösungsrepräsentationen oft verletzt. In diesem Falle kann der Operator auch eine implizite Mutation durchführen. (vgl. [Wei02] S. 65, 99f)

Der **Positional Bias** ist ein Problem, bei dem die Austauschwahrscheinlichkeit eines Gens durch Anwendung eines genetischen Operators von der Position des Gens im Chromosom abhängt. Dies ist meistens nicht erwünscht. (vgl. [GKK04] S. 87)

3.3.5.1. Mutation

Die Mutation ist allgemein formuliert ein Operator, der ein Chromosom zufällig verändert. Im Folgenden sind einige Mutations-Operatoren aus der Literatur aufgelistet:

- **Reelle Mutation**. Jedes Gen des Chromosoms, unabhängig von dessen Funktion, kann mit einer Wahrscheinlichkeit p_{mut} einzeln und ohne Berücksichtigung der anderen Gene um einen Zufallswert r innerhalb der Schrittweite w verändert werden.
- **Permutationserhaltende Mutation**. Hier wird lediglich die Position von Genen verändert. Dazu gehören **Tausch** zweier Gene, **Inversion**, **Permutation** und **Verschieben** eines Teilstrings. (vgl. [BHS07] S. 79)

Diese Operatoren lassen die Funktion der einzelnen Gene außer Acht. Dies bedeutet speziell bei der ID des Einrichtungsgegenstandes, dass bereits geringe Veränderungen des Gens eine große Veränderung der subjektiven Wahrnehmung des Phänotyps bedeuten. Bei der hier vorliegenden Zielstellung und der in Kapitel 3.3.1 definierten Kodierung werden problemspezifische Mutations-Operatoren vorgeschlagen, die die funktionale Bedeutung einzelner Gene und deren Relation zueinander berücksichtigen:

- Beim **Verschieben und Austauschen** wird jeder Einrichtungsgegenstand mit einer Wahrscheinlichkeit p_{move} um zufällige Werte im Raum verschoben und gedreht. Zusätzlich wird jeder Einrichtungsgegenstand mit einer Wahrscheinlichkeit $p_{exchange}$ gegen einen ähnlichen Gegenstand getauscht. Die Ähnlichkeit ergibt sich aus einem geringen genotypischen Abstand, der in Kapitel 3.3.3 beschrieben wurde. Zur Begrenzung der Operatoreffekte werden die Schrittweiten rot_{max} für die Rotation, $move_{max}$ für die Translation und $dist_{max}$ für den Abstand der ID eingeführt.
- Beim **Positionstausch** bleiben die Einrichtungsgegenstände erhalten. Es werden lediglich deren Positionen mit einer Wahrscheinlichkeit p_{switch} untereinander getauscht.

3.3.5.2. Crossover

Der Crossover-Operator kombiniert zwei Individuen und erzeugt ein oder mehrere Nachkommen. Dadurch können gute Eigenschaften von zwei Individuen in einem neuen Individuum zusammengeführt werden. (vgl. [BHS07] S. 79) Im Folgenden ist ein Auszug von allgemeinen Crossover-Varianten aufgeführt:

- Beim **n-Punkt-Crossover** werden die Chromosomen der Eltern an n Positionen zerschnitten und die entstandenen Teilabschnitte zwischen den Eltern getauscht. (vgl. [BHS07] S. 80)
- Beim **Parametrisierten Uniform Crossover** wird für jedes Gen einzeln ausgewürfelt, ob es zwischen den Elternteilen getauscht wird. Parameter des Operators ist die Austauschwahrscheinlichkeit p_c. Besitzt p_c den festen Wert 0,5, so handelt es sich um ein *Uniform Crossover*. (vgl. [GKK04] S. 89)
- Beim **Shuffle-Crossover** werden die Gene der Chromosomen zufällig vertauscht, bevor ein weiterer Crossover-Operator angewendet wird. Nach dem Crossover werden die Gene wieder in ihre ursprüngliche Reihenfolge gebracht. Dies soll dem *Positional Bias* vorbeugen. (vgl. [GKK04] S. 90)
- Zu den **permutationserhaltenden Crossover-Operatoren** zählen das *Partially Matched Crossover* (*PMX*), das *Ordered Crossover* (*OX*) und das *Cycle Crossover* (*CX*). *PMX* und *OX* zerschneiden die Chromosomen der Eltern an zwei Positionen und stellen die Permutation der Gene der Nachkommen wieder her. Beim *Cycle Crossover* wird für jedes Gen einzeln entschieden, ob es ausgetauscht werden darf. Sobald der Permutationskreis komplett ist, können die restlichen Gene ausgetauscht werden. (vgl. [BHS07] S. 80, [GKK04] S. 91ff)
- Das **Arithmetische Crossover** ist bei Genotypen mit reellen Zahlen anwendbar. Die Chromosomen der Nachkommen setzen sich aus Linearkombinationen der Eltern zusammen. $X' := a \cdot X + (1 - a) \cdot Y$ und $Y' := a \cdot Y + (1 - a) \cdot X$, wobei $a \in [0, 1]$ zufällig gewählt wird. (vgl. [GKK04] S. 151, [BHS07] S. 80)

Es sind auch folgende problemspezifische Operatoren denkbar:

- Beim **Segmentaustausch** wird ein Segment, welches einen Einrichtungsgegenstand mit seiner Position und Drehung kodiert, mit einer Wahrscheinlichkeit p_{swap} gegen das Segment getauscht, das sich im anderen Chromosom an der gleichen Position befindet. Sollte ein Chromosom länger als das andere sein, so werden die Segmente an der jeweiligen Position in das kürzere Chromosom verschoben. Dieser Operator basiert auf dem Uniform Crossover.
- Das **Segment-n-Punkt-Crossover** basiert auf dem n-Punkt-Crossover, tauscht jedoch ganze Segmente statt einzelner Gene. Wird eine Schnittposition außerhalb des kürzeren Chromosoms gewählt, so wird der Teilabschnitt des längeren Chromosoms an das kürzere angehängt.
- Die **gemittelte Kombination** funktioniert ähnlich dem Arithmetischen Crossover. Es werden Nachkommen erzeugt, die einen geringen Abstand zu beiden Eltern haben. Der erste Nachkomme enthält die Linearkombinationen der Positionen und Rotationen der einzelnen Segmente. Die IDs werden vom ersten Elter übernommen. Der zweite Nachkomme enthält die gemittelten IDs der Einrichtungsgegenstände. Dazu werden jene ausgewählt, die einen geringen Abstand (siehe Kapitel 3.3.3) zu den IDs der Eltern haben. Die Positionen und Rotationen werden vom zweiten Elter übernommen.
- Das **Segment-Schnitt-Crossover** wird mit dem Ziel vorgeschlagen, die Entwurfsprinzipien aus [Wei02] zu erfüllen und dabei den *Positional Bias* zu vermeiden. Es berücksichtigt besonders die Struktur der Genotypen. Die Funktionsweise sei so erklärt: Im ersten Schritt werden die Chromosomen c_1 und c_2 der Eltern als Mengen von Segmenten angesehen. Um gemeinsame Segmente vor Veränderungen zu schützen, wird die Schnittmenge $c_\cap := c_1 \cap c_2$ gebildet. Dann werden die Differenzmengen der Chromosomen zur Schnittmenge gebildet: $c_A := c_1 - c_\cap$ und $c_B := c_2 - c_\cap$. Die daraus entstandenen Mengen enthalten dann nur noch die exklusiven Elemente von c_1 und c_2. Die Funktion $rndorder$ bringt die Elemente einer Menge in eine zufällige Reihenfolge. Durch die Anwendung dieser Funktion entstehen die Mengen $c_A' := rndorder(c_A)$ und $c_B' := rndorder(c_B)$. Dieser Schritt eliminiert den *Positional Bias*. Danach wird eine Zufallszahl r erzeugt, wobei gilt $1 \leq r < max(|c_A'|, |c_B'|)$. Die Mengen c_A' und

$c_B{}'$ werden nun an der Stelle r zerschnitten. Dadurch entstehen die Mengen c_{A1}', c_{A2}', c_{B1}' und c_{B2}'. Auf c_{A1}' und c_{B1}' wird das Uniform Crossover auf Parameterebene angewendet. Dadurch ist die Verschmelzungseigenschaft erfüllt. Anschließend werden die neuen Chromosomen c_3 und c_4 gebildet, wobei c_{A2}' und c_{B2}' getauscht werden: $c_3 := c_\cap \cup c_{A1}' \cup c_{B2}'$ und $c_4 := c_\cap \cup c_{B1}' \cup c_{A2}'$.

3.3.5.3. Bewertung der Operatoren

Tabelle 3.2 fasst die vorgestellten Operatoren zusammen und zeigt deren Eigenschaften hinsichtlich der Entwurfsprinzipien aus [Wei02]. Weiterhin ist ersichtlich, ob es sich um problemangepasste Operatoren handelt, sie unter dem Problem des *Positional Bias* leiden und ob ganze Segmente erhalten bleiben. Dabei kann die das Erhalten von Segmenten weder als positiv, noch als negativ gesehen werden.

Zur Mutation eignen sich die Reelle Mutation und das Verschieben und Austauschen, da bei beiden alle Punkte im Suchraum erreicht werden können.

Der in dieser Arbeit entwickelte Genotyp hat besondere Eigenschaften, die von klassischen Crossover-Operatoren nicht berücksichtigt werden:

- Die Länge der Chromosomen ist variabel. Daher muss bei einem Crossover berücksichtigt werden, dass die beiden Eltern unterschiedlich lang sein können. Klassische Operatoren sind hier ungeeignet.
- Es existiert keine feste Reihenfolge der Segmente im Chromosom. Dies erweist sich als problematisch, da die Erhaltung gemeinsamer Eigenschaften der Eltern bei rein Index-basierten Crossover-Operatoren nicht garantiert werden kann.

Diese Eigenschaften werden nur beim Segment-Schnitt-Crossover-Operator berücksichtigt, indem Mengenoperationen verwendet werden. Der *Positional Bias* wird eliminiert, die Verschmelzung von Eigenschaften ist möglich.

Art	**Operator**	**problem-angepasst**	**Erreichbarkeit aller Punkte**	**Positional Bias**	**Verschmelzung**	**Segmente bleiben erhalten**	**gemeinsame Eigenschaften bleiben erhalten**
Mutation	Reelle Mutation	–	×	–			
	Permutationserhaltende Mutation	–	–	–			
	Verschieben und Austauschen	×	×	–			
	Positionstausch	×	–	–			
Crossover	n-Punkt-Crossover	–		×	×	–	$(-)^3$
	Parametrisiertes Uniform Crossover	–		–	×	–	$(-)^3$
	Shuffle-Crossover	–		–	×	–	$(-)^3$
	PMX, OX, CX	–		–	–	–	–
	Arithmetisches Crossover	–		–	–	–	$(-)^3$
	Segmentaustausch	×		$(\times)^1$	$(\times)^2$	×	$(-)^3$
	Segment-n-Punkt-Crossover	×		$(\times)^1$	$(\times)^2$	×	–
	gemittelte Kombination	×		–	–	–	–
	Segment-Schnitt-Crossover	×		–	×	–	×

× Eigenschaft vorhanden

– Eigenschaft nicht vorhanden

$(\times)^1$ Eigenschaft nur auf Parameterebene vorhanden

$(\times)^2$ Eigenschaft nur auf Segmentebene vorhanden

$(-)^3$ Eigenschaft bei dem gewählten Genotyp nicht vorhanden

Tabelle 3.2.: Eigenschaften genetischer Operatoren im Kontext der Zielstellung

3.3.6. Selektion

Neben den genetischen Operatoren ist die Selektion maßgeblich an der Funktion eines evolutionären Algorithmus beteiligt. Die folgenden Selektionsmethoden wären für die hier beschriebene Problemstellung denkbar:

- Bei der **fitnessproportionalen Selektion** ergibt sich die Selektionswahrscheinlichkeit aus der relativen Fitness eines Individuums in der Population. Diese Methode erfordert die Existenz eines reellen Fitnesswertes für jedes Individuum. Sie ist nicht duplikatfrei, daher kann ein Individuum mehrmals ausgewählt werden. Bekannte Vertreter dieser Methode sind die *Roulette-Selektion* und das *Stochastic Universal Sampling*. (vgl. [Wei02] S. 70ff, [BHS07] S. 74)
- Die **rangbasierte Selektion** hat eine geringere Selektionsintensität. Aus den Fitnesswerten der Individuen wird eine Rangliste erstellt. Die Selektionswahrscheinlichkeit eines Individuums basiert auf dessen Rang. Diese Methode erfordert nicht zwingend reelle Fitnesswerte. Die Rangliste kann auch erstellt werden, indem die Individuen vom Benutzer (siehe Kapitel 2.3.1) oder bei Fitnessintervallen mit Hilfe der probabilistischen Dominanz (siehe Kapitel 2.6.4) geordnet werden. Diese Selektionsmethode ist ebenfalls nicht duplikatfrei.
  (vgl. [Wei02] S. 72, [BHS07] S. 74f)
- Bei der **Turnierselektion** werden k Individuen aus der Population gleichverteilt ausgewählt. Diese Individuen werden miteinander verglichen und nur das beste darf sich reproduzieren. Der Vergleich kann auf Basis reeller Fitnesswerte oder der probabilistischen Dominanz (siehe Kapitel 2.6.4) erfolgen. Es wäre auch denkbar, dass dem Benutzer k Individuen präsentiert werden und dieser den „Turniersieger“ auswählt. Die Turnierselektion ist nicht duplikatfrei. (vgl. [Wei02] S. 72, [BHS07] S. 75)

Die Selektionsmethoden können mit der *Elite-Strategie* kombiniert werden, wobei das beste Individuum der Population immer in die nächste Generation übernommen wird (vgl. [BHS07] S. 75). Die Elite-Strategie soll bei der *Sparse Fitness Evaluation* nur auf sicheren Bewertungen basieren, also nur direkt evaluierte Elite-Individuen auswählen.

In Kapitel 2.6.4 wurde bereits angesprochen, dass die fitnessproportionale Selektion nicht bei Fitnessintervallen anwendbar ist, da keine reellen Fitnesswerte existieren. [GG07] schlägt die Turnierselektion vor.

3.3.7. Initialisierung

Zur Erzeugung der Startpopulation eines evolutionären Algorithmus sind folgende Methoden denkbar:

- Die Individuen werden zufällig erzeugt. Werden dabei Randbedingungen verletzt, so können die in Kapitel 3.3.4 vorgestellten Methoden zur Erzeugung gültiger Individuen verwendet werden. Bei Individuen mit einer variablen Struktur werden die Struktur und die einzelnen Werte zufällig erzeugt. (vgl. [Wei02] S. 44)
- Die Individuen der Startpopulation können durch das Optimierungsproblem vorgegeben werden (vgl. [Wei02] S. 44). Dies trifft bei der Zielstellung dieser Arbeit nicht zu.
- Die Individuen der Startpopulation können durch den Benutzer vorgegeben werden. Der Benutzer muss zunächst Einrichtungspläne anlegen, wodurch mehr Interaktionen erforderlich sind. Eine manuelle Erzeugung von Einrichtungsplänen ist in der Zielstellung ausdrücklich nicht gewünscht.
- Bei wiederkehrenden Optimierungsproblemen können die Ergebnisse vorangegangener Optimierungen verwendet werden (vgl. [Wei02] S. 44). Für die Initialisierung einer Population der Größe n werden m Individuen aus dem vorangegangenen Durchläufen übernommen, wobei gilt $m \leq n$. Ist die gewünschte Populationsgröße danach noch nicht erreicht, können die restlichen $n - m$ Individuen zufällig erzeugt werden. Durch diese Methode wird eine Makropopulation P_M eingeführt, die nach jedem Optimierungsdurchlauf aktualisiert und persistent gespeichert wird. Sie enthält Individuen mehrerer Durchläufe enthalten und hat eine definierte Maximalgröße. Wird diese Größe beim Hinzufügen der k besten Individuen des aktuellen Durchlaufs überschritten, werden die ältesten Individuen entfernt.

3.3.8. Terminierung

Am Ende des evolutionären Zyklus wird überprüft, ob ein bestimmtes Abbruchkriterium erfüllt ist und das Ziel der Optimierung bereits erreicht wurde. Bei interaktiven evolutionären Algorithmen kann die Terminierung auf folgende Arten erfolgen:

- Der Benutzer gibt das Ziel der Optimierung vor. Sobald dieses Ziel erreicht ist, terminiert der Algorithmus.
- Der Algorithmus terminiert nach einer vorgegebenen Zeit.
- Der Algorithmus terminiert nach einer vorgegebenen Anzahl von Generationen.
- Der Algorithmus überwacht den Verlauf der durchschnittlichen Fitness der Population. Wenn diese sich über mehrere Generationen nicht verbessert oder sogar verringert, terminiert der Algorithmus.
- Manueller Abbruch. Der Benutzer teilt dem Programm mit, dass das gewünschte Ergebnis erreicht ist und beendet so den Algorithmus.

Für die hier zu entwickelnde Anwendung soll der manuelle Abbruch umgesetzt werden.

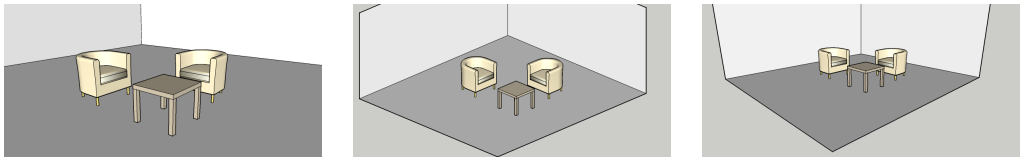
3.4. Benutzerschnittstelle

3.4.1. Ein- und Ausgabemodell

In Kapitel 2.3.3 wurden bereits verschiedene Modelle vorgestellt, mit denen eine Menge von Individuen dargestellt wird und vom Benutzer implizit oder explizit bewertet werden kann. Für die Zielstellung dieser Arbeit werden folgende Entscheidungen getroffen:

Individuen sollen explizit vom Benutzer bewertet werden. Gemäß der Empfehlung aus [Tak01] soll der Benutzer in fünf Stufen bewerten können (siehe Kapitel 2.4). Dazu wird das Kachelmodell verwendet. Es soll immer nur ein Individuum angezeigt werden, so dass der Benutzer sich ganz auf dessen Details konzentrieren kann. Die Forderung nach einer problemangepassten Darstellung

wurde bereits in Kapitel 2.3.2 formuliert. Daher sollen die Einrichtungspläne dreidimensional dargestellt werden.



(a) Freie Perspektive (b) Parallelprojektion (c) Vogelperspektive

Abbildung 3.1.: Perspektiven zur Darstellung eines Einrichtungsplans

Die möglichen Perspektiven sind in Abbildung 3.1 dargestellt. Die freie Perspektive in Abbildung 3.1(a) vermittelt dem Benutzer den Eindruck, er würde direkt im Raum stehen. Dies kommt der natürlichen Perspektive am nächsten. Diese Perspektive ist jedoch ungeeignet, da sie keine Übersicht über den gesamten Raum bietet. Die Parallelprojektion in Abbildung 3.1(b) bietet eine Übersicht über den ganzen Raum. Die Projektionsstrahlen verlaufen parallel, was jedoch auf den Benutzer unnatürlich wirken kann. Die Vogelperspektive in Abbildung 3.1(c) bietet einen Fluchtpunkt und kommt damit einer natürlichen Draufsicht am nächsten. Eine Übersicht über den ganzen Raum ist ebenfalls möglich.

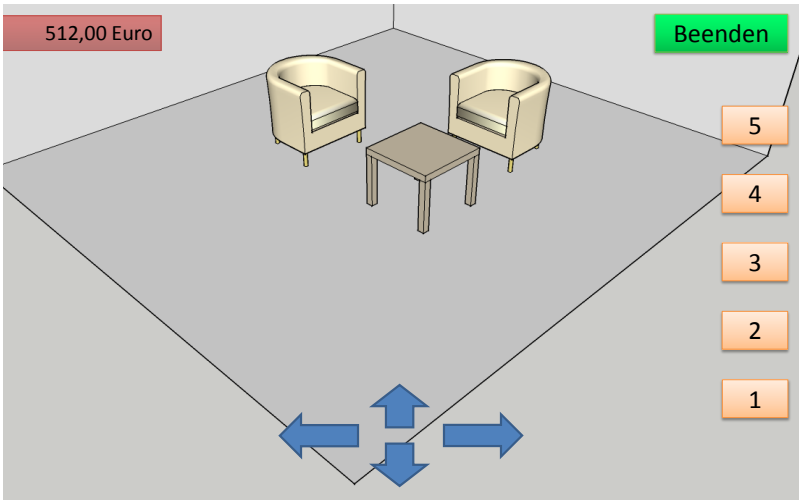


Abbildung 3.2.: Konzept der Benutzerschnittstelle

Für die Darstellung wird die Vogelperspektive gewählt. Abbildung 3.2 zeigt das Konzept der Benutzerschnittstelle. Um die Inneneinrichtung aus mehreren Richtungen betrachten zu können, soll der Benutzer den Betrachtungswinkel durch Ziehen mit der Maus verändern können. Alternativ soll dies auch durch

anklickbare Pfeile im unteren Teil des Bildschirms möglich sein. Die Kosten einer Einrichtung sollen dem Benutzer jederzeit in der linken oberen Ecke angezeigt werden. Durch Anklicken der Schaltflächen rechts im Bild kann eine Bewertung von eins bis fünf Punkten vergeben werden. Wenn der Benutzer die gewünschte Inneneinrichtung gefunden hat, kann er durch Anklicken der Schaltfläche „Beenden“ den Algorithmus terminieren.

3.4.2. Grafikbibliothek

Die *jMonkeyEngine* ist eine in *Java* geschriebene *3D-Engine*, welche die Bibliothek *LWJGL*[3] zum Rendern verwendet und *OpenGL 2.0* bis *OpenGL 4.0* voll unterstützt. Das primäre Entwurfsparadigma der Engine, der *Szenengraph*, stellt einen hierarchischen Baum dar, in dem die Knoten basierend auf ihrer räumlichen Position angeordnet werden. Durch die Strukturierung einzelner Knoten in Bäumen können ganze Spielszenen zusammengestellt werden. Eine übergeordnete Struktur ist der *AppState*, durch den sich beispielsweise komplette Szenen, Spielabschnitte, Menüs oder Controller kapseln lassen. *AppState*s können beliebig aktiviert, deaktiviert oder kombiniert werden. (vgl. [Láz08], [JME12]) Da *jMonkeyEngine* auf *Java* basiert, ist eine gute Portierbarkeit gegeben.

3.5. Optimierung des Algorithmus

3.5.1. Konzept für die Sparse Fitness Evaluation

In Kapitel 2.6 wurde die Methode der *Sparse Fitness Evaluation* vorgestellt. Diese soll für den in dieser Arbeit vorgestellten Anwendungsfall umgesetzt werden. Die Clusterbildung soll auf Basis des *k-Means*-Algorithmus durchgeführt werden, da dieser ein häufig bewährtes Standard-Cluster-Verfahren darstellt. Zum Cluster-Verfahren des *IGA-LPS* aus [GYM08] existieren hingegen keine ausreichenden Erfahrungsberichte. Für die Fitnesszuweisung soll die Methode der Intervall-Fitness umgesetzt werden. Eine geeignete Funktion zur Berechnung des Abstands von Genotypen wurde bereits in Kapitel 3.3.3 beschrieben.

[3] http://www.lwjgl.org/

3.5.2. Zufallszahlengenerierung

Auf Zufallszahlen und Wahrscheinlichkeiten basierende Optimierungsverfahren, sogenannte *Monte-Carlo-Algorithmen*, stellen hohe Anforderungen an Pseudo-Zufallszahlen-Generatoren. Die Zufallszahlen sollten eine hohe Gleichverteilung aufweisen und eine Folge von Zufallszahlen darf sich nicht zu früh wiederholen. Des Weiteren sollte die Zufallszahlenfolge mit einem Minimum an verwendeten Ressourcen berechnet werden können. Außerdem sollten die Zahlen ausreichend zufällig sein, so dass keine Muster auftreten, die den Algorithmus beeinflussen könnten. Die Klasse `java.util.Random`[4] der *Java* Klassenbibliothek verwendet einen herkömmlichen Kongruenzgenerator, und erfüllt diese Voraussetzungen nur ungenügend. Der Mersenne-Twister-Algorithmus ist in fast allen Aspekten einem Kongruenzgenerator überlegen und daher dem Algorithmus aus der Klassenbibliothek vorzuziehen (vgl. [MN97], [WHDY05]).

[4] http://docs.oracle.com/javase/7/docs/api/java/util/Random.html

4. Realisierung

Dieses Kapitel beschäftigt sich mit der Umsetzung der Anforderungen und der technischen Realisierung der Software. Im Vordergrund stehen dabei die Eigenschaften der Architektur. Es wird Anhand von Code- und Konfigurationsbeispielen gezeigt, wie eigene evolutionäre Algorithmen umgesetzt werden können.

4.1. Architektur des Systems

Unter Berücksichtigung der Qualitätsattribute Modifizierbarkeit, Wiederverwendbarkeit und Testbarkeit wurde das Gesamtsystem in Teilkomponenten zerlegt. Die Pakete des Systems sind in Abbildung 4.1 dargestellt.

Externe Bibliotheken
Hier sind die wichtigsten externen Bibliotheken aufgelistet. **JME** beinhaltet alle Klassen, der zur Darstellung verwendeten *jMonkeyEngine*. Diese 3D-Engine basiert auf **LWJGL**, der Lightweight Java Game Library. LWJGL verwendet eine native **OpenGL**-Implementierung. **H2 Database** ist ein dateibasiertes relationales Datenbankmanagementsystem. Zur objektrelationalen Abbildung wird das **Hibernate**-Framework verwendet.

Eigene Komponenten der Software
Die Entitäten der Einrichtungsgegenstände und Annotationen, sowie Klassen zur Datenbankkommunikation sind in dem Paket `FurnDB` enthalten. Das Paket `GA` ist die Bibliothek für evolutionäre Algorithmen. `GA-View` beinhaltet die Benutzerschnittstellen für interaktive evolutionäre Algorithmen und benötigt die Pakete `GA` und `JME`. `FurnGA` erweitert das Paket `GA` um die Entitäten aus `FurnDB` und enthält spezialisierte Klassen von Individuen, genetischen Operatoren und Regeln. Im Paket `FurnyApp` werden `GA-View` und `FurnGA` zusammengeführt. Außerdem sind dort Klassen zur Datenbankadministration enthalten. Um die

Entkopplung der Pakete schon während der Entwicklung sicherzustellen, wurde für jedes dieser Pakete ein eigenes Eclipse-Projekt erstellt.

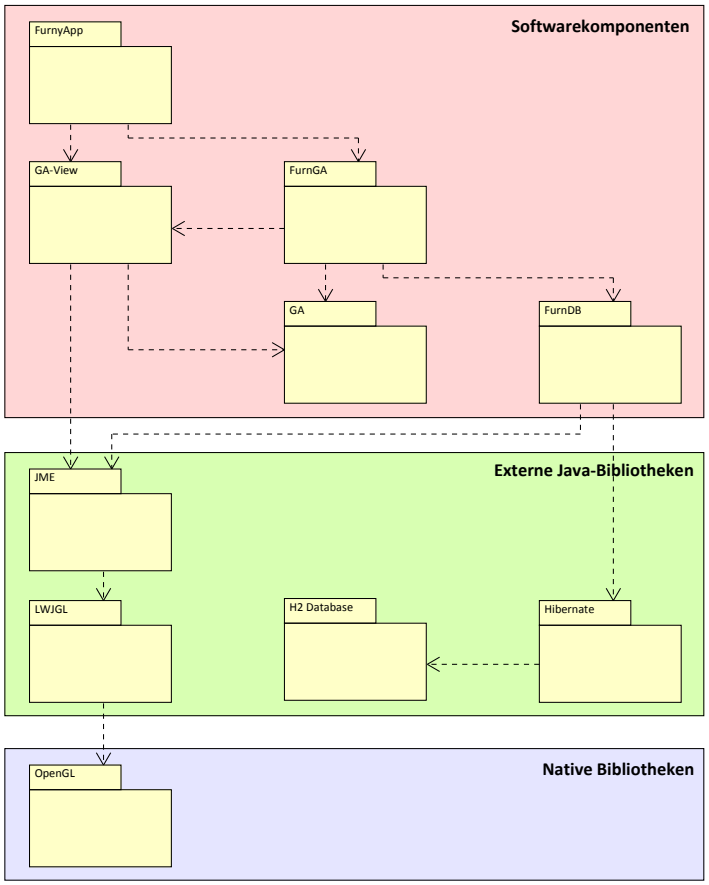


Abbildung 4.1.: Pakete des Systems

4.2. Bibliothek für evolutionäre Algorithmen

Als Grundlage der Software wurde eine eigenständige Bibliothek für evolutionäre Algorithmen entwickelt. Diese Entscheidung wurde getroffen, da bestehende Bibliotheken wie *ECJ*[5] oft nicht die Entwicklung interaktiver evolutionärer Algorithmen vorsehen und wenig Freiheiten in der Anpassung des Algorithmus bieten. Durch eine Eigenentwicklung einer solchen Bibliothek bestanden keine Einschränkungen beim Entwurf des Evolutionsprozesses. Außerdem konnten nicht standardisierte Methoden wie direkte Eingriffe in die

[5] http://cs.gmu.edu/~eclab/projects/ecj

Population, Clusterbildung und Fitnessintervalle ermöglicht werden. Generische Klassen und Schnittstellen wurden entwickelt, um die Implementierung von problemspezifischen Algorithmen mit wenig Aufwand zu ermöglichen.

Abbildung 4.2 zeigt die abstrakte Klasse `AbstractSIGA` sowie alle Schnittstellen der Bibliothek. Diese bilden die Grundlage zur Implementierung eines eigenen evolutionären Algorithmus. Die meisten Schnittstellen sind generisch mit dem Typ `<T extends IIndividual>` deklariert (nicht in der Grafik ersichtlich). Dadurch besteht während der Entwicklung eine Typensicherheit, so dass Methoden die jeweilige Unterklasse von `IIndividual` zurückgeben oder als Parameter akzeptieren.

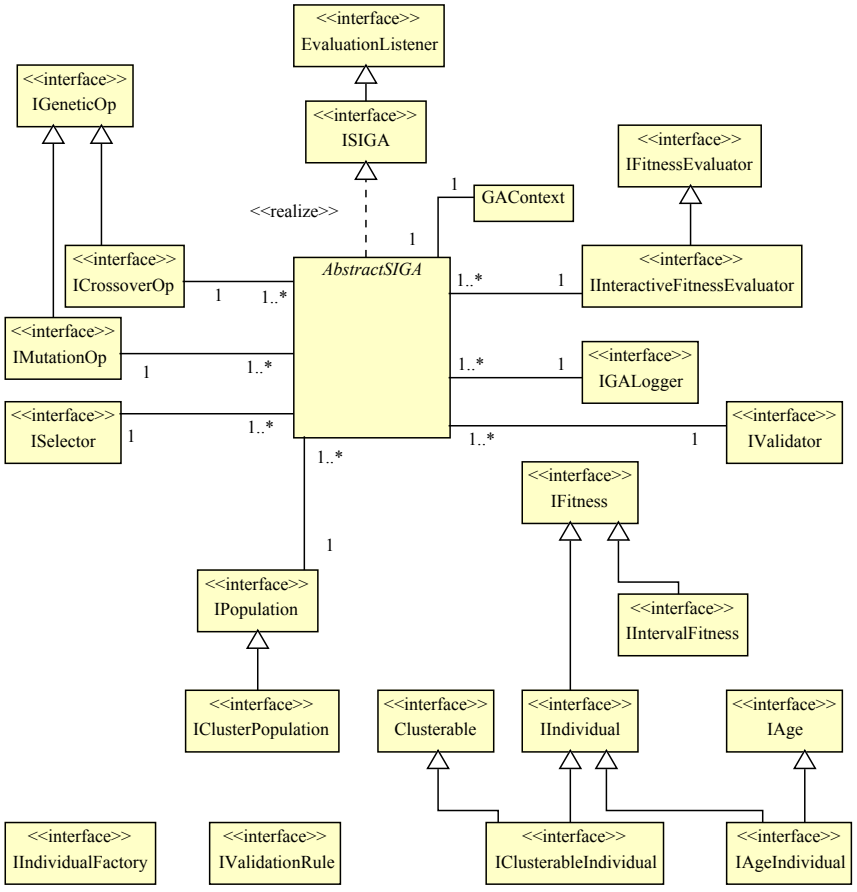


Abbildung 4.2.: Schnittstellen und abstrakter interaktiver evolutionärer Algorithmus der GA-Bibliothek

4.2.1. Individuen

Zur Implementierung eigener Individuen muss eine Klasse geschrieben werden, die eine oder mehrere der Schnittstellen `IIndividual` für ein normales Individuum, `IClusterableIndividual` für ein clusterbares Individuum oder `IAgeIndividual` für ein Individuum mit Alter implementiert. Außerdem kann die Schnittstelle `IIntervalFitness` implementiert werden, wenn das Individuum ein Fitnessintervall verwenden soll.

Listing 4.1 zeigt ein Individuum, das drei ganzzahlige Gene enthält. In dem angeführten Beispiel sollen für eine quadratische Funktion $x = a^2 \cdot b + c$ Werte für a, b und c gefunden werden, bei denen x einem vorgegebenen Wert x_{ziel} möglichst nahe kommt.

```
1  package ga.examples.quadratic;
2
3  import ga.core.algorithm.util.RandomSingleton;
4  import ga.core.individual.IIndividual;
5  import ga.core.validation.GAContext;
6
7  import java.util.ArrayList;
8  import java.util.List;
9  import java.util.Random;
10 import java.util.concurrent.atomic.AtomicLong;
11
12 public final class QuadraticIndividual implements
13     IIndividual<QuadraticIndividual> {
14   public static final int MAX_NUM = 8; // Wertebereich 0-8
15   public static final int PARAMETER_COUNT = 3; // 3 Parameter/Gene
16   private static final AtomicLong ID_GENERATOR = new AtomicLong();
17
18   private final long id; // Identifikationsnummer
19   private transient double fitness = UNEVALUATED; // Fitnesswert
20   private transient int result = Integer.MIN_VALUE; // Ergebnis der Evaluation
21   private final List<Integer> genotype = new ArrayList<Integer>(); // Genom
22
23   private final Random rnd = RandomSingleton.getRandom(); // Zufallsgenerator
24
25   public QuadraticIndividual() {
26     id = ID_GENERATOR.incrementAndGet();
27   }
28
29   @Override
30   public long getId() {
31     return id;
```

```
32  }
33
34  @Override
35  public void setContext(final GAContext context) {}
36
37  @Override
38  public void initRandomly() {
39    fitness = UNEVALUATED;
40    genotype.clear();
41
42    for (int i = 0; i < PARAMETER_COUNT; i++) {
43      genotype.add(rnd.nextInt(MAX_NUM + 1));
44    }
45  }
46
47  @Override
48  public void setFitness(final double fitness) {
49    this.fitness = fitness;
50  }
51
52  @Override
53  public double getFitness() {
54    return fitness;
55  }
56
57  public List<Integer> getGenotype() {
58    return genotype;
59  }
60
61  @Override
62  public String toString() {
63    String s = genotype.get(0) + "^2*" + genotype.get(1) + "+"
64        + genotype.get(2);
65
66    s += isEvaluated() ? ("=" + result + " Fitness: " + fitness) : " unevaluated";
67    return s;
68  }
69
70  public void setResult(final int result) {
71    this.result = result;
72  }
73
74  public int getResult() {
75    return result;
76  }
77
78  @Override
79  public boolean isEvaluated() {
```

```
80      return fitness != UNEVALUATED;
81    }
82
83    @Override
84    public QuadraticIndividual clone() {
85      final QuadraticIndividual ind = new QuadraticIndividual();
86      ind.genotype.addAll(this.genotype);
87      return ind;
88    }
89  }
```

Listing 4.1: Beispielimplementierung eines Individuums

4.2.2. Population

Die Population kann unter Angabe des generischen Typs instanziiert werden. In der Bibliothek gibt es die Klasse `ArrayListPopulation`, die die Schnittstelle `IPopulation` implementiert und zur Speicherung von Individuen eine `ArrayList`[6] verwendet. Des Weiteren gibt es die Klasse `KMeansClusterPopulation`, die `IClusterPopulation` implementiert. Sie verwendet den `KMeansPlusPlusClusterer` aus der *Apache Commons Math*[7] Bibliothek. `KMeansClusterPopulation` erfordert, dass die Individuen die Schnittstelle `IClusterableIndividual` implementieren. Sind diese Bedingungen erfüllt, so führt der Algorithmus eine Clusterbildung mittels *k-Means++* (vgl. [KMN+02]) und eine Fitnesszuweisung mit Fitness-Intervallen durch. Zur Instanziierung beider Arten von Populationen muss eine `IIndividualFactory` übergeben werden. Diese ist zur Erzeugung neuer Individuen notwendig. Als Referenzimplementierung steht die Klasse `TemplateIndividualFactory` zur Verfügung. Sie erzeugt neue Individuen, indem sie Klone des Template-Individuums zurückgibt. Das Template muss dem Konstruktor übergeben werden. Listing 4.2 zeigt, wie eine einfache Population mit 20 Individuen erzeugt wird.

```
1  final int populationSize = 20;
2  final IIndividualFactory<QuadraticIndividual> factory = new
       TemplateIndividualFactory<QuadraticIndividual>(
3        new QuadraticIndividual());
```

[6] http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html
[7] http://commons.apache.org/math/

```
4   final ArrayListPopulation<QuadraticIndividual> population = new
        ArrayListPopulation<QuadraticIndividual>(
5        factory, populationSize);
```

Listing 4.2: Beispiel der Instanziierung einer Population

4.2.3. Genetische Operatoren

Die genetischen Operatoren müssen problemangepasst implementiert werden. Dazu muss von der Operator-Klasse die Schnittstelle `IMutationOp` für die Mutation und `ICrossoverOp` für das Crossover implementiert werden. Die neue Operator-Klasse kann von der abstrakten Hilfsklasse `PropabilityOp` abgeleitet werden, um deren Funktionen zu nutzen. Der Crossover-Operator für das oben genannte Beispiel ist in Listing 4.3 zu sehen. Dies ist die Implementierung eines 1-Punkt-Crossovers.

```
1  package ga.examples.quadratic;
2
3  import ga.core.goperators.ICrossoverOp;
4  import ga.core.goperators.ProbabilityOp;
5  import ga.core.individual.IndividualList;
6  import ga.core.validation.GAContext;
7
8  public class QuadraticOnePointCrossoverOp extends ProbabilityOp implements
9      ICrossoverOp<QuadraticIndividual> {
10
11   public QuadraticOnePointCrossoverOp(final int pCrossover) {
12     super(pCrossover);
13   }
14
15   @Override
16   public IndividualList<QuadraticIndividual> crossover(
17       final QuadraticIndividual individual1,
18       final QuadraticIndividual individual2, final GAContext context) {
19     final IndividualList<QuadraticIndividual> list = new IndividualList<
         QuadraticIndividual>(); // Liste für die neuen Individuen
20
21     // Kopien der alten Individuen erzeugen
22     final QuadraticIndividual ind1 = individual1.clone();
23     final QuadraticIndividual ind2 = individual2.clone();
24
25     // wenn Operatorwahrscheinlichkeit getroffen wurde,
26     // also random.nextInt(100) < pCrossover
```

```
27    if (doOperate()) {
28
29      // zufälligen Schnittindex erzeugen
30      final int iCut = getRandom().nextInt(QuadraticIndividual.PARAMETER_COUNT);
31
32      // alle Gene vor dem Schnittindex tauschen
33      for (int i = 0; i < iCut; i++) {
34        ind2.getGenotype().set(i,
35            ind1.getGenotype().set(i, ind2.getGenotype().get(i)));
36      }
37    }
38
39    // neue Individuen zur Liste hinzufügen
40    list.add(ind1);
41    list.add(ind2);
42
43    return list;
44  }
45 }
```

Listing 4.3: Beispielimplementierung eines Crossover-Operators

Der Mutations-Operator für das Beispiel ist in Listing 4.4 zu sehen. Dies ist eine Punktmutation, bei der ein zufällig ausgewähltes Gen einen zufälligen Wert innerhalb des Wertebereichs erhält.

```
1  package ga.examples.quadratic;
2
3  import ga.core.goperators.IMutationOp;
4  import ga.core.goperators.ProbabilityOp;
5  import ga.core.validation.GAContext;
6
7  public class QuadraticPointMutationOp extends ProbabilityOp implements
8      IMutationOp<QuadraticIndividual> {
9
10   public QuadraticPointMutationOp(final int pMutate) {
11     super(pMutate);
12   }
13
14   @Override
15   public QuadraticIndividual mutate(final QuadraticIndividual individual,
16       final GAContext context) {
17   // Kopie des alten Individuums erzeugen
18     final QuadraticIndividual newInd = individual.clone();
19
20     // wenn Operatorwahrscheinlichkeit getroffen wurde,
21     // also random.nextInt(100) < pMutate
```

```
22      if (doOperate()) {
23        // Mutationsindex zufällig erzeugen
24        final int iMutate = getRandom().nextInt(
25            QuadraticIndividual.PARAMETER_COUNT);
26
27        // neuen Wert im Wertebereich erzeugen
28        final int newNum = getRandom().nextInt(QuadraticIndividual.MAX_NUM);
29        newInd.getGenotype().set(iMutate, newNum);
30      }
31
32      return newInd;
33    }
34  }
```

Listing 4.4: Beispielimplementierung eines Mutations-Operators

Die Klassenhierarchie der Beispiel-Operatoren mit Methoden und Attributen ist in Abbildung 4.3 dargestellt.

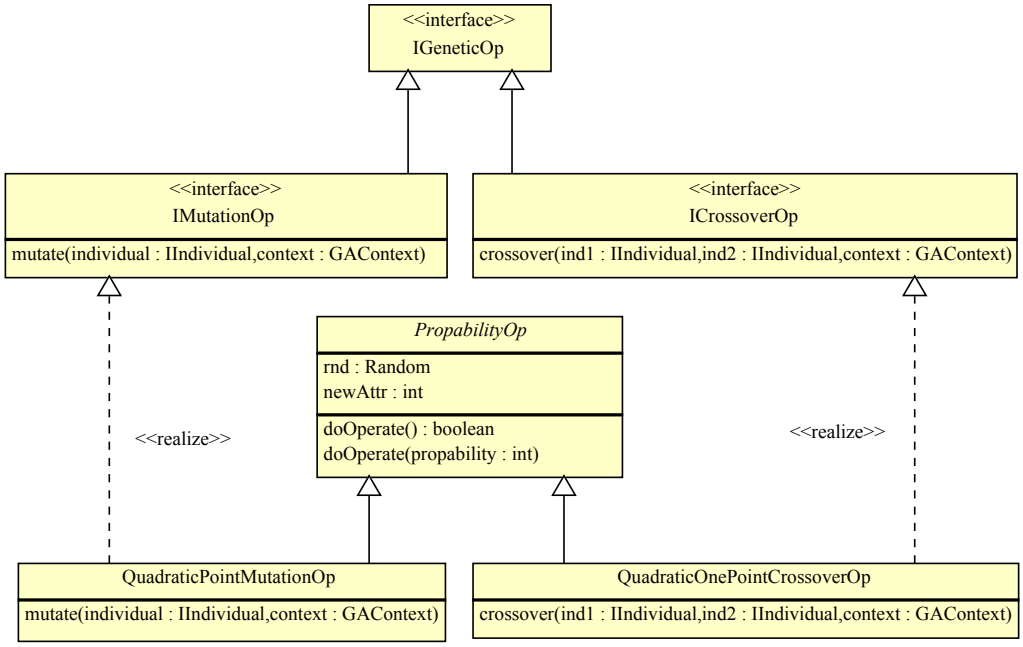


Abbildung 4.3.: Genetische Operatoren des Beispiels

4.2.4. Selektion

Der Selektions-Operator muss nicht problemangepasst implementiert werden und kann unter Angabe des generischen Typs instanziiert werden. Die Biblio-

thek bietet derzeit nur eine Turnierselektion an. Diese wird in Listing 4.5 für das Beispiel mit einer Turniergröße von 3 initialisiert.

```
1  final int tournamentSize = 3;
2  TournamentSelector<QuadraticIndividual> selector =
3      new TournamentSelector<QuadraticIndividual>(tournamentSize);
```

Listing 4.5: Beispiel der Instanziierung eines Selektions-Operators

4.2.5. Evaluation

Zur Bestimmung der Fitness von Individuen muss die Schnittstelle `IFitnessEvaluator` für automatische oder `IInteractiveFitnessEvaluator` für interaktive evolutionäre Algorithmen in einer Klasse realisiert werden. Während der automatische Algorithmus mit `IFitnessEvaluator` blockierend arbeitet und auf das Ergebnis der Fitnessbestimmung wartet, ist bei dem interaktiven `IInteractiveFitnessEvaluator` der Zeitpunkt der Bewertung nicht festgelegt. Daher deklariert die Schnittstelle Methoden, um `EvaluationListener` hinzuzufügen und zu entfernen. Der Algorithmus kann sich als Listener beim *Evaluator* anmelden, um informiert zu werden, sobald ein Individuum bewertet wurde. Er muss außerdem darüber informiert werden, wenn der interaktive Evaluator ein neues Individuum anzeigen will. So ist sichergestellt, dass der Evaluator nicht die konkrete Klasse des Algorithmus kennen muss, jedoch über eine schmale Schnittstelle mit diesem kommunizieren kann.

Für die Beispielanwendung wurde ein automatischer Evaluator implementiert. Dieser ist in Listing 4.6 zu sehen.

```
1  package ga.examples.quadratic;
2
3  import ga.core.evaluation.IFitnessEvaluator;
4
5  public class QuadraticEvaluator implements
6      IFitnessEvaluator<QuadraticIndividual> {
7
8    private final int target;
9
10   public QuadraticEvaluator(final int target) {
11     this.target = target;
```

```
12   }
13
14   public int getTarget() {
15     return target;
16   }
17
18   @Override
19   public void evaluate(final QuadraticIndividual individual) {
20   // die Gene a, b und c holen
21     final int a = individual.getGenotype().get(0);
22     final int b = individual.getGenotype().get(1);
23     final int c = individual.getGenotype().get(2);
24
25     // das Ergebnis berechnen
26     final int result = (int) Math.pow(a, 2) * b + c;
27
28     // normierte Fitness berechnen
29     final double fitness = 1d / Math.max(Math.abs(result - target), 1);
30
31     // Ergebnis und Fitness im Individuum speichern
32     individual.setResult(result);
33     individual.setFitness(fitness);
34   }
35 }
```

Listing 4.6: Beispielimplementierung eines automatischen Evaluators

4.2.6. Algorithmen

Die Bibliothek stellt vier verschiedene Algorithmen zur Verfügung:

- `SGAGeneration` ist der klassische generationsbasierte *Simple Genetic Algorithm*.
- `SGA` ist die Implementierung des *Simple Genetic Algorithm* mit *Steady-State*-Arbeitsweise.
- `SIGAGeneration` ist ein generationsbasierter interaktiver evolutionärer Algorithmus und eine Unterklasse von `AbstractSIGA`.
- `SIGA` ist ein interaktiver evolutionärer Algorithmus mit *Steady-State*-Arbeitsweise und eine Unterklasse von `AbstractSIGA`.

Jeder der vier Algorithmen unterstützt die Clusterbildung mit Fitnessintervallen. Dazu die Individuen `IClusterableIndividual` implementieren. Außerdem ist eine Population erforderlich, die `IClusterPopulation` implementiert.

Für die Beispielanwendung kann ein automatischer Algorithmus erzeugt und ausgeführt werden, wie in Listing 4.7 gezeigt wird.

```
1   final IIndividualFactory<QuadraticIndividual> factory =
2       new TemplateIndividualFactory<QuadraticIndividual>(new QuadraticIndividual
            ());
3   final ArrayListPopulation<QuadraticIndividual> population =
4       new ArrayListPopulation<QuadraticIndividual>(factory, 20);
5
6   // die quadratische Funktion soll das Ergebnis 42 liefern
7   final QuadraticEvaluator evaluator = new QuadraticEvaluator(42);
8   final TournamentSelector<QuadraticIndividual> selector = new TournamentSelector
        <QuadraticIndividual>(3);
9   final QuadraticOnePointCrossoverOp crossOverOp = new
        QuadraticOnePointCrossoverOp(25);
10  final QuadraticPointMutationOp mutationOp = new QuadraticPointMutationOp(5);
11
12  // automatischen genetischen Algorithmus erzeugen
13  final SGAGeneration<QuadraticIndividual> algorithm =
14      new SGAGeneration<QuadraticIndividual>(
15          population, evaluator, selector, mutationOp, crossOverOp);
16
17  // Algorithmus initialisieren
18  algorithm.init();
19
20  // 100 Generationen
21  for (int i = 0; i <= 100; i++) {
22    // Generation und ausgeben
23    System.out.println("Generation " + i);
24    // gesamte Population ausgeben
25    System.out.println(algorithm.getPopulation());
26    // Schritt ausführen, neue Generation erzeugen und bewerten
27    algorithm.step();
28
29    // bestes Individuum ausgeben
30    final QuadraticIndividual fittest = algorithm.getPopulation()
31        .getFittestIndividual();
32    System.out.println("Fittest: " + fittest);
33
34    // abbrechen, wenn das Ziel erreicht wurde
35    if (fittest.getResult() == evaluator.getTarget()) {
```

```
36          // Generation ausgeben, in der das Ziel erreicht wurde
37          System.out.println("Generation " + i);
38          break;
39        }
40      }
```

Listing 4.7: Beispiel zur Erzeugung und Ausführung eines SGA

4.2.7. Hilfsklassen

Die Hilfsklassen bieten für oft verwendete Funktionen statische Methoden an. Die Klasse `ClusterUtil` kapselt Hilfsmethoden für den Umgang mit Clustern. Die Methode `calculateCentroid(Collection<T> c)` berechnet das Zentrum eines Clusters, indem ein Objekt im Cluster gesucht wird, das zu allen anderen Objekten im Cluster den geringsten Abstand hat. Die Methode `assignFitness(List<Cluster<T>> clusters, T ind)` führt die Zuweisung der Fitness aus. Wenn das übergebene Individuum die Schnittstelle `IIntervalFitness` implementiert, so wird eine Zuweisung von Fitnessintervallen durchgeführt (siehe Kapitel 2.6). Andernfalls erfolgt die direkte Zuweisung der Fitness. Die Klasse `PopulationUtil` bieten Hilfsfunktionen zur Berechnung der minimalen, mittleren und maximalen Fitness der Population. Die Klasse `RandomSingleton` ermöglicht eine gemeinsame Nutzung einer Instanz des `MersenneTwister`-Zufallsgenerators aus der *Apache Commons Math*[8] Bibliothek.

4.3. Datenbank für Einrichtungsgegenstände

Das Paket `FurnDB` ist die Persistenzschicht mit den Entitäten für Einrichtungsgegenstände und deren Metadaten. Der Katalog von Einrichtungsgegenständen wird durch eine lokale, dateibasierte Datenbank repräsentiert. Dazu wird das relationale Datenbankmanagementsystem *H2 Database*[9] verwendet. Die objektrelationale Abbildung wird durch das Framework *Hibernate*[10] durchgeführt.

[8] http://commons.apache.org/math/
[9] http://www.h2database.com
[10] http://www.hibernate.org/

4.3.1. Entitäten und Relationen

Abbildung 4.4 zeigt die Entitäten und Relationen des Pakets. Das sich daraus ergebende Modell wird auf von *Hibernate* auf Datenbanktabellen abgebildet.

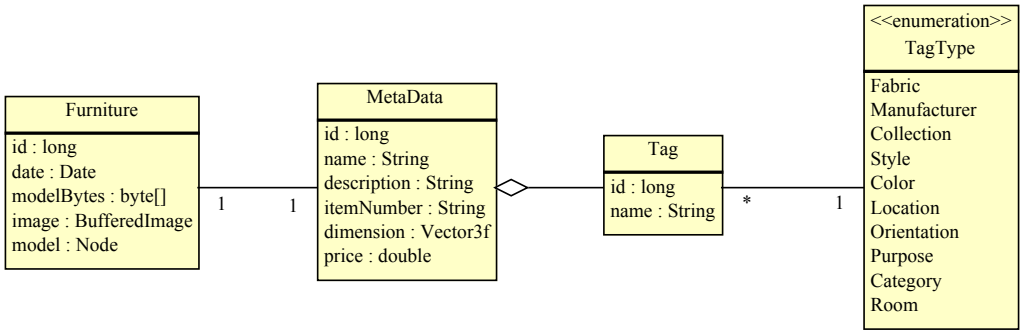


Abbildung 4.4.: Entitäten der Datenbank für Einrichtungsgegenstände

Die Klasse `Furniture` repräsentiert einen Einrichtungsgegenstand. Dieser hat jeweils ein Objekt der Klasse `MetaData`, das die Metadaten speichert und Objekte des Typs `Tag` aggregiert. Zu den Metadaten gehören der Name, eine Beschreibung, die Artikelnummer und der Preis. Mit `Tag`-Objekten werden die in Kapitel 3.3.3 vorgestellten Annotationen realisiert. Es können beliebig viele `Tag`-Objekte existieren. Jedes `Tag`-Objekte hat einen `TagType`, der den Schlüssel das Parameters repräsentiert. `TagType` ist ein Java `enum`, daher sind die Instanzen vordefiniert und nur programmatisch erweiterbar. Der Datentyp `Node` in der Klasse `Furniture` stammt aus der Grafik-API *jMonkeyEngine* und repräsentiert einen Knoten in einem Szenengraphen. Kind-Objekte eines `Node` können weitere `Node`-Objekte oder ein `Geometry`-Objekt sein, das ein Drahgittermodell enthält. Um die Speicherung dieser Modelle in einer relationalen Datenbank zu ermöglichen, werden sie serialisiert und als `byte`-Array zwischengespeichert. Dieses Array kann in der Datenbank als BLOB[11] gespeichert werden.

4.3.2. Datenzugriff

Für das Anlegen, Lesen, Aktualisieren und Löschen von Einrichtungsgegenständen wurde die Klasse `FurnDBManager` in Abbildung 4.5 implementiert.

[11] Datentyp in Datenbanksystemen und Akronym für „Binary Large Object“

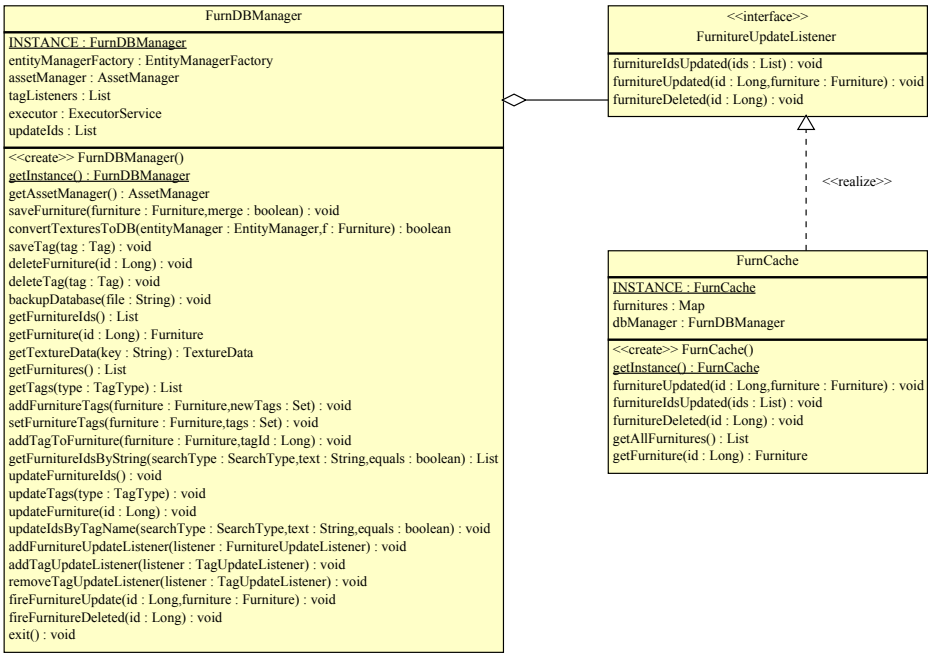


Abbildung 4.5.: Klassen zum Datenbankzugriff

Außerdem bietet sie Methoden zum Hinzufügen und Entfernen und Benachrichtigen von Listenern an. Die Klasse `FurnCache` beschleunigt den Zugriff auf die vorhandenen `Furniture`-Objekte, in dem diese einmal angefragt und im Arbeitsspeicher gehalten werden. Da diese Methode `FurnitureUpdateListener` implementiert und sich als Listener beim `FurnDBManager` registriert, wird sie auch über Änderungen an `Furniture`-Objekten informiert, die aus anderen Programmteilen kommen. Somit sind die vorgehaltenen Entitäten immer aktuell.

4.3.3. Import von 3D-Modellen

Zum Anlegen eines neuen Einrichtungsgegenstandes muss ein 3D-Modell importiert werden. Die derzeit größte freie Datenbank solcher Modelle ist das *Google 3D Warehouse*[12]. Es ermöglicht, 3D-Modelle nach verschiedenen Kriterien zu suchen und kostenlos herunterzuladen. Eine Anmeldung ist dafür nicht erforderlich. Angemeldete Benutzer können darüber hinaus ihre eigenen Modelle hochladen und Kollektionen erstellen. Das Format für den Datenaustausch sind *skp*-Dateien. Diese können mit der Software *Google SketchUp*[13]

[12] http://sketchup.google.com/3dwarehouse/
[13] http://sketchup.google.com/

erstellt, gespeichert, geöffnet und bearbeitet werden. Die angebotenen Modelle sind meist maßstabsgetreu. Andernfalls können sie in wenigen Schritten mit *SketchUp* skaliert werden.

Ein weit verbreitetes Datenformat für 3D-Modelle im Open Source-Anwendungsbereich ist das offene XML-Format der 3D-Grafik-Bibliothek *Ogre*[14], kurz *OgreXML* . Da dieses Format von *jMonkeyEngine* gut unterstützt wird, wurde es auch für dieses Projekt gewählt.

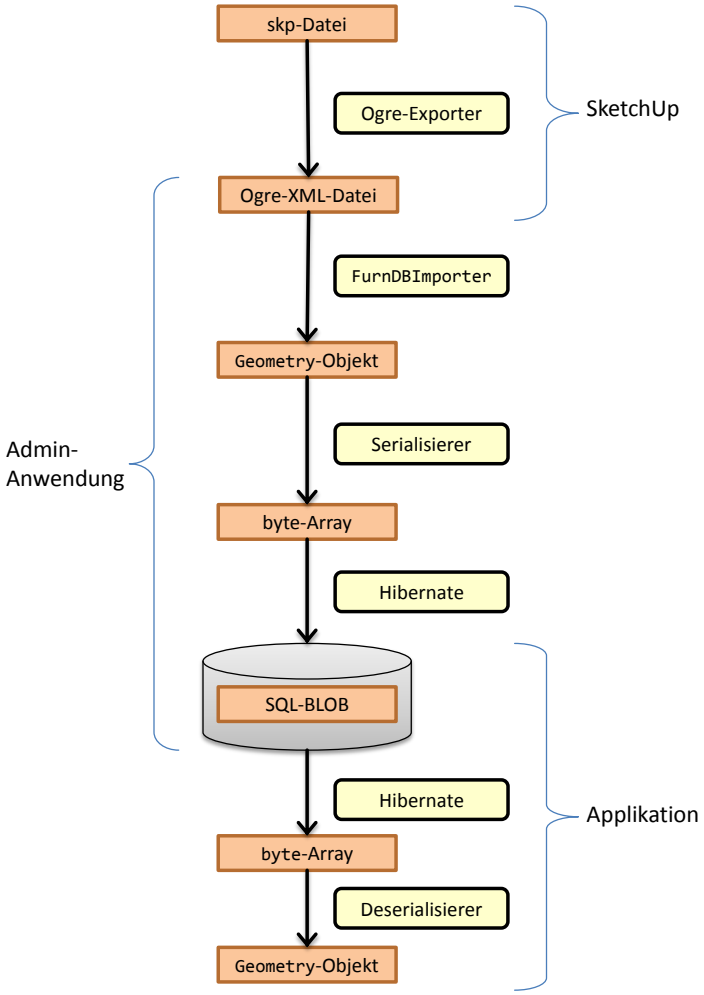


Abbildung 4.6.: Workflow für den Import von 3D-Modellen

Abbildung 4.6 zeigt den Workflow für den Import von 3D-Modellen. Das Eingabeformat ist eine *skp*-Datei, die heruntergeladen oder auch selbst erstellt wurde. Diese wird vom Benutzer in *SketchUp* geladen und mit Hilfe eines

[14] http://www.ogre3d.org/

Plugins als *OgreXML* exportiert. Die Klasse `FurnDBImporter` liest diese Datei ein und erzeugt eine `Geometry`-Instanz, welche Hierarchie, Drahtgittermodell und Texturen des Modells enthält. Ein Serialisierer erzeugt aus dem Objekt ein `byte`-Array, das anschließend von *Hibernate* als *SQL-BLOB* in der Datenbank gespeichert wird. Eine Applikation, die dieses Modell nutzen will, kann das `byte`-Array von *Hibernate* aus der Datenbank laden lassen. Anschließend wird es in ein `Geometry`-Objekt deserialisiert und kann in *jMonkeyEngine* verwendet werden. Die wichtigsten mit `FurnDBImporter` assoziierten Klassen und Schnittstellen sind in Abbildung 4.7 dargestellt.

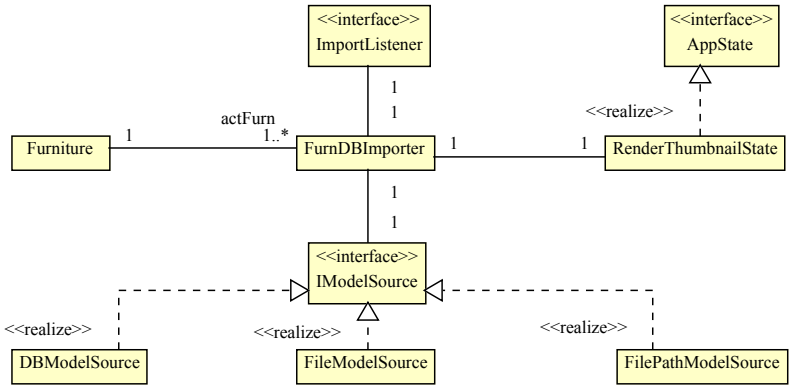


Abbildung 4.7.: Klassen und Schnittstellen zum Import von 3D-Modellen

Die zu importierenden Modelle kommen aus einer der Quellen, die `IModelImporter` implementieren. `DBModelSource` holt die Modelle aus einer bereits bestehenden Datenbank, `FileModelSource` importiert eine einzelne *OgreXML* -Datei und `FilePathModelSource` alle *OgreXML* -Dateien aus einem gegebenen Verzeichnis. Durch den Import wird ein Objekt der Klasse `Furniture` erstellt und anschließend ein Vorschau-Bild gerendert, welches im `Furniture`-Objekt gespeichert wird. Über einen `ImportListener` kann sich eine Komponente registrieren und Benachrichtigungen empfangen, wenn der Import abgeschlossen ist.

4.4. Benutzerschnittstelle

Das Paket `GA-View` erweitert die Bibliothek `GA` um eine dreidimensionale Darstellung mittels *jMonkeyEngine* (siehe Kapitel 3.4.2). Es enthält Evaluatoren zur Darstellung der Phänotypen und zur Eingabe der Fitnesswerte.

4.4.1. Evaluatoren

Es wurden drei Benutzerschnittstellen implementiert, welche die Schnittstelle `IInteractiveFitnessEvaluator` realisieren. Das vereinfachte Klassendiagramm in Abbildung 4.8 zeigt, wie diese Evaluatoren an die *jMonkeyEngine* angebunden werden.

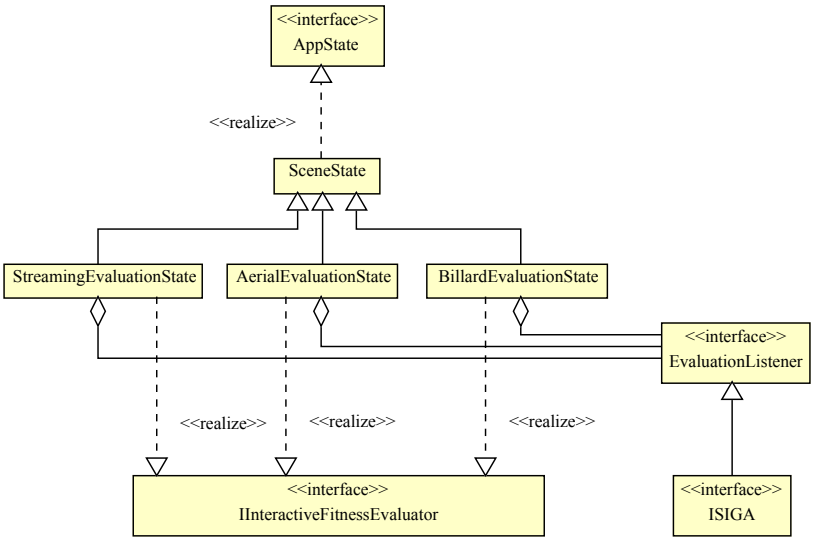


Abbildung 4.8.: Klassen und Schnittstellen zur Anbindung der *jMonkeyEngine* an den evolutionären Algorithmus

Die Schnittstelle `AppState` aus der *jMonkeyEngine*-API wird durch die Klasse `SceneState` implementiert, die wiederum als Basisklasse für die Evaluatoren dient. Jeder Evaluator informiert über einen `EvaluationListener` den interaktiven evolutionären Algorithmus, der die Schnittstelle `ISIGA` implementieren muss. Wenn es erforderlich sein sollte, können sich beliebig viele `EvaluationListener` bei einem Evaluator anmelden. Die Klasse `StreamingEvaluationState` ist eine Implementierung des dreistufigen Strom-Modells aus [Fis08], welches in Kapitel 2.3.3 genauer erläutert wurde. Die Klasse `AerialEvaluationState` setzt die Benutzerschnittstelle um, die in Kapitel 3.4.1 entwickelt wurde. Das Billardtisch-Modell aus [Dre11] wurde in der Klasse `BillardEvaluationState` umgesetzt.

4.4.2. Ablauf des interaktiven evolutionären Algorithmus

Abbildung 4.9 zeigt den Ablauf eines generationsbasierten interaktiven evolutionären Algorithmus. Der Evaluator informiert nach dem Start den Algorithmus über einen Listener, dass er bereit ist, ein Individuum anzuzeigen. Sobald der Algorithmus die Initialisierung abgeschlossen hat, wählt er ein beliebiges Individuum, das noch nicht bewertet wurde aus und erzeugt den Phänotyp dazu. Anschließend wird der Evaluator über einen Listener darüber informiert,

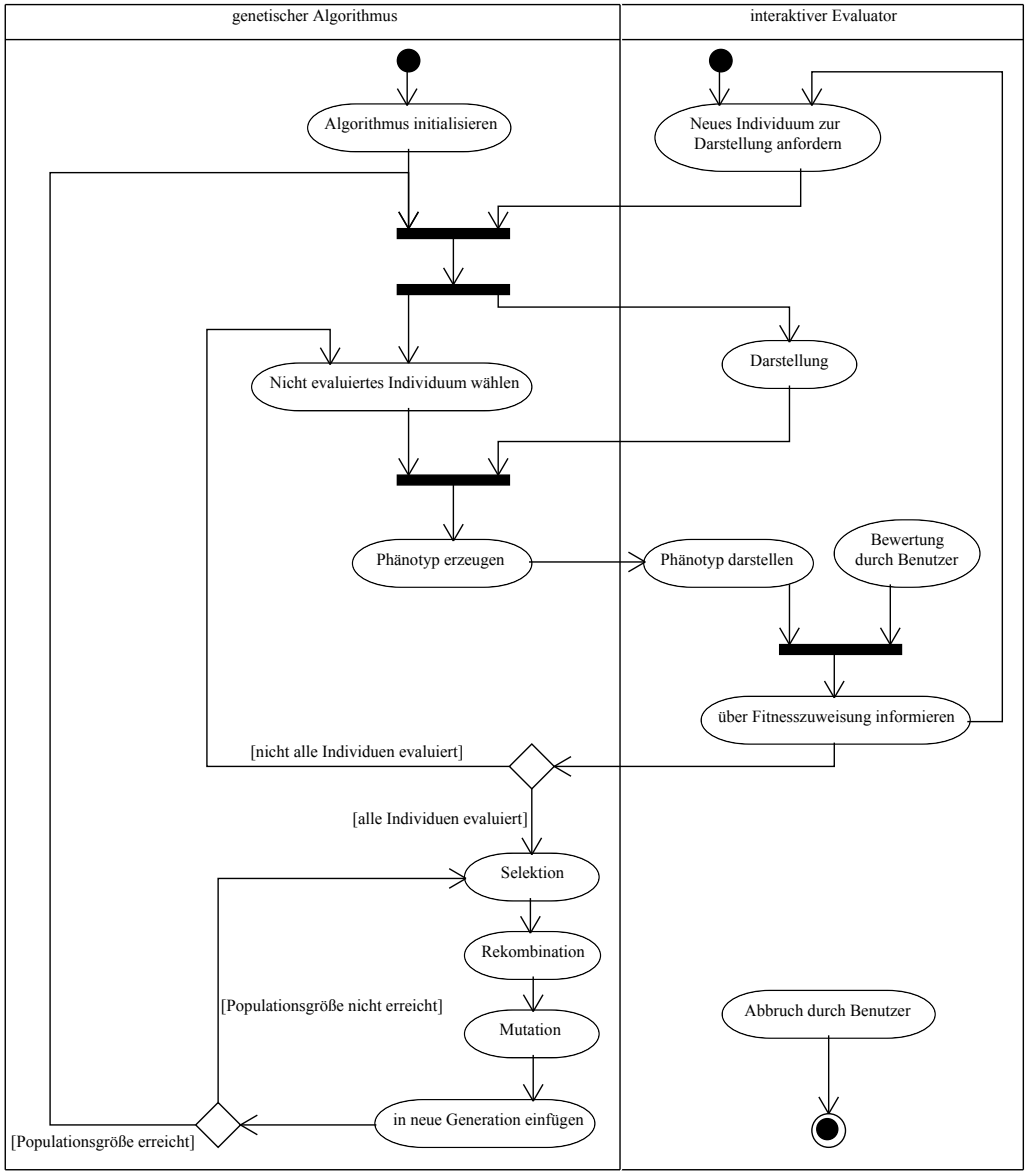


Abbildung 4.9.: Aktivitätsdiagramm eines interaktiven evolutionären Algorithmus

dass dieser Phänotyp angezeigt werden kann. Sobald der Benutzer zu diesem Phänotyp eine Bewertung abgegeben hat, wird geprüft ob alle Individuen bewertet wurden. Wenn dies nicht der Fall ist, wird ein weiteres Individuum ausgewählt, angezeigt und bewertet. Wenn alle Individuen bewertet wurden, wird Selektion, Rekombination, Mutation und Einfügeselektion durchgeführt, bis die neue Generation die eingestellte Populationsgröße erreicht. Nach diesem Schritt beginnt ein neuer Durchlauf des evolutionären Zyklus. Der Benutzer kann jederzeit das Programm beenden.

4.4.3. XML-Konfiguration

Zur Definition des Algorithmus sowie zur Konfiguration aller Parameter wurde eine XML-Schnittstelle implementiert. Das Programm kann einer XML-Konfigurationsdatei gestartet werden und bezieht alle nötigen Informationen daraus. Zur Definition der XML-Datei wurde ein XML Schema entwickelt. Bei der Entwicklung neuer Konfigurationen mit Hilfe eines XML-Editors ist das Schema hilfreich, da falsche Eingaben so weit wie möglich ausgeschlossen werden. Das Schema befindet sich in Anhang B.

In Listing 4.8 ist eine Beispiel-Konfiguration aufgeführt. Die Kommentare erklären die Funktionen der einzelnen Elemente.

```
<?xml version="1.0" encoding="UTF-8"?>
<gasettings xmlns="http://www.example.org/gasettings"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/gasettings gasettings.xsd">

  <!-- Konfiguration eines Algorithmus -->
  <setting>
    <!-- Allgemeine Parameter sowie Parameter, die spaeter referenziert werden -->
    <values>
      <!-- Name des Algorithmus -->
      <simpleValue id="name" type="string" value="Simple Object Streaming" />
      <!-- das Icon, welches im Hauptmenue angezeigt wird -->
      <simpleValue id="iconImage" type="string"
        value="ga/view/example/resource/model_streaming_icon.png" />
      <!-- Mutationswahrscheinlichkeit -->
      <simpleValue id="pMutate" type="int" value="50" />
      <!-- Crossover-Wahrscheinlichkeit -->
      <simpleValue id="pCrossover" type="int" value="25" />
      <!-- Populationsgroesse -->
```

```
20      <simpleValue id="populationInitSize" type="int" value="20" />
21      <!-- Turniergroesse -->
22      <simpleValue id="tournamentSize" type="int" value="3" />
23      <!-- Elite-Strategie verwenden -->
24      <simpleValue id="useEliteStrategy" type="boolean" value="true" />
25      <!-- Klasse des zu verwendenden Individuums -->
26      <objectValue id="individual"
27        class="ga.view.examples.simple.SimpleObjectIndividual" />
28    </values>
29    <!-- Klasse des Generators fuer Phaenotypen (Dekodierungsfunktion) -->
30    <phenotypeGenerator
31      class="ga.view.examples.simple.SimpleObjectPhenotypeGenerator" />
32    <!-- Klasse des interaktiven Evaluators -->
33    <evaluator class="ga.view.streaming.StreamingEvaluationState">
34      <!-- Parameter des Konstruktors der Klasse; muessen unter values vorher
            definiert werden -->
35      <objectParameter index="0" id="phenotypeGenerator" />
36    </evaluator>
37    <!-- Klasse fuer die Factory, die aus Templates neue Individuen erzeugt -->
38    <individualFactory class="ga.core.individual.TemplateIndividualFactory">
39      <objectParameter index="0" id="individual" />
40    </individualFactory>
41    <!-- Klasse fuer die Population -->
42    <population class="ga.core.individual.population.ArrayListPopulation">
43      <!-- Konstruktorparameter an Index 0 -->
44      <objectParameter index="0" id="individualFactory" />
45      <!-- Konstruktorparameter an Index 1 -->
46      <simpleParameter index="1" id="populationInitSize" />
47    </population>
48    <!-- Klasse fuer die Selektion -->
49    <selector class="ga.core.selection.TournamentSelector">
50      <simpleParameter index="0" id="tournamentSize" />
51    </selector>
52    <!-- Klasse fuer den Mutations-Operator -->
53    <mutationOp class="ga.view.examples.simple.SimpleObjectMutationOp">
54      <simpleParameter index="0" id="pMutate" />
55    </mutationOp>
56    <!-- Klasse fuer den Crossover-Operator -->
57    <crossoverOp class="ga.view.examples.simple.SimpleObjectCrossoverOp">
58      <simpleParameter index="0" id="pCrossover" />
59    </crossoverOp>
60    <!-- Klasse fuer den interaktiven genetischen Algorithmus -->
61    <ga class="ga.core.algorithm.interactive.SIGA">
62      <objectParameter index="0" id="population" />
63      <objectParameter index="1" id="evaluator" />
64      <objectParameter index="2" id="selector" />
65      <objectParameter index="3" id="mutationOp" />
66      <objectParameter index="4" id="crossoverOp" />
```

```
67      <simpleParameter index="5" id="useEliteStrategy" />
68    </ga>
69  </setting>
70
71  <!-- Hier koennten weitere setting-Tags kommen -->
72 </gasettings>
```

Listing 4.8: Beispielkonfiguration eines interaktiven evolutionären Algorithmus

4.5. Spezialisierter genetischer Algorithmus

Das Paket `FurnGA` beinhaltet die Implementierung eines genetischen Algorithmus für Einrichtungsgegenstände. Es ist abhängig von der Persistenzschicht mit den Entitäten `FurnDB`, von der Bibliothek für genetische Algorithmen `GA` und der Benutzerschnittstellen-Bibliothek `GA-View`.

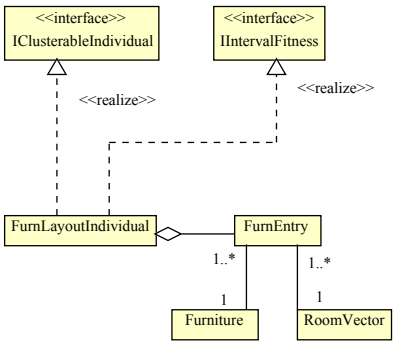


Abbildung 4.10.: Individuum für Raumpläne

Abbildung 4.10 zeigt die Klasse `FurnLayoutIndividual`, welche die Schnittstellen `IClusterableIndividual` und `IIntervalFitness` des *GA*-Pakets realisiert. Der Genotyp besteht aus Instanzen der Klasse `FurnEntry`, die jeweils eine Instanz von `Furniture` und `RoomVector` enthalten. Die Klasse `RoomVector` enthält die Gene für die X- und Y-Position, sowie die Anzahl der Rotationen. Das hier implementierte Individuum kann geclustert werden und unterstützt Fitnessintervalle. Dazu wurde die Abstandsfunktion aus Kapitel 3.3.3 implementiert.

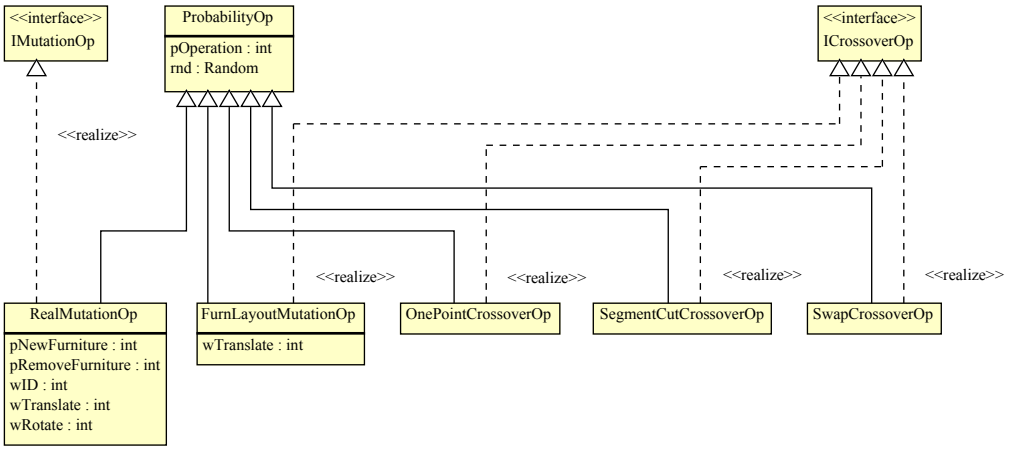


Abbildung 4.11.: Genetische Operatoren mit Attributen

Abbildung 4.11 zeigt alle implementierten genetischen Operatoren und die jeweiligen Attribute. Die Klassen `RealMutationOp`, `FurnLayoutMutationOp`, `OnePointCrossoverOp`, `SegmentCutCrossoverOp` und `SwapCrossoverOp` setzen die in Kapitel 3.3.5 beschriebenen Operatoren Reelle Mutation, Verschieben und Austauschen, 1-Punkt-Crossover, Segmentaustausch und Segment-Schnitt-Crossover um. Zu Testzwecken wurden die verschiedenen Crossover-Operatoren implementiert, wobei nur `SegmentCutCrossoverOp` in Kapitel 3.3.5.3 als geeignet bewertet wurde.

Einige der in Kapitel 3.3.4 vorgestellten Regeln wurden implementiert. Abbildung 4.12 zeigt die Klassen dazu. Jede Regel implementiert die Schnittstelle `IValidationRule`. Es gibt drei Typen von Regeln. `MANDATORY`-Regeln müssen zwingend erfüllt werden. Numerisch bedeutet dies, dass sie einen Grenzwert von 1,0 haben. Unterschreitet ein Individuum bei einer Regel den Grenzwert, so fällt das Individuum durch die Validierung durch. `PROPOSED`-Regeln haben einen Grenzwert von 0,5 und `PERFECTION`-Regeln einen Grenzwert von 0,3.

- Die Klasse `RoomContainsRule` setzt die Randbedingung um, dass sich alle Einrichtungsgegenstände innerhalb des Raums befinden müssen und auch keine Wand überschneiden. Die Erfüllung dieser Regel ist zwingend vorgeschrieben, daher ist sie vom Typ `MANDATORY`.
- Die Klasse `FurnIntersectionRule` setzt die Randbedingung um, dass sich kein Einrichtungsgegenstand mit einem anderen überschneiden darf. Diese Regel ist ebenfalls vom Typ `MANDATORY`.

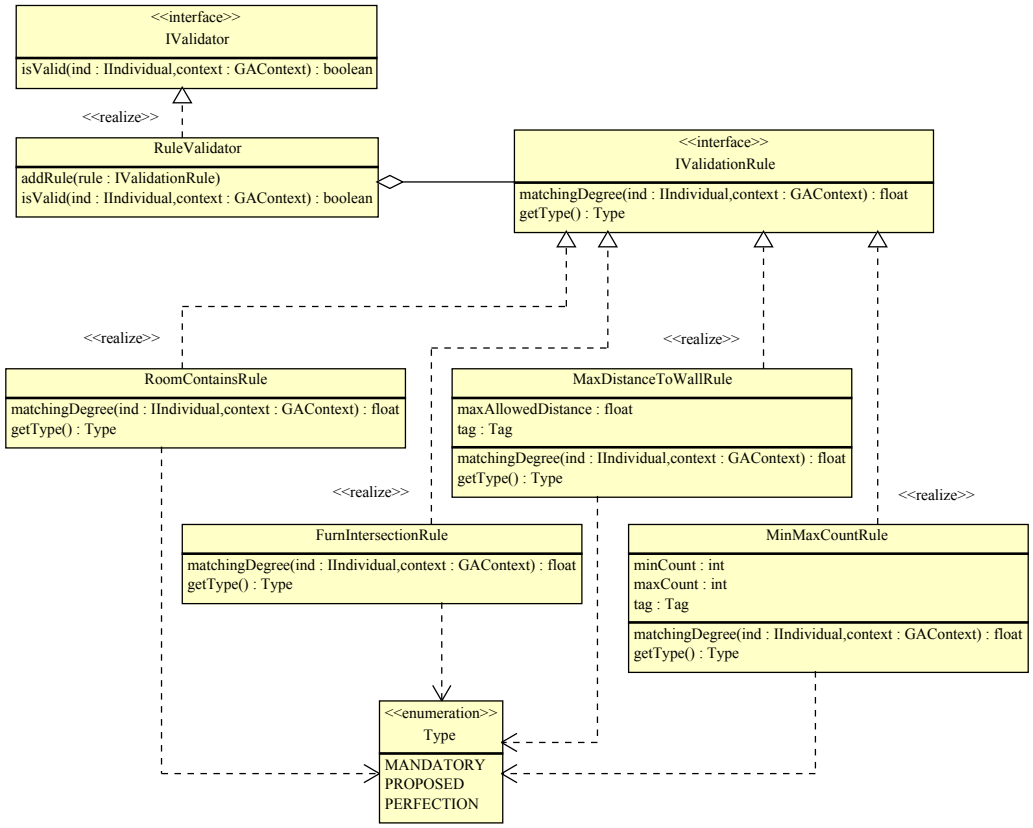


Abbildung 4.12.: Implementierte Regeln zur Validierung

- Die weiche Randbedingung, dass die Anzahl eines Einrichtungsgegenstandes ein Minimum und ein Maximum haben kann, wird durch die Klasse `MinMaxCountRule` umgesetzt. Die Regel ist vom Typ `PROPOSED`. Wenn die Anzahl eines Einrichtungsgegenstandes von der vorgeschriebenen Anzahl um 1 abweicht, so gibt die Regel noch 0,5 zurück und gilt daher als erfüllt. Bei einer höheren Abweichung fällt das Individuum durch die Validierung durch.

- Die weiche Randbedingung, dass der Abstand von einer Wand ein Maximum haben kann, wird durch die Klasse `MaxDistanceToWallRule` realisiert. Die Regel ist ebenfalls vom Typ `PROPOSED`. Die Regel gibt $min(1, \frac{d_{erlaubt}}{d_{max}})$ zurück. $d_{erlaubt}$ ist ein Parameter der Regel, d_{max} ist der maximale Abstand aller Einrichtungsgegenstände, auf die die Regel zutrifft, von einer Wand. Ist der Abstand doppelt so hoch wie erwünscht, gibt die Regel 0,5 zurück und gilt somit noch als erfüllt.

4.6. Administrationsoberfläche und Starter-Klasse

Das Paket `FurnyApp` ist die oberste Anwendungsschicht und setzt auf allen hier beschriebenen Paketen auf. Es enthält eine Klasse zum Starten der Anwendung, eine grafische Benutzeroberfläche zur Administration des Katalogs und eine Anwendung zur Analyse des evolutionären Verlaufs.

4.6.1. Benutzeroberfläche zur Administration

Nach dem Öffnen der Administrationsanwendung `FurnDBAdmin` ist das Hauptfenster zu sehen. Die Benutzeroberfläche gliedert sich in verschiedene Tabs mit den einzelnen Funktionen.

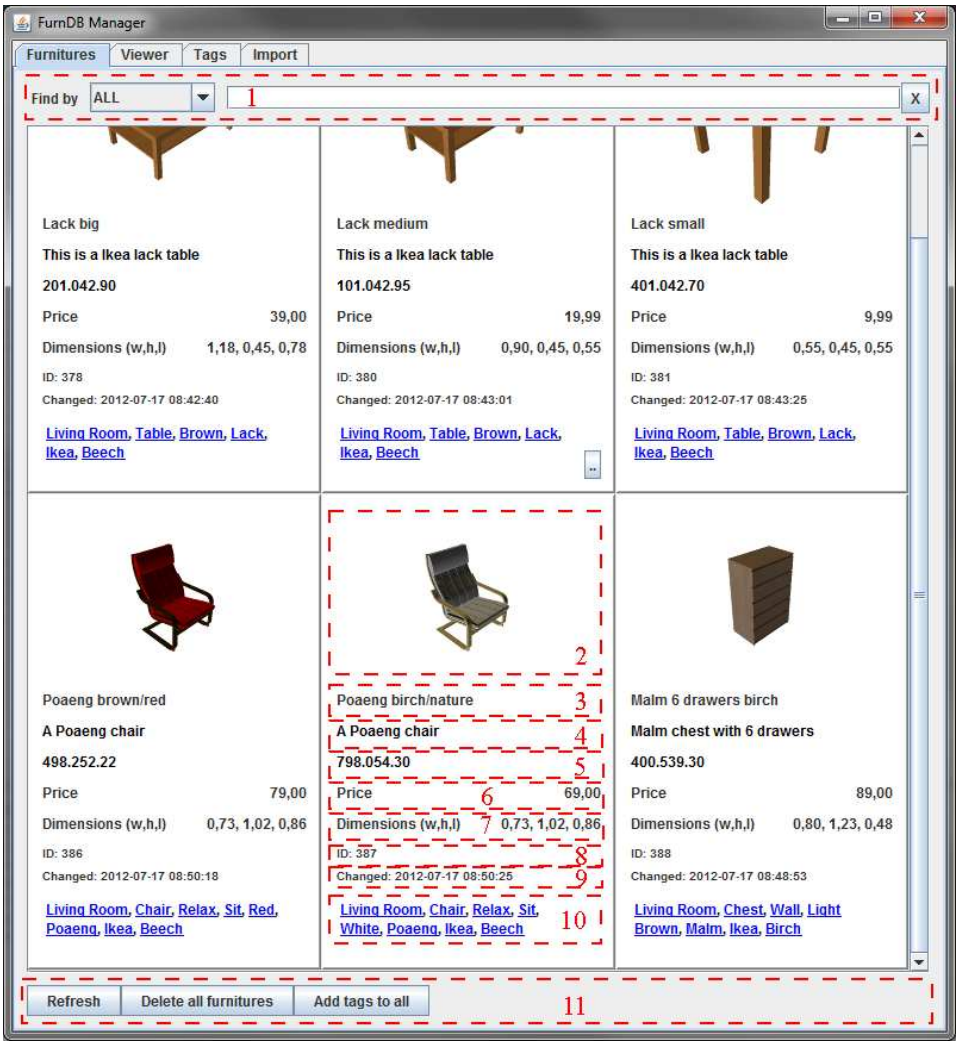


Abbildung 4.13.: Hauptfenster der Administrationsanwendung

In Abbildung 4.13 ist der Tab für die Einrichtungsgegenstände dargestellt. Oben im Bild ist die Suchleiste zu sehen (1). Der mit 2 markierte Bereich ist das Vorschaubild eines Einrichtungsgegenstandes. 3 ist der Name, 4 ist ein Beschreibungstext. Die einzelnen Daten können durch Anklicken geändert werden. Die Artikelnummer kann unter 5 angegeben werden, der Preis unter 6. Unter 7 sind die aus dem Modell berechneten Dimensionen zu sehen. Nicht veränderbar sind die ID (8) und das Datum der letzten Änderung (9). Das darunter liegende Feld (10) zeigt alle Tags, die dem Einrichtungsgegenstand zugeordnet wurden. Durch einen Klick auf einen Tag wird die Suchfunktion gestartet und die Übersicht zeigt alle Einrichtungsgegenstände mit diesem Tag an. Mit den Schaltflächen darunter (11) können die Ansicht aktualisiert, Gegenstände gelöscht und Tags allen sichtbaren Einrichtungsgegenständen zugewiesen werden. Durch Anklicken der Schaltfläche (12) öffnet sich ein Fenster zum Bearbeiten der Tags (siehe Abbildung 4.14).

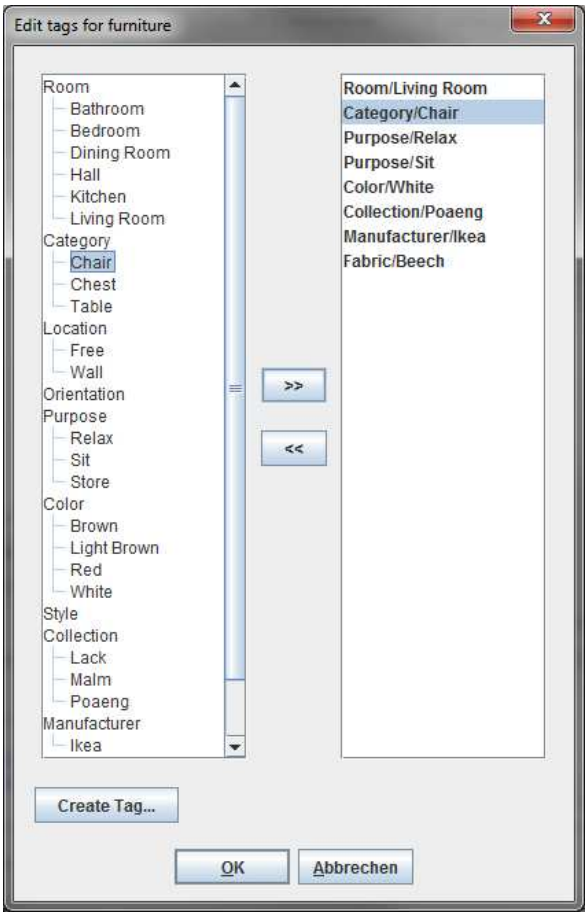


Abbildung 4.14.: Fenster zum Zuweisen und Entfernen der Tags eines Einrichtungsgegenstandes

Hier können verfügbare Tags (links im Bild) den Tags des Gegenstands (rechts im Bild) hinzugefügt werden. Das Anlegen neuer Tags ist ebenfalls möglich.

Der zweite Tab im Hauptfenster bietet eine dreidimensionale Ansicht von Einrichtungsgegenständen (siehe Abbildung 4.15). Diese Ansicht öffnet sich automatisch durch einen Doppelklick auf ein Vorschaubild im ersten Tab. Das Modell kann frei gedreht und gezoomt werden. Des Weiteren existieren Schaltflächen um den Mittelpunkt des Modells zu zentrieren und das Modell um 90° nach links oder rechts zu drehen. Der Pfeil in der Abbildung zeigt die Orientierung des Modells. Diese kann bei Regeln eine Rolle spielen, beispielsweise wenn ein Stuhl zu einem Tisch ausgerichtet werden soll.

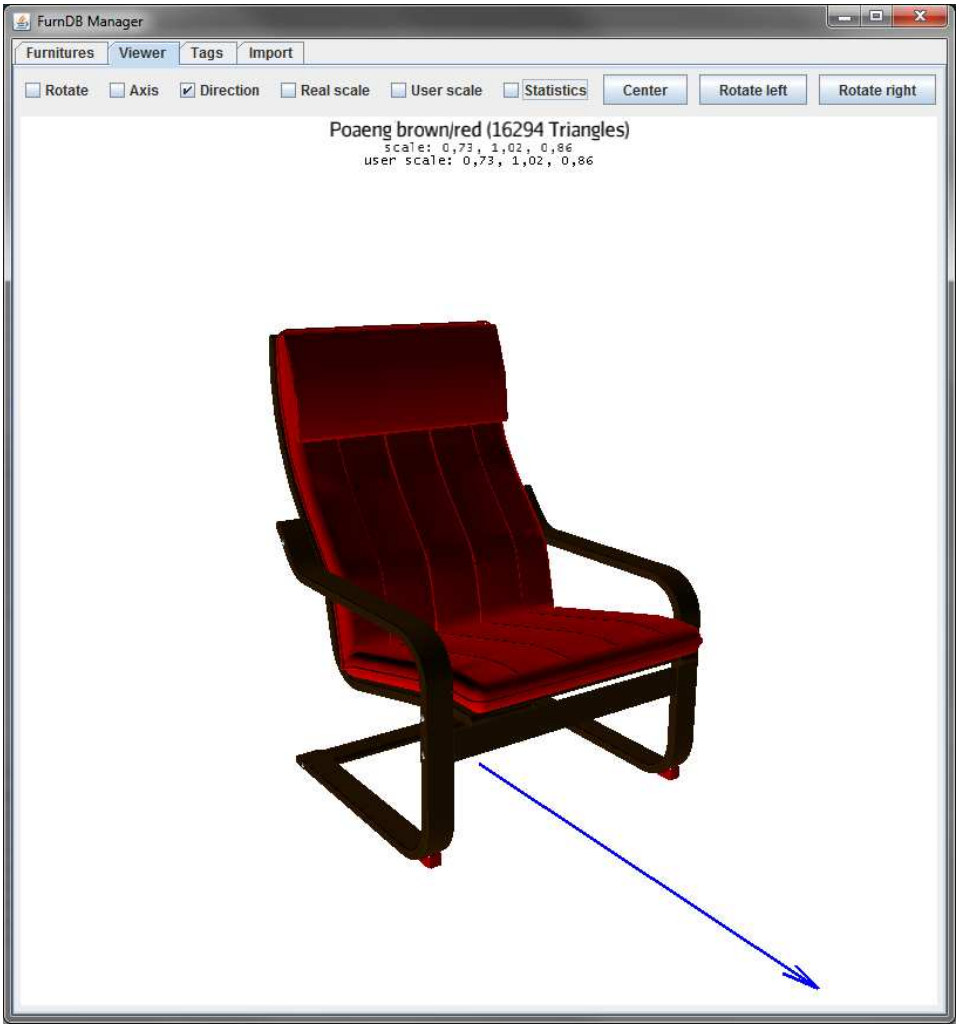


Abbildung 4.15.: 3D-Ansicht des Einrichtungsgegenstandes

Im dritten Tab des Hauptfensters können die vorhandenen Tags verwaltet werden (siehe Abbildung 4.16). Die Tags werden hier nach ihrem Typ gruppiert dargestellt. Es gibt Schaltflächen zum Erstellen neuer Tags und zum Löschen der ausgewählten Tags. Gelöschte Tags werden automatisch von allen Einrichtungsgegenständen entfernt.

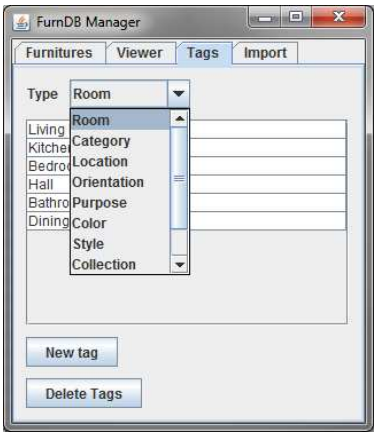


Abbildung 4.16.: Benutzeroberfläche zum Verwalten der vorhandenen Tags

Der vierte Tab des Hauptfensters in Abbildung 4.17 dient zum Import von 3D-Modellen. Hier kann das Verzeichnis ausgewählt werden, welches die zu importierenden Dateien enthält. Durch Anklicken der Import-Schaltfläche öffnet sich ein Fenster zur Auswahl der einzelnen Dateien. Der Import kann dort gestartet werden. Diese Benutzeroberfläche verwendet die in Kapitel 4.3.3 beschriebene Klasse `FurnDBImporter`.

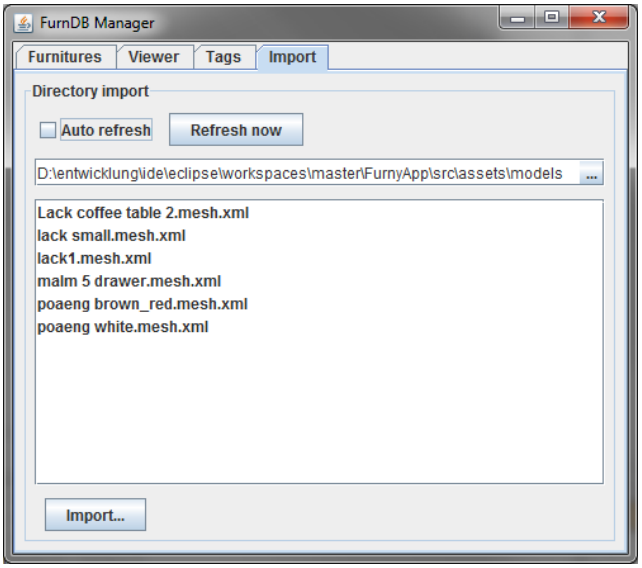


Abbildung 4.17.: Benutzeroberfläche zum Importieren von 3D-Modellen

4.6.2. Genotypen-Editor

Zu Analyse- und Testzwecken wurde eine Benutzeroberfläche zur Bearbeitung von Genotypen entwickelt. Diese ist in Abbildung 4.18 zu sehen. Das mit 1 markierte Textfeld ermöglicht die Anzeige und Bearbeitung eines Genotyps. Die textuelle Repräsentation wurde in Kapitel 3.3.1 beschrieben. Wenn der Genotyp bearbeitet wurde, kann die Vorschau des Phänotyps mit der Schaltfläche 2 aktualisiert werden. Mit der Schaltfläche 3 kann ein zufälliges Individuum erzeugt werden. Dabei kann die minimale und maximale Anzahl der Einrichtungsgegenstände konfiguriert werden. Außerdem existieren Schaltflächen (4) zum Laden und Speichern von Individuen als Dateien. Die Größe des Raums (5) kann eingestellt werden. Im Bereich 6 ist eine Vorschau des Phänotypen zu sehen.

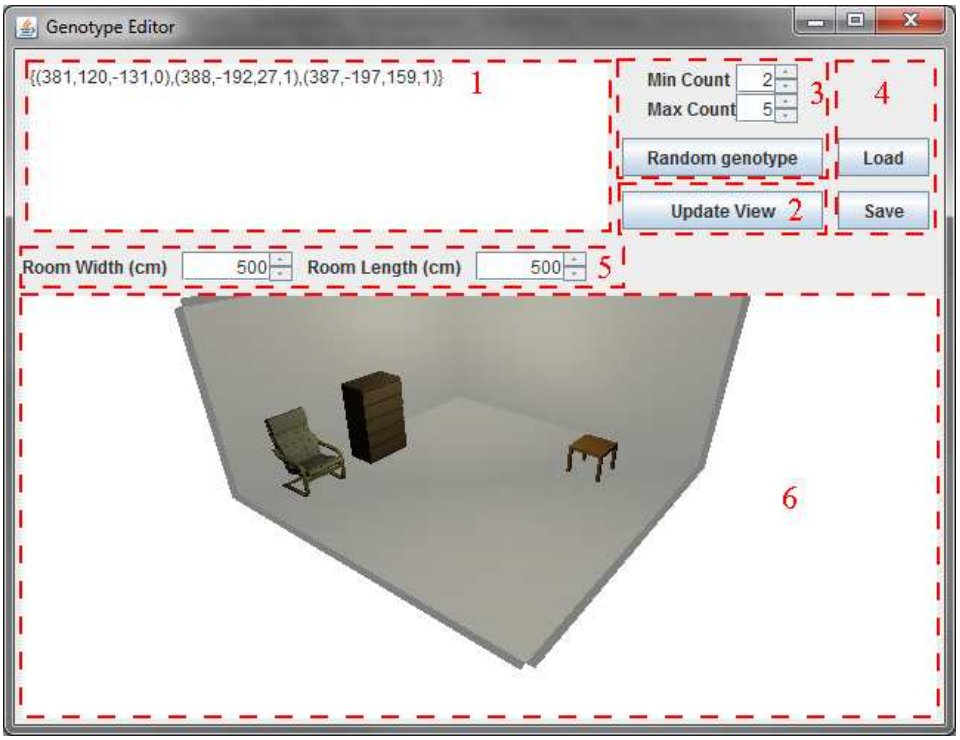


Abbildung 4.18.: Benutzeroberfläche zum Bearbeiten von Genotypen

4.6.3. Die Starter-Klasse der Anwendung

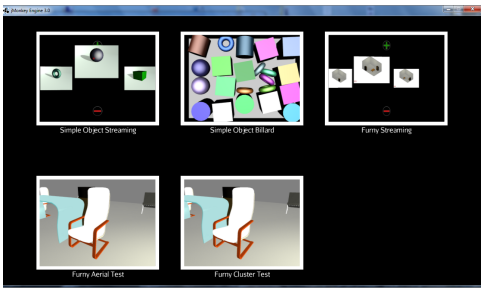
Über die Klasse `Furny` können die einzelnen Komponenten der Software gestartet werden. Die Startparameter der Klasse sind in Tabelle 4.1 aufgelistet.

Parameter	Funktion
`-h` `--help`	Hilfe anzeigen
`-v` `--verbose`	Mehr Ausgaben auf der Konsole.
`-d` `--dialog`	Den Einstellungsdialog anzeigen. Dort können Auflösung, Monitorfrequenz und Vollbildmodus vor dem Starten der Anwendung konfiguriert werden.
`-a` `--admin`	Die Administrationsanwendung (siehe Kapitel 4.6.1) starten.
`-s` `--statistics`	Die Anwendung zur Anzeige statistischer Daten starten. (nur zu Testzwecken implementiert)
`-g` `--genotype`	Die Anwendung zum Bearbeiten und Anzeigen von Genotypen (siehe Kapitel 4.6.2) starten.
`-q=QUALITAET` `--quality=QUALITAET`	Das Niveau der Anzeigequalität setzen. Der Standardwert 0 ist die geringste Qualität und eignet sich für langsame Grafikkarten. Je höher das Niveau ist, umso besser ist die Anzeigequalität.
`-vs=EINSTELLUNGSDATEI` `--viewsettings=EINSTELLUNGSDATEI`	Hier kann eine Datei mit den Einstellungen (Auflösung, Farbtiefe, uvm) der grafischen Benutzeroberfläche angegeben werden. Dieser Parameter wird gegenüber `-quality` bevorzugt behandelt.
`-f=KONFIGURATION` `--file=KONFIGURATION`	Setzt die XML-Konfigurationsdatei für den genetischen Algorithmus (siehe Kapitel 4.4.3). Wird dieser Parameter nicht gesetzt, verwendet das Programm die interne Standard-Konfiguration.
`-db=DB_DATEI` `--database=DB_DATEI`	Setzt die Datenbank-Datei für Einrichtungsgegenstände. Wird dieser Parameter nicht gesetzt, verwendet das Programm die interne Standard-Datenbank. Durch Kombination mit `-a` kann eine neue Datenbank erzeugt werden.
`-sdb=SDB_DATEI` `--statisticsdatabase=SDB_DATEI`	Setzt die Datenbank-Datei für Statistiken. (nur zu Testzwecken implementiert)

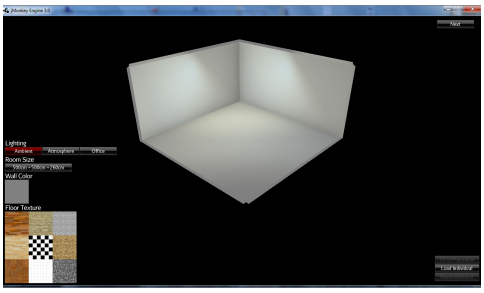
Tabelle 4.1.: Startparameter der Anwendung Furny

4.6.4. Benutzeroberfläche

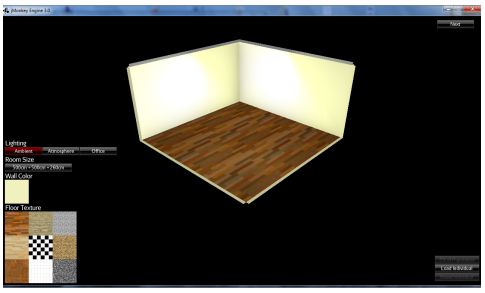
Abbildung 4.19 zeigt Screenshots der Anwendung `FurnyApp`. In Abbildung 4.19(a) ist der Startbildschirm zu sehen, der die in der XML-Konfigurationsdatei definierten evolutionären Algorithmen anzeigt.



(a) Auswahlmenü

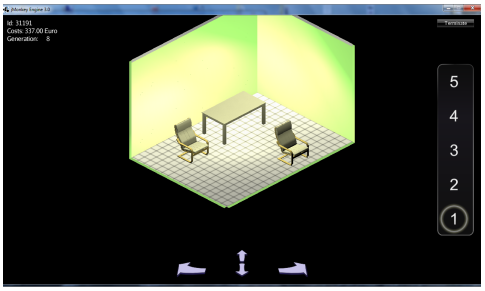


(b) Konfiguration des Raums

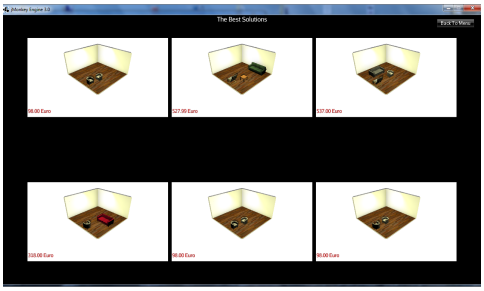


(c) Veränderte Konfiguration des Raums

(d) Evaluator in der Vogelperspektive



(e) Evaluator in der Parallelprojektion



(f) Übersicht über die besten Lösungen

Abbildung 4.19.: Benutzeroberfläche von FurnyApp

Abbildung 4.19(b) zeigt den `InitializerPreState`, der für `FurnyApp` entwickelt wurde. Dieser dient der Gestaltung des Raums, der eingerichtet werden soll. Links im Bild kann die Art der Beleuchtung ausgewählt und die Raumgröße eingegeben werden. Außerdem kann der Benutzer die Wandfarbe konfigurieren und aus einem Satz von Bodenbelägen auswählen. Diese beiden Parameter dienen nur der Visualisierung und haben keine Auswirkungen auf den Algorithmus. Rechts im Bild befindet sich die Schaltfläche zum Starten des Algo-

rithmus sowie Schaltflächen, mit denen ein Individuum für die Startpopulation geladen werden kann. Der Anteil der Population, den dieses geladene Start-Individuum einnimmt, kann ebenso konfiguriert werden. Abbildung 4.19(c) zeigt den `InitializerPreState`, nachdem die Wandfarbe konfiguriert und ein Bodenbelag ausgewählt wurde. Abbildung 4.19(d) zeigt die Benutzeroberfläche des Evaluators. Dies ist die Umsetzung des Konzepts aus Kapitel 3.4.1. Unten im Bild sind Schaltflächen zum Drehen der Perspektive. Rechts im Bild sind Schaltflächen, um die Bewertung von eins bis fünf Punkten zu vergeben. Über die Schaltfläche rechts oben im Bild kann die Evaluation vom Benutzer beendet werden. Die Kosten werden links oben dargestellt. Die Perspektive wurde für diesen Screenshot gedreht. Die Art der Perspektive kann mit der Taste „v“ verändert werden. Abbildung 4.19(e) zeigt den Evaluator mit einer anderen Wand- und Bodenkonfiguration. Die Perspektive ist hier die Parallelprojektion. Abbildung 4.19(f) zeigt den `SummaryPostState`, der für `FurnyApp` entwickelt wurde. Dieser bietet eine Übersicht über die besten sechs Individuen der Population. Diese können auch als Datei gespeichert werden. Weitere Bilder der Benutzeroberfläche sind in Anhang C zu finden.

5. Versuche und Funktionsnachweis

Dieses Kapitel enthält den Aufbau, die Dokumentation und die Bewertung der praktischen Versuche zur Clusterbildung und zum entwickelten evolutionären Algorithmus. Letzterer wird mit Hilfe einer funktionalen Fitnessbewertung und einer manuellen Evaluation getestet.

5.1. Clusterbildung

5.1.1. Ziel

In diesem Versuch sollte die Funktion des Clusterers und der Abstandsfunktion nachgewiesen und veranschaulicht werden.

5.1.2. Versuchsaufbau

Zur Durchführung der Versuche wurden sieben vorher ausgewählte Individuen in die Population eingefügt und anschließend das *k-Means++*-Verfahren mit drei Clustern angewendet. Durch das Ergebnis wurde auf die Konsole ausgegeben.

Folgende Individuen wurden für den Versuch erzeugt:

1: $(388, -67, -94, 0), (380, 114, -18, 3)$

2: $(388, -67, -94, 0), (387, 114, -18, 3)$

3: $(380, -67, -94, 0), (438, 114, -18, 3)$

4: $(380, -67, -94, 0), (380, 50, 5, 1), (380, 114, -18, 3)$

5: $(388, 114, -18, 3)$

6: $(388, 0, 0, 0)$

7: $(380, -100, 100, 0), (388, -50, -100, 0), (387, 200, -150, 0)$

5.1.3. Ergebnisse

Tabelle 5.1 zeigt die Cluster, die während der Versuche entstanden sind. Es wurde deutlich, dass die Clusterbildung mit *k-Means++* nicht deterministisch ist. So ergaben sich verschiedene Konstellationen von Clustern. Auffällig war jedoch, dass die Individuen fünf und sechs immer gemeinsam in ein Cluster eingefügt wurden. Das Gleiche ist bei eins und vier der Fall. Die Individuen zwei und sieben kamen in sechs von sieben Versuchen im selben Cluster vor.

Versuch Nr.	**Cluster 1**	**Cluster 2**	**Cluster 3**
1.1	2, 7	1, 3, 4	5, 6
1.2	1, 4, 5, 6	3	2, 7
1.3	1, 3, 4	5, 6	2, 7
1.4	2, 5, 6	1, 4, 7	3
1.5	3	1, 4, 5, 6	2, 7
1.6	1, 4, 5, 6	3	2, 7
1.7	5, 6	2, 3, 7	1, 4

Tabelle 5.1.: Versuchsprotokoll für die Clusterbildung

Abbildung 5.1 zeigt das Ergebnis des letzten Versuchs in grafischer Form. Oben im Bild ist der erste Cluster zu sehen. Er enthielt Individuen mit nur einem Schrank. Mittig im Bild ist der zweite Cluster dargestellt. Das Clusterzentrum ist mittig angeordnet. Links vom Zentrum ist ein Individuum zu sehen, das einen Tisch und ein Sitzmöbelstück mit dem Zentrum gemein hatte. Rechts vom Zentrum ist ein Individuum zu sehen, das einen Schrank und einen Stuhl mit dem Zentrum gemein hatte. Der dritte Cluster unten im Bild enthielt zwei Individuen, die als Gemeinsamkeit lediglich einen Tisch hatten, der sich jedoch bei beiden Individuen an exakt der gleichen Position befand.

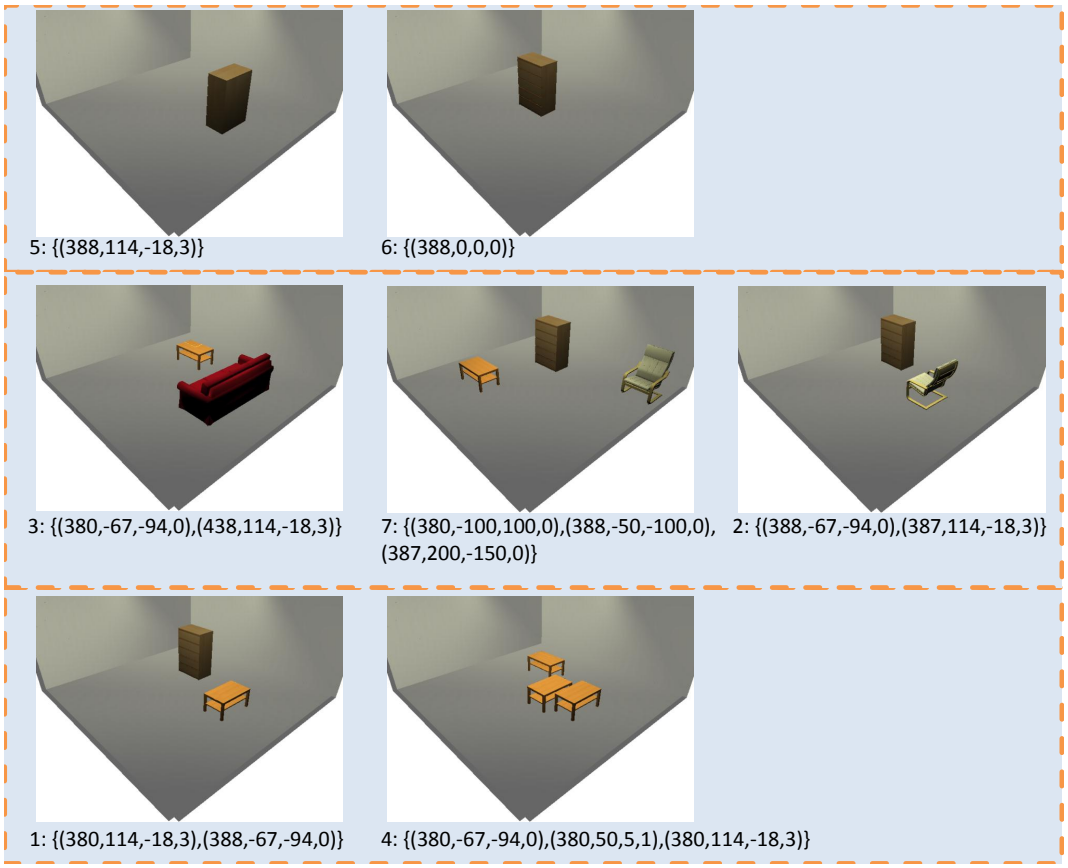


Abbildung 5.1.: Population nach der Bildung von Clustern

5.2. Automatische Evaluation

5.2.1. Ziel

Dieser Versuch diente dem Funktionsnachweis des genetischen Algorithmus. Die Fitness sollte funktional bestimmt werden. Außerdem sollte der Verlauf der Fitness aufgezeichnet und die Konvergenzeigenschaften des Algorithmus bestimmt werden.

5.2.2. Versuchsaufbau

Zur funktionalen Fitnessbestimmung wurde ein Ziel-Individuum festgelegt:
$c_{target} = (380, 114, -18, 3), (388, -67, -94, 0)$

Die Fitness jedes Individuums wurde mit Hilfe der Abstandsfunktion aus Kapitel 3.3.3 wie folgt berechnet:
$fit(c_i) = (1 - dist(c_i, c_{target})) \cdot 100$

Folgende Parameter waren bei allen Versuchen gleich:

- Bei allen Versuchen betrug die Raumgröße 5 m in der Länge sowie 5 m in der Breite.
- Die Wahrscheinlichkeit, dass bei einer Mutation Segmente entfernt oder hinzugefügt wurde, betrug jeweils 5 %.
- Die Mutationsweiten betrugen: wID=2, wTranslate=40, wRotate=2.
- Die Anzahl der verfügbaren Einrichtungsgegenstände belief sich auf 29.

Tabelle 5.2 zeigt die für die Versuche verwendeten Parametersätze des evolutionären Algorithmus. Diese sind die Art der Mutation, des Crossovers und der Selektion, die Wahrscheinlichkeiten für Mutation und Crossover, die Größe der Population, die Verwendung des Clusterings, die Anzahl der Cluster und Generationen sowie die Verwendung der Elite-Strategie. Für die Versuche wurden verschiedene Wahrscheinlichkeiten für Mutation und Crossover verwendet. Die Parametersätze 1 bis 4 verwendeten eine konventionelle Population, die Sätze 5 bis 8 verwendeten Clustering und damit die Methode der *Sparse Fitness Evaluation*. Beim den Parametersätzen 7 und 8 wurde eine kleinere Population von 20 Individuen verwendet. Alle Versuche wurden mit einem generationsbasierten *SGA* durchgeführt.

Parametersatz Nr.	Mutation	Crossover	Selektion	pMutation	pCrossover	Populationsgröße	Clustering	n Generationen	Elite-Strategie
1	Reelle Mutation	Segment-Schnitt	Turnierselektion (n=10)	5 %	55 %	100	nein	100	nein
2	Reelle Mutation	Segment-Schnitt	Turnierselektion (n=10)	55 %	5 %	100	nein	100	nein
3	Reelle Mutation	Segment-Schnitt	Turnierselektion (n=10)	15 %	15 %	100	nein	100	nein
4	Reelle Mutation	Segment-Schnitt	Turnierselektion (n=10)	5 %	55 %	100	nein	100	ja
5	Reelle Mutation	Segment-Schnitt	Turnierselektion (n=10)	15 %	15 %	100	ja (n=10)	100	nein
6	Reelle Mutation	Segment-Schnitt	Turnierselektion (n=10)	15 %	85 %	100	ja (n=50)	100	nein
7	Reelle Mutation	Segment-Schnitt	Turnierselektion (n=10)	5 %	95 %	20	ja (n=10)	100	nein
8	Reelle Mutation	Segment-Schnitt	Turnierselektion (n=10)	5 %	95 %	20	ja (n=10)	100	ja

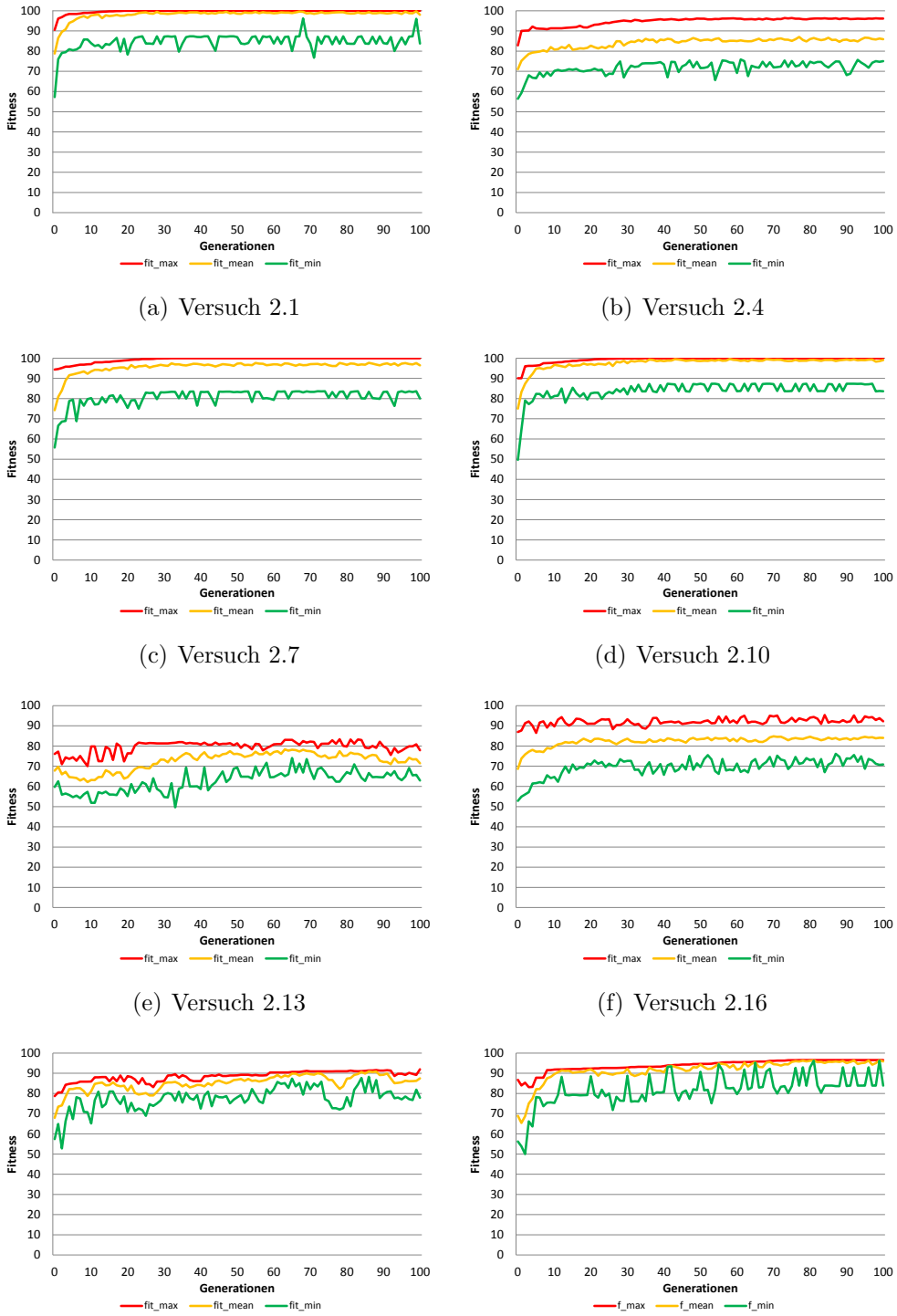
Tabelle 5.2.: Parametersätze für die Versuche mit einer automatischen Evaluation

5.2.3. Ergebnisse

Tabelle 5.3 fasst die Ergebnisse der Versuche zusammen. Das Optimum, also eine Fitness von 100 wurde nur mit den Parametersätzen 1, 3 und 4 erreicht.

Versuch Nr.	Parametersatz Nr.	Generationen zum Ziel	Fitness nach 100 Generationen		
			min	mean	max
2.1	1	44	96,00	99,74	100,00
2.2	1	54	86,99	99,19	100,00
2.3	1	44	87,23	98,95	100,00
2.4	2	-	75,04	85,89	96,15
2.5	2	-	71,98	85,32	95,54
2.6	2	-	79,40	87,27	90,36
2.7	3	55	80,08	96,47	100,00
2.8	3	-	76,98	93,56	96,55
2.9	3	60	83,61	96,83	100,00
2.10	4	53	83,65	99,05	100,00
2.11	4	-	88,72	98,92	99,56
2.12	4	77	87,37	98,56	100,00
2.13	5	-	62,90	71,47	77,90
2.14	5	-	66,71	69,53	72,69
2.15	5	-	69,28	73,01	76,09
2.16	6	-	70,85	84,04	92,24
2.17	6	-	71,78	84,14	96,85
2.18	6	-	73,63	84,29	91,91
2.19	7	-	77,91	87,51	91,97
2.20	7	-	80,68	89,31	90,44
2.21	7	-	80,54	89,38	91,15
2.22	8	-	83,88	95,86	96,51
2.23	8	-	88,42	91,14	91,30
2.24	8	-	87,14	98,63	99,93

Tabelle 5.3.: Protokoll der Versuche mit einer automatischen Evaluation



(a) Versuch 2.1

(b) Versuch 2.4

(c) Versuch 2.7

(d) Versuch 2.10

(e) Versuch 2.13

(f) Versuch 2.16

(g) Versuch 2.19

(h) Versuch 2.22

Abbildung 5.2.: Verlauf der Fitness bei Versuchen mit verschiedenen Parametersätzen

Die Diagramme in Abbildung 5.2 zeigen für jeden Parametersatz den Verlauf der Fitness über 100 Generationen. Abbildung 5.2(a) zeigt einen schnellen

Anstieg der maximalen und mittleren Fitness. Die hohe Crossover-Wahrscheinlichkeit und geringe Mutationswahrscheinlichkeit erzeugten einen starken Selektionsdruck. Abbildung 5.2(b) zeigt den Verlauf beim vierten Versuch. Die höhere Mutationswahrscheinlichkeit im zweiten Parametersatz erzeugten hier eine höhere *Diversität*, jedoch wurde das Optimum nicht erreicht, wenn fast ausschließlich Mutation eingesetzt wurde. In Versuch 1.7 (Abbildung 5.2(c)) wurde ebenfalls das Optimum erreicht. Die *Diversität* war höher als im ersten Versuch, was durch die geringeren Mittelwerte der Fitness deutlich wurde. Bei Versuch 1.10 wurden die Parameter des ersten Versuchs verwendet, zusätzlich aber noch die Elite-Strategie. Die weiteren vier Diagramme zeigen den Verlauf der Fitness bei Verwendung der *Sparse Fitness Evaluation*. Bei Versuch 1.22 wurde eine kleine Population und die Elite-Strategie verwendet, wodurch zwar nicht das Optimum erreicht werden konnte, jedoch eine bessere Konvergenz als bei den anderen Versuchen mit *Sparse Fitness Evaluation* erkennbar ist.

5.2.4. Interpretation der Ergebnisse

Während der Versuche mit einer konventionellen Population wurde die erwartete Konvergenz erkennbar. Diese fiel je nach Wahl der Parameter höher oder geringer aus. Der Nachweis der Funktion ist somit erbracht. Bei den Versuchen mit *Sparse Fitness Evaluation* konnte zunächst nicht eindeutig nachgewiesen werden, dass der Algorithmus konvergiert. Erst bei Verwendung der Elite-Strategie traten Fitnesswerte über 95 auf und die Kurve der maximalen Fitness war weniger verrauscht.

5.3. Interaktive Evaluation

5.3.1. Ziel

In diesem Versuch sollte der Funktionsnachweis des interaktiven genetischen Algorithmus erbracht werden. Die Fitness wurde dabei durch den Benutzer zugewiesen. Der Verlauf der Fitness sollte aufgezeichnet und die Konvergenzeigenschaften des Algorithmus bestimmt werden. Der Benutzer konnte bestimmen, wann das Ziel erreicht wurde.

5.3.2. Versuchsaufbau

Bei diesem Versuch sollte der Benutzer eine Anordnung von Einrichtungsgegenständen zu finden, die subjektiv jener in Abbildung 5.3 ähnelt. Es war ein Raum von 5 m Länge und 5 m Breite einzurichten. Dieser sollte einen Couchtisch sowie eine Couch und einen Sessel in der gezeigten Anordnung enthalten. Die Couch und der Sessel sollten eine ähnliche, helle Farbe haben. Die Couch sollte etwa mittig an der Wand stehen, der Tisch direkt vor der Couch.

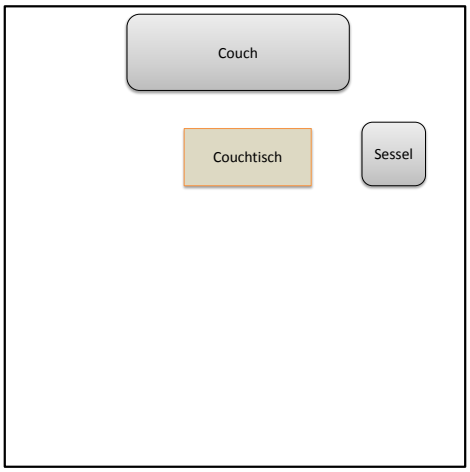


Abbildung 5.3.: Optimierungsziel des manuellen Versuchs

Punkte	Kriterien
5	Couch, Couchtisch und Sessel sind vorhanden.
	Sitzmöbel und beliebiger Tisch in der gewünschten Anordnung sind vorhanden.
4	Zwei der gesuchten Einrichtungsgegenstände sind vorhanden.
	Drei ähnliche Einrichtungsgegenstände (Sitzmöbel, beliebiger Tisch) sind vorhanden.
	Ein beliebiger Tisch steht genau vor einer Couch.
	Ein Sessel steht in der gewünschten Anordnung zu einer Couch.
3	Einer der gesuchten Einrichtungsgegenstände ist vorhanden.
	Zwei ähnliche Einrichtungsgegenstände (Sitzmöbel, beliebiger Tisch) sind vorhanden.
2	Sitzmöbel oder beliebiger Tisch vorhanden, jedoch nicht beides.
1	Weder Sitzmöbel noch Tische vorhanden.

Tabelle 5.4.: Bewertungsmaßstäbe für die manuelle Evaluation

Um einheitliche Bewertungsmaßstäbe zu schaffen, wurden die zu vergebenden Punkte für bestimmte Kritieren definiert. Tabelle 5.4 fasst diese zusammen.

Weiterhin wurde die Punktzahl durch die Kritieren in Tabelle 5.5 auf- oder abgewertet.

Kriterium	**Punktekorrektur**
Die relative Position der Einrichtungsgegenstände untereinander entspricht der Zielsetzung	+1
Die Ausrichtung der Einrichtungsgegenstände entspricht der Zielsetzung	+1
Der Einrichtungsplan bedeutet eine deutliche Verbesserung gegenüber dem bisherigen Verlauf	+1
Von einem Einrichtungsgegenstand befinden sich zu viele im Raum.	-1
Es befinden sich unerwünschte Einrichtungsgegenstände im Raum.	-1
Der Einrichtungsplan bedeutet eine deutliche Verschlechterung gegenüber dem bisherigen Verlauf	-1

Tabelle 5.5.: Kriterien zur Auf- oder Abwertung der Punktzahl

Tabelle 5.6 zeigt die für die Versuche verwendeten Parametersätze des evolutionären Algorithmus. Diese orientieren sich an den Parametern aus dem Versuch in Kapitel 5.2, außer dass aufgrund der zeitintensiven Bewertung die Populations- und Turniergröße sowie die Anzahl der Cluster verringert wurden. Folgende Parameter waren bei allen Versuchen gleich und sind nicht in der Tabelle gelistet:

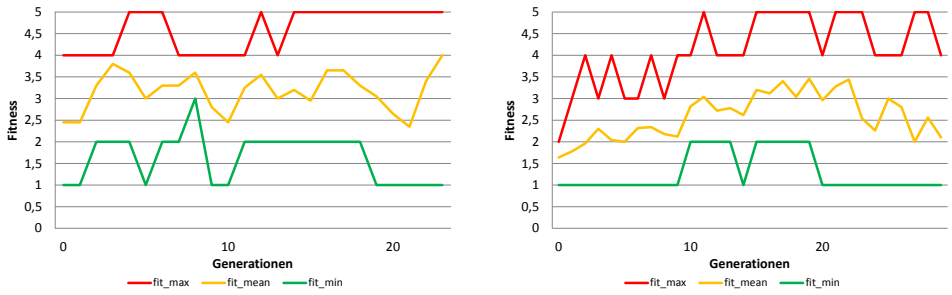
- Bei allen Versuchen betrug die Raumgröße 5 m in der Länge sowie 5 m in der Breite.
- Die Wahrscheinlichkeit, dass bei einer Mutation Segmente entfernt oder hinzugefügt wurde, betrug jeweils 5 %.
- Die Mutationsweiten betrugen: wID=2, wTranslate=40, wRotate=2.
- Die Anzahl der verfügbaren Einrichtungsgegenstände belief sich auf 29.

Parametersatz Nr.	**Mutation**	**Crossover**	**Selektion**	**pMutation**	**pCrossover**	**Populations-größe**	**Clustering**	**Elite-Strategie**
1	Reelle Mutation	Segment-Schnitt	Turnier-selektion (n=3)	5 %	55 %	20	nein	nein
2	Reelle Mutation	Segment-Schnitt	Turnier-selektion (n=3)	55 %	5 %	20	nein	nein
3	Reelle Mutation	Segment-Schnitt	Turnier-selektion (n=3)	15 %	15 %	20	nein	nein
4	Reelle Mutation	Segment-Schnitt	Turnier-selektion (n=3)	5 %	55 %	20	nein	ja
5	Reelle Mutation	Segment-Schnitt	Turnier-selektion (n=6)	15 %	15 %	40	ja (n=10)	nein
6	Reelle Mutation	Segment-Schnitt	Turnier-selektion (n=6)	15 %	85 %	40	ja (n=10)	nein
7	Reelle Mutation	Segment-Schnitt	Turnier-selektion (n=3)	5 %	95 %	20	ja (n=10)	nein
8	Reelle Mutation	Segment-Schnitt	Turnier-selektion (n=3)	5 %	95 %	20	ja (n=10)	ja

Tabelle 5.6.: Parametersätze für die Versuche mit der manuellen Evaluation

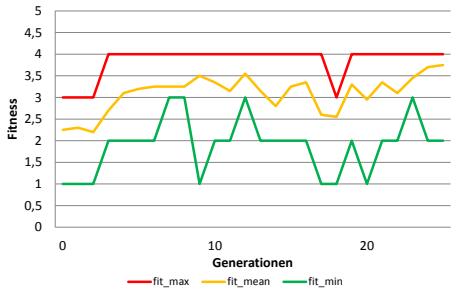
5.3.3. Ergebnisse und Beobachtungen

Abbildung 5.4 enthält die Diagramme des Fitnessverlaufs der Versuche.

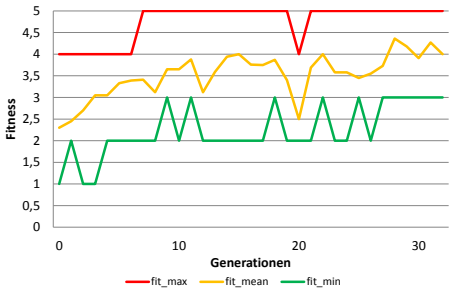


(a) Versuch 3.1

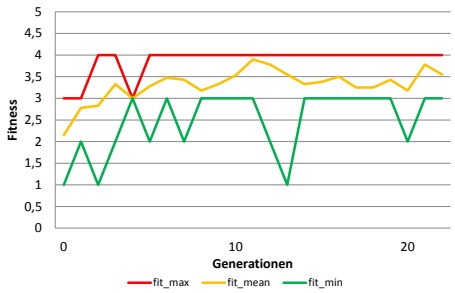
(b) Versuch 3.2



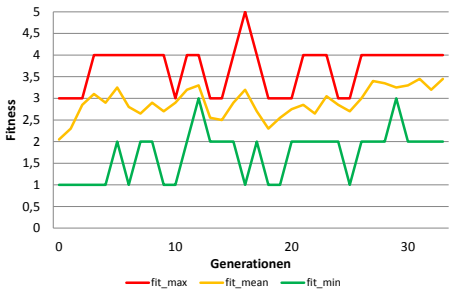
(c) Versuch 3.3



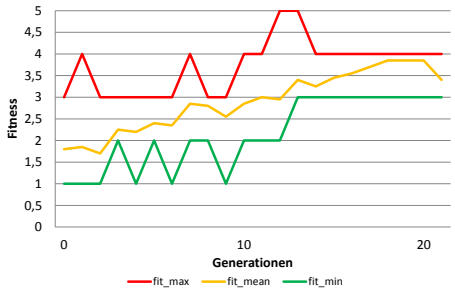
(d) Versuch 3.4



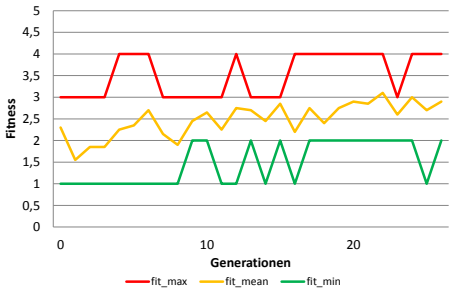
(e) Versuch 3.5



(f) Versuch 3.6



(g) Versuch 3.7



(h) Versuch 3.8

Abbildung 5.4.: Verlauf der Fitness bei den manuellen Versuchen

Folgende Beobachtungen wurden während der Versuche gemacht:

Versuch 3.1: Eine Konvergenz war spürbar und es gab gute Teillösungen. Eine Kombination dieser Teillösungen trat jedoch nicht auf. Der Algorithmus wurde nach 23 Generationen abgebrochen.

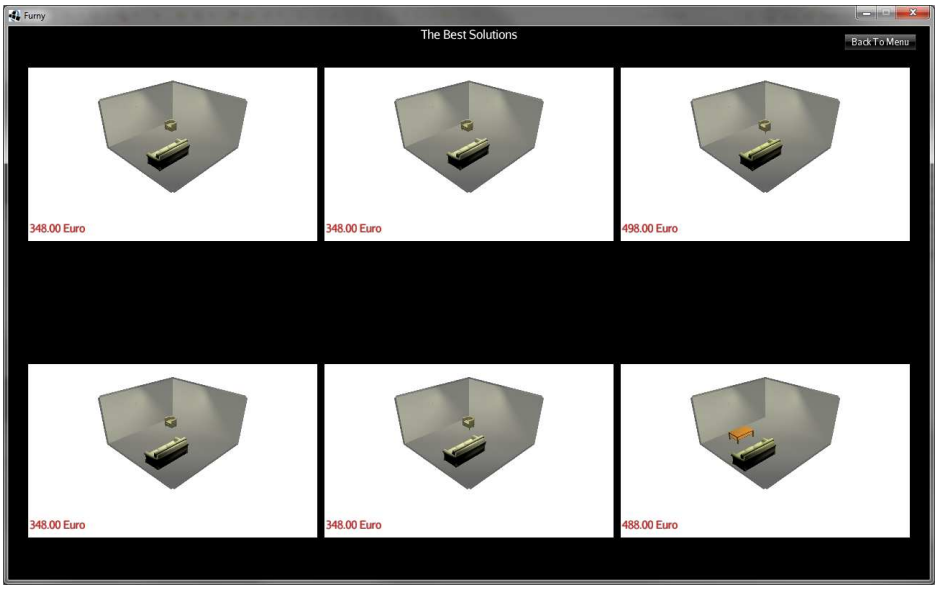


Abbildung 5.5.: Die besten Individuen bei Versuch 3.1

Versuch 3.2: Hier trat anfangs eine Verbesserung ein, jedoch fiel die mittlere Güte der Lösungen wieder merklich ab, so dass der Algorithmus nach 29 Generationen abgebrochen wurde.

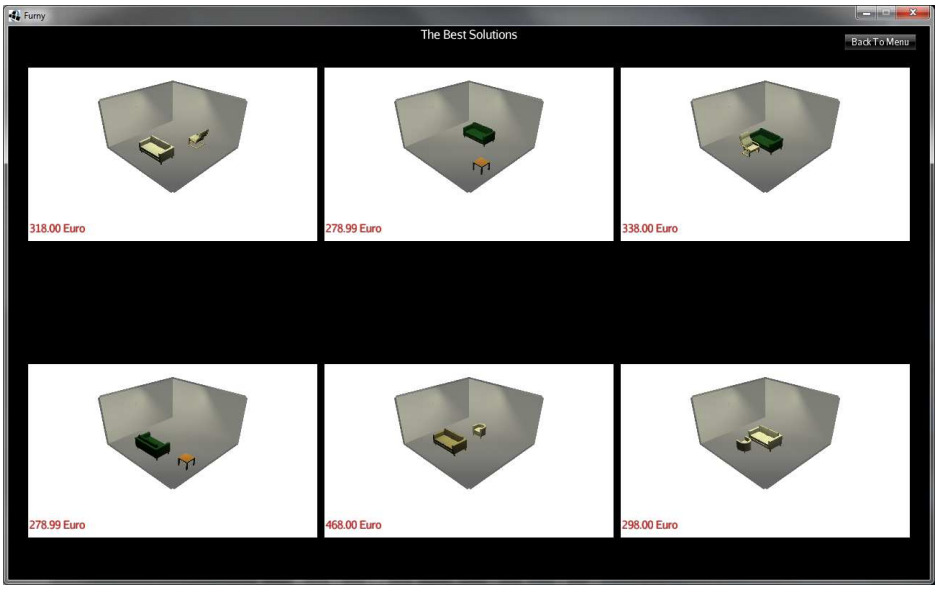


Abbildung 5.6.: Die besten Individuen bei Versuch 3.2

Versuch 3.3: In diesem Versuch trat eine vorzeitige Konvergenz auf, wodurch das Allel für den Sessel schnell aus der Population verschwand und nicht wiederkehrte. Folglich bekam kein Individuum fünf Punkte. Nach 25 Generation wurde der Versuch beendet.

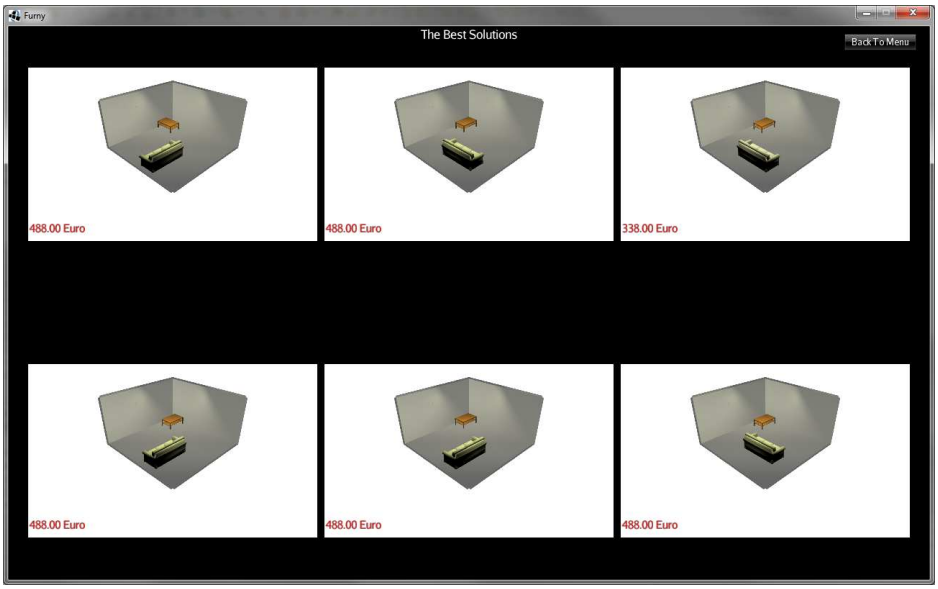


Abbildung 5.7.: Die besten Individuen bei Versuch 3.3

Versuch 3.4: Hier trat eine deutliche Konvergenz auf, so dass eine zufriedenstellende Lösung bereits in Generation 19 erreicht wurde (Siehe Abbildung 5.8). Da auch anderen Individuen zu dieser Zeit die maximale Fitness zugewiesen wurde, verschwand dieses Optimum jedoch wieder aus der Population. Nach 34 Generationen wurde der Algorithmus abgebrochen.

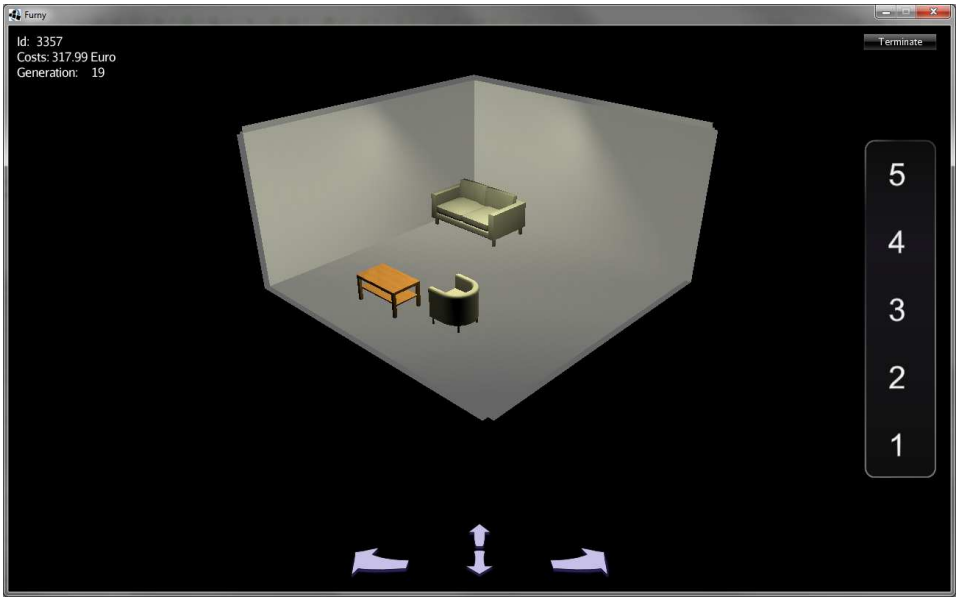


Abbildung 5.8.: Zufriedenstellende Lösung in Versuch 3.4

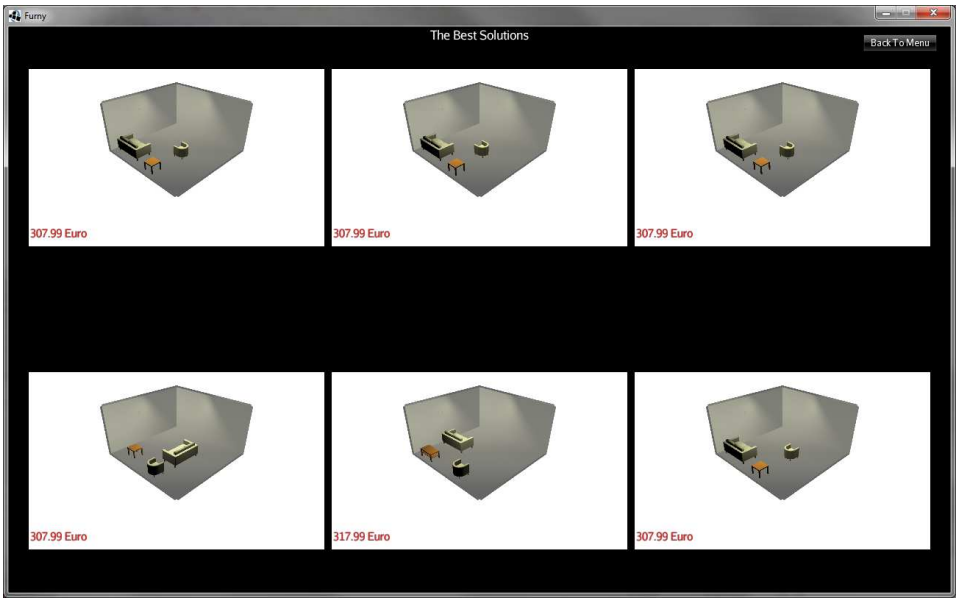


Abbildung 5.9.: Die besten Individuen bei Versuch 3.4

Versuch 3.5: Die Individuen verbesserten sich anfangs, jedoch verschwand das Allel für den Sessel aus der Population. In Generation 22 trat keine Verbesserung mehr auf, so dass der Algorithmus beendet wurde.

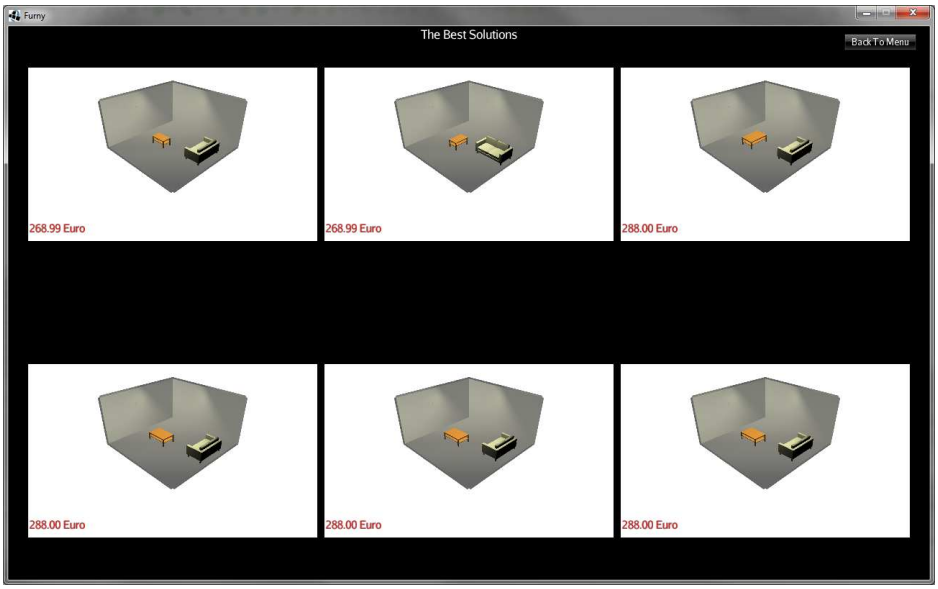


Abbildung 5.10.: Die besten Individuen bei Versuch 3.5

Versuch 3.6: In diesem Versuch traten anfangs nur wenig gute Lösungen auf. Eine Reproduktion dieser fand kaum statt, und so verschwanden sie wieder. Das Allel für den Couchtisch starb frühzeitig aus und der Versuch wurde in Generation 33 beendet.

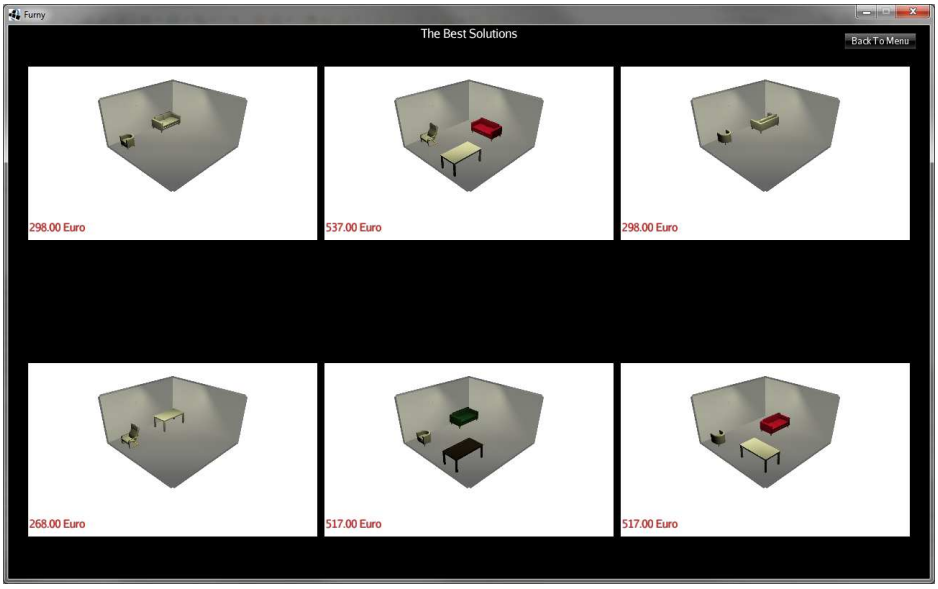


Abbildung 5.11.: Die besten Individuen bei Versuch 3.6

Versuch 3.7: Hier stieg die durchschnittliche Fitness anfangs merkbar an, jedoch konnte kein zufriedenstellendes Individuum gefunden werden. Der Algorithmus wurde nach 23 Generationen abgebrochen.

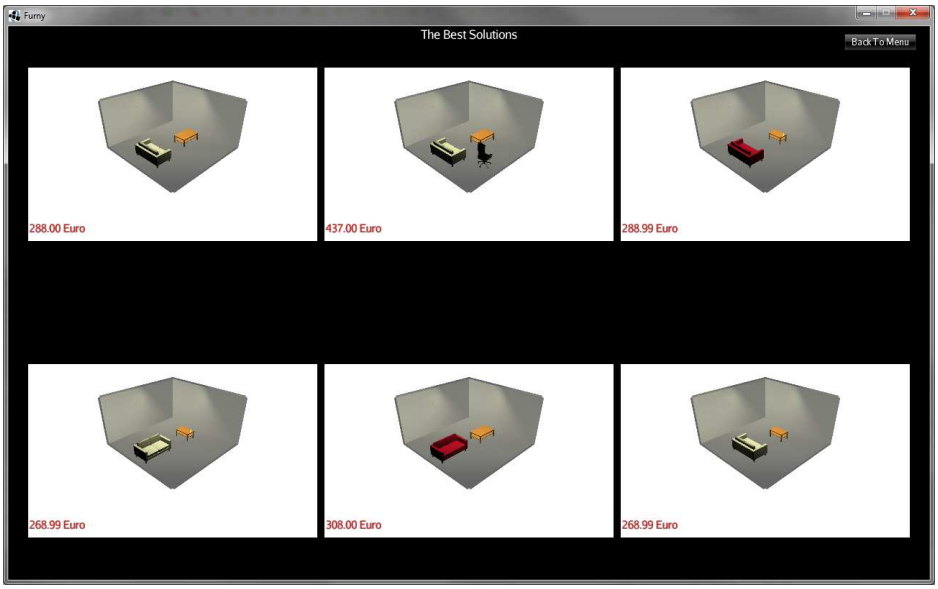


Abbildung 5.12.: Die besten Individuen bei Versuch 3.7

Versuch 3.8: Im letzten Versuch trat eine vorzeitige Konvergenz auf, wodurch das Tisch-Allel frühzeitig aus der Population verschwand. Nach 28 Generationen wurde der Algorithmus abgebrochen.

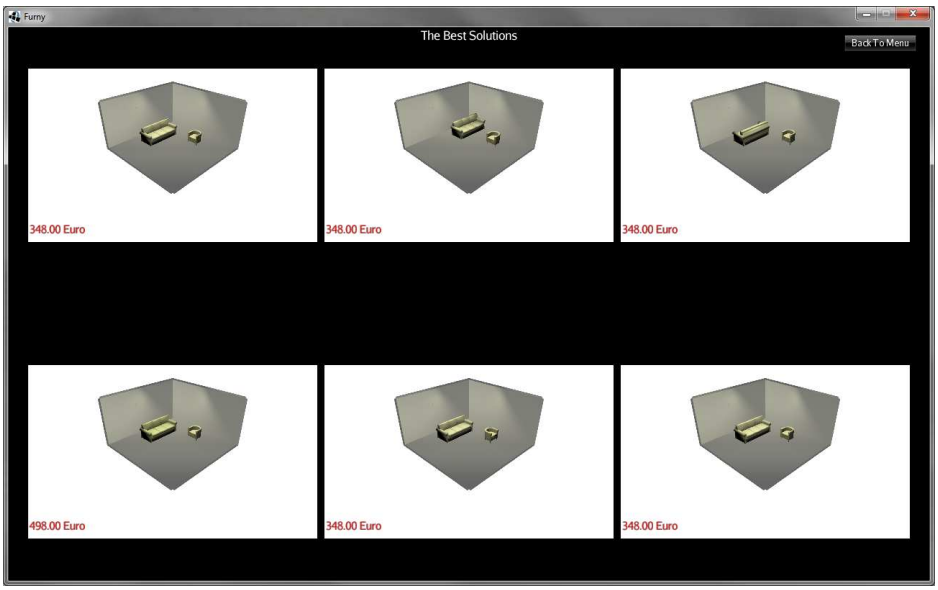


Abbildung 5.13.: Die besten Individuen bei Versuch 3.8

5.3.4. Interpretation der Ergebnisse

Die Versuche konnten einen eingeschränkten Funktionsnachweis erbringen. So wurde klar, dass eine Optimierung im Sinne einer Verbesserung eintrat. Dabei wurden jedoch nur in Versuch 3.4 zufriedenstellende Lösungen, also globale Optima gefunden. Oft trat eine vorzeitige Konvergenz auf, was sich durch das schnelle Aussterben von Genen bemerkbar machte. Dies war durch den Selektionsdruck bedingt, der von der Populationsgröße, der Turniergröße bei der Selektion und den Wahrscheinlichkeiten der genetischen Operatoren abhängt. Bei fünf Bewertungsabstufungen war es oft nicht möglich, eine gute Lösung von einer deutlich besseren Lösung abzugrenzen. Bei den Ergebnissen muss berücksichtigt werden, dass für die Versuche objektive Bewertungskritieren formuliert wurden, deren Optimalität nicht nachgewiesen ist. Der Anwendungsfall verlangt jedoch nicht nach einer zielgerichteten Evolution, sondern die Software soll dem Benutzer neue Lösungsvorschläge anbieten, die dieser dann bewertet.

6. Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse der Arbeit reflektiert und mit den Anforderungen verglichen. Außerdem wird in Aussicht gestellt, welche Möglichkeiten zur Weiterentwicklung das Konzept und die Software bieten.

6.1. Zusammenfassung

Das Ziel dieser Arbeit war die Entwicklung einer Anwendung zur Assistenz bei der Einrichtungsplanung. Der Benutzer sollte so in der Lage sein, ohne fachliche Kenntnisse eine problemorientierte Optimierung von Einrichtungsplänen durchzuführen. Die Anwendung sollte eine dreidimensionale grafische Darstellung und eine einfach zu bedienende Eingabeschnittstelle bieten. Weitere Anforderungen wurden in Kapitel 3.2 vorgestellt. Alle geforderten Funktionen wurden umgesetzt. Neben der prototypischen Anwendung zur Einrichtungsplanung entstand eine flexible Bibliothek zur Implementierung von automatischen und interaktiven evolutionären Algorithmen, die in einem frühen Stadium bereits in [Dre11] zur Entwicklung eines Prototypen genutzt wurde. Durch die strenge Einhaltung softwarearchitektonischer Richtlinien stieg der initiale Entwicklungsaufwand stark an, jedoch konnte im späteren Verlauf der Entwicklung durch die Abstraktion, Flexibilität und Konfigurierbarkeit Aufwand eingespart werden. Dies machte sich besonders bei der Durchführung der Versuche bemerkbar, als weitere Konfigurationsmöglichkeiten, die Ausgabe des Fitnessverlaufs als *CSV*-Datei, sowie eine schnelle Veränderung der Parameter notwendig wurden. Auch das Einpflegen neuer Einrichtungsgegenstände war unkompliziert.

Der Entwurf des interaktiven evolutionären Algorithmus erwies sich als problematisch. Der Genotyp in Form einer ungeordneten Menge von Chromosomensegmenten variabler Anzahl erforderte den Entwurf spezialisierter genetischer

Operatoren zur Erforschung und Feinabstimmung. Die Probleme wurden noch verstärkt, als zur Umsetzung der *Sparse Fitness Evaluation* der Entwurf einer Abstandsfunktion für Genotypen erforderlich wurde. Diese soll das komplexe subjektive Empfinden der Ähnlichkeit zwischen zwei Individuen auf eine Zahl reduzieren. Zur Lösung dieser Aufgabe wurde zunächst eine Funktion entworfen und implementiert, die sich später als ungeeignet erwies, da sie mehr eine mathematische als subjektive Ähnlichkeit berechnete. Für den zweiten Entwurf der Funktion (siehe 3.3.3) konnten einzelne Elemente übernommen werden, jedoch werden einzelne Einrichtungsgegenstände eines Individuums zunächst im zweiten Individuum gesucht und deren euklidischer Abstand ermittelt. Zur umfassenden Bearbeitung dieser Problematik wäre eine psychologische Betrachtung erforderlich, wodurch der Rahmen dieser Arbeit gesprengt worden wäre. Dennoch wird vermutlich keine perfekte Abstandsfunktion entworfen werden können.

Wie in den Anforderungen vorgegeben war, wurde eine Benutzerschnittstelle entwickelt, die eine optisch ansprechende dreidimensionale Darstellung der Einrichtungspläne bietet. Jedes Individuum kann mit wenigen Interaktionen bewertet werden und die Perspektive ist drehbar. Außerdem wurde eine Ansicht zur Konfiguration des Raums entwickelt. Es können jedoch nur rechteckige Räume konfiguriert werden. Eine weitere Ansicht bietet eine Übersicht über die Individuen mit der größten Fitness. Das Laden und Speichern von Individuen ist ebenfalls möglich.

In praktischen Versuchen konnte nachgewiesen werden, dass durch den Algorithmus eine Optimierung stattfindet und die Fitness sich verbessert. Die ersten Versuche wurden mit einer mathematischen Bewertungsfunktion durchgeführt. Da diese auf der Abstandsfunktion aus Kapitel 3.3.3 basiert, wurde deren Funktion ebenfalls nachgewiesen. Bei den Versuchen mit einer konventionellen Population, also ohne *Sparse Fitness Evaluation* wurde das Optimum mit mehreren Parametersätzen erreicht. In einem Versuch wurde eine hohe Mutationswahrscheinlichkeit und eine geringe Crossover-Wahrscheinlichkeit gewählt, wobei das Ziel nicht innerhalb von 100 Generationen erreicht werden konnte. In den Versuchen mit *Sparse Fitness Evaluation* konnte das Optimum nicht erreicht werden. Vermutlich liegt dies an der Unsicherheit der prognostizierten Fitnesswerte und dem nicht deterministischen Vergleich der Fitnessintervalle. Dennoch war in den letzten beiden automatischen Versuchen ein deutlicher Anstieg der Fitness erkennbar. Es entstand die Vermutung, dass sich diese Me-

thode zwar nicht zur Suche eines globalen Maximums eignet, aber bei interaktiven evolutionären Algorithmen eine Lösung finden kann, die der Benutzer als gut genug erachtet. Diese Vermutung konnte während der manuellen Versuche nicht bestätigt werden und das gewünschte Individuum wurde nicht gefunden. Bei einem hohen Selektionsdruck trat in einigen Fällen eine vorzeitige Konvergenz auf und für die Lösung benötigte Allele starben frühzeitig aus. Bei einem von acht manuellen Versuchen mit einer konventionellen Population konnte ein zufriedenstellendes Individuum gefunden werden. Für die zielgerichtete Evolution erwies sich der Algorithmus daher als wenig geeignet.

Die in Kapitel 3.3.4 vorgeschlagenen harten Randbedingungen waren unerlässlich für die korrekte Funktion des Algorithmus. Die weichen Randbedingungen machten jedoch einen instabilen und kontraproduktiven Eindruck, da die Freiheit des Benutzers stets eingeschränkt wird. Zur Umsetzung der Regeln wurde der Kindstod gewählt, wodurch in einigen Fällen die Rekombination guter Lösungskandidaten verhindert wurde, da die Kind-Individuen ungültig waren. Dies trat beispielsweise bei einer Überschneidung von zwei Einrichtungsgegenständen auf. Möglicherweise hätten die Methoden des genetischen Reparierens und der Straffunktionen bessere Ergebnisse geliefert. Jegliche Randbedingungen sowie Methoden zur Verringerung der Benutzerermüdung wie die *Sparse Fitness Evaluation* bringen wieder neue Parameter mit sich, so dass sich die Fragestellung entwickelt: „Wer optimiert den Optimierer?“

6.2. Ausblick

In einer weiterführenden Arbeit könnte untersucht werden, wie die Probleme zusätzlicher Parameter gelöst werden können. Möglicherweise können diese Parameter mit den Individuen evolvieren oder ein einfaches Lernverfahren zur Optimierung eingesetzt werden. Zusätzlich könnte untersucht werden, ob für den hier vorgestellten Anwendungsfall ein Genotyp fester Länge verwendet werden kann und wie die Abstandsfunktion noch verbessert werden kann. Die Parameter der genetischen Operatoren könnten variabel gestaltet werden, so dass der Verlauf der Fitness aufgezeichnet wird und bei einer deutlichen Verringerung der Fitness beispielsweise die Mutationswahrscheinlichkeit oder Mutationsweiten erhöht werden.

Die jetzige Anwendung zur evolutionären Einrichtungsplanung könnte noch erweitert werden. Wenn die Anwendung Benutzern zur Verfügung gestellt werden soll, sind ausschließlich rechteckige Räume nicht ausreichend. Es sollte ein Raumerzeugungssystem implementiert werden, mit dem Wände, Fenster, Türen und Heizkörper platziert werden können. In diesem Zusammenhang wären Regeln erforderlich, um zu verhindern, dass Einrichtungsgegenstände vor Türen oder Fenstern platziert werden. Außerdem sollte die Begehbarkeit sichergestellt werden, indem immer ein Durchgangsbereich freigehalten wird. Weiterhin könnten alle in Kapitel 3.3.4 vorgeschlagenen Randbedingungen als Regeln implementiert werden. Dem Benutzer könnte auch die Möglichkeit gegeben werden, in die Evolution einzugreifen und Möbel zu verschieben.

Für den Einsatz der Software im produktiven Umfeld sollte die Benutzeroberfläche noch ansprechender gestaltet werden. Die Software könnte den Benutzer auch direkt zur Bestellung führen. Weiterhin könnten die Ergebnisse jeder Optimierung in einer Datenbank gespeichert und zur Initialisierung zukünftiger Populationen verwendet werden.

Der Import von Einrichtungsgegenständen könnte auch noch verbessert werden. Der Umweg über *SketchUp* und *OgreXML*-Dateien sowie die manuelle Annotation sollten in Zukunft nicht mehr nötig sein. Stattdessen sollte untersucht werden, mit welchen gängigen Formaten Einrichtungsplaner arbeiten und wie die Metadaten einfacher integriert werden können.

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Glossar

Allel

Ein einzelnes Allel stellt einen Wert dar, den ein *Gen* annehmen kann (vgl. [Wei02] S. 27).

Annotation

Im Kontext dieser Arbeit: Eine Erweiterung des Informationsgehalts eines Allels durch das Hinzufügen von Metadaten.

Chromosom

Ein Chromosom ist die Erbinformation eines Individuums, siehe *Genotyp*.

Clusterbildung

Verfahren zur Gruppierung von Individuen mit ähnlichen Eigenschaften.

Diversität

Die Diversität einer *Population* bezeichnet die Vielfalt an Allelen für jedes einzelne Gen innerhalb einer Population oder eine hohe Verteilung der Individuen im Suchraum. (vgl. [Wei02] S. 83).

EA

siehe *evolutionärer Algorithmus*.

Elite-Strategie

Bei der Elite-Strategie wird unabhängig von der Selektion immer das beste Individuum in die nächste Generation übernommen. (vgl. [BHS07] S. 75).

Evaluator

Eine Softwarekomponente zur Selektion oder Bewertung von Individuen..

Evolutionsstrategien

Evolutionsstrategien (kurz: ES) sind *evolutionäre Algorithmen*, bei denen die Genotypen immer reellwertig sind. ES verwenden die Komma- sowie Plus-Selektion und einen Mutations-Operator. (vgl. [Wei02] S. 131).

evolutionärer Algorithmus

Evolutionäre Algorithmen (kurz: EA) übertragen die natürliche Evolution auf technische Systeme. Sie stellen eine Möglichkeit zur Optimierung von Problemlösungen dar (vgl. [BHS07]).

evolutionäres Programmieren

Evolutionäres Programmieren (kurz: EP) ist ein *evolutionärer Algorithmus*, bei dem die Genetik nicht berücksichtigt wird. Bei dem Nachkommen spielt nur die phänotypische Ähnlichkeit eine Rolle. EP verwendet keinen Rekombinations-Operator. (vgl. [Wei02] S. 138).

Fitness

Die Fitness beschreibt die Güte des Phänotyps und beeinflusst die Reproduktionswahrscheinlichkeit des Individuums. Sie kann als explizite Fitness (numerischer Wert) oder implizite Fitness (z.B. Rang in der Population) auftreten.

Gen

Die kleinste genetische Informationseinheit (vgl. [Wei02] S. 27).

Gendrift

Gendrift bezeichnet den Effekt, wenn *Allele* einzelner *Gene* in kleinen Populationen durch *Selektion* oder Zufallseffekte aussterben. (vgl. [Wei02] S. 31).

genetischer Algorithmus

Eine Form von evolutionären Algorithmen, bei der der Genotyp eines Individuums als ein Vektor von Parametern oder als einziger Bitstring dargestellt wird. (vgl. [BHS07]) Der Terminus „genetischer Algorithmus“ wird oft mit GA abgekürzt.

genetischer Operator

Ein genetischer Operator ist eine Abbildung einer Menge von Individuen auf eine zweite Menge von Individuen. Es wird hier zwischen Variation (Mutation) und Rekombination (Crossover) unterschieden.

genetisches Programmieren

Genetisches Programmieren (kurz: GP) ist ein *evolutionärer Algorithmus* zur Entwicklung von Funktionen oder Programmbäumen.

Genotyp

Darstellung eines Individuums im Parameterraum, in der Literatur auch manchmal als Chromosom bezeichnet. Entspricht den Genen eines Lebewesens und kodiert die vererbbaren Informationen.

IEA

siehe *interaktiver evolutionärer Algorithmus*.

Individuum

Ein Individuum ist eine mögliche Problemlösung in der Optimierung mit evolutionären Algorithmen.

interaktive Fitness

Fitnesswert, der durch den Benutzer zugewiesen wird.

interaktiver evolutionärer Algorithmus

Evolutionärer Algorithmus mit *interaktiver Fitness*.

jMonkeyEngine

Eine in Java entwickelte 3D-Engine auf Basis von Szenengraphen, die OpenGL zum Rendern verwendet..

Konvergenz

Konvergenz bezeichnet bei einem *EA* die Annäherung an das gesuchte Optimum (vgl. [GKK04] S. 56).

Phänotyp

Ausprägung der Eigenschaften, die durch den Genotyp des Individuums kodiert sind, ähnlich dem Körper eines Lebewesens.

Population

Eine Population ist die Menge aller Individuen zu einem bestimmten Zeitpunkt.

Selektion

Die Selektion bestimmt, welche Individuen reproduziert oder ersetzt werden. Es handelt sich um eine Abbildung einer Individuenmenge I^m auf eine andere Individuenmenge I^n. I^n ist eine Teilmenge von I^m.

Literaturverzeichnis

[BHS97] Bäck, T. ; Hammel, U. ; Schwefel, H. P.: Evolutionary computation: comments on the history and current state. In: *IEEE Transactions on Evolutionary Computation* 1 (1997), Nr. 1, 3–17. http://dx.doi.org/10.1109/4235.585888. – DOI 10.1109/4235.585888. – ISSN 1089–778X

[BHS07] Boersch, Ingo ; Heinsohn, Jochen ; Socher, Rolf: *Wissensverarbeitung: Eine Einführung in die Künstliche Intelligenz für Informatiker und Ingenieure*. Spektrum Akademischer Verlag, 2007. – ISBN 3827418445

[BRT05] Brintrup, A.M ; Ramsden, J. ; Tiwari, A.: Integrated qualitativeness in design by multi-objective optimization and interactive evolutionary computation. In: *Evolutionary Computation, 2005. The 2005 IEEE Congress on* Bd. 3, 2005. – ISBN 0–7803–9363–5, 2154–2160

[Dar09] Darwin, Charles: *On the Origin of Species by Means of Natural Selection*. BiblioLife, LLC, 2009. – ISBN 1116871319

[Dre11] Dreyer, Stephan: *Benutzerschnittstellen für die Interaktive Evolution*, Fachhochschule Brandenburg, Studienarbeit, 2011. http://ots.fh-brandenburg.de/downloads/abschlussarbeiten/sa_StephanDreyer.pdf

[FF94] Fonseca, Carlos M. ; Fleming, Peter J.: *An Overview of Evolutionary Algorithms in Multiobjective Optimization*. Sheffield : University of Sheffield, Dept. of Automatic Control and Systems Engineering, 1994

[Fis08] Fischer, Robert: *Entwicklung einer Applikation zur Layoutoptimierung von Webseiten mit evolutionären Algorithmen*

und interaktiver Fitness, Fachhochschule Brandenburg, Bachelorarbeit, 2008. `http://ots.fh-brandenburg.de/downloads/abschlussarbeiten/ba_robert_fischer.pdf`

[GG07] Gong, Dunwei ; Guo, G.: Interactive Genetic Algorithms with Interval Fitness of Evolutionary Individuals. Version: 2007. `http://www.iocen.com/upload/201107/20110703145639525.pdf`. In: Watam Press (Hrsg.): *Complex Systems and Applications*. 2007, 446–450

[GKK04] Gerdes, Ingrid ; Klawonn, Frank ; Kruse, Rudolf: *Evolutionäre Algorithmen: Genetische Algorithmen - Strategien und Optimierungsverfahren - Beispielanwendungen*. Vieweg, 2004. – ISBN 3528055707

[GYM08] Gong, Dunwei ; Yuan, Jie ; Ma, Xiaoping: Interactive Genetic Algorithms with Large Population Size. In: *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, 2008. – ISBN 978–1–4244–1822–0, 1678–1685

[GYY09] Gong, Dunwei ; Yao, Xin ; Yuan, Jie: Interactive Genetic Algorithms with Individual Fitness not Assigned by Human. In: *Journal of Universal Computer Science* 15 (2009), Nr. 13, S. 2446–2462

[Hol75] Holland, John H.: *Adaptation in Natural and Artificial Systems - An Introductory: Analysis with Applications to Biology, Control, and Artificial : Intelligence*. Ann Arbor : The University of Michigan Press, 1975

[HT00] Hayashida, Norimasa ; Takagi, Hideyuki: Visualized IEC - Interactive Evolutionary Computation with Multidimensional Data Visualization. In: *IEEE International Conference on Industrial Electronics, Control and Instrumentation (IECON2000)* (27. Oktober 2000), S. 2738–2743

[IT09] Inoue, Makoto ; Takagi, Hideyuki: EMO-based Architectural Room Floor Planning. In: *IEEE International Conference on Systems, Man and Cybernetics (SMC2009)* (Oktober 2009), S. 524–529

[JME12] JME: *jMonkeyEngine Homepage*. `http://jmonkeyengine.com`. Version: 22.07.2012, Abruf: 22.07.2012. – Online; Zuletzt geprüft am 22.07.2012

[KMN+02] Kanungo, T. ; Mount, D.M ; Netanyahu, N.S ; Piatko, C.D ; Silverman, R. ; Wu, A.Y: An efficient k-means clustering algorithm: analysis and implementation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002), Nr. 7, 881–892. `http://dx.doi.org/10.1109/TPAMI.2002.1017616`. – DOI 10.1109/TPAMI.2002.1017616. – ISSN 0162–8828

[KZTA05] Kamalian, Raffi ; Zhang, Ying ; Takagi, Hideyuki ; Agogino, Alice M.: Reduced Human Fatigue Interactive Evolutionary Computation for Micromachine Design. In: *4th International Conference on Machine Learning and Cybernetics (ICMLC 2005)* (August 2005), S. 5666–5671

[Láz08] Lázaro, Tomás: *JMonkey Engine User's Guide*. http://jmonkeyengine.com/documentation/JMonkey.pdf, 2008. – Online; Zuletzt geprüft am 22.07.2012

[LC99] Lee, Joo-Young ; Cho, Sung-Bae: Sparse Fitness Evaluation for Reducing User Burden. In: *1999 IEEE International Fuzzy Systems Conference Proceedings* (August 1999), S. 998–1003

[LKC01] Lee, Jong-Ha ; Kim, Hee-Su ; Cho, Sung-Bae: Accelerating Evolution by Direct Manipulation for Interface Fashion Design. In: *4th International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2001)* (2001), S. 343–347

[LSA+06] Llorà, Xavier ; Sastry, Kumara ; Alías, Francesc ; Goldberg, David E. ; Welge, Micheal: *Analyzing Active Interactive Genetic Algorithms using Visual Analytics*. Februar 2006 (IlliGAL Report)

[MN97] Matsumoto, Makoto ; Nishimura, Takuji: *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*. Bonn : MPI, 1997

[Nis97] Nissen, Volker: *Einführung in evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution*. Braunschweig : Vieweg, 1997. – ISBN 9783528054991

[PC97] Poli, Riccardo ; Cagnoni, Stefano: Genetic Programming with User-Driven Selection: Experiments on the Evolution of Algorithms for Image Enhancement. In: *Genetic Programming*, Morgan Kaufmann, 1997, S. 269–277

[Sat02] Sato, Yuji: Voice Conversion Using Interactive Evolution Of Prosodic Control. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. San Francisco and CA and USA : Morgan Kaufmann Publishers Inc, 2002 (GECCO '02). – ISBN 1–55860–878–8, 1204–1211

[Tak00] Takagi, Hideyuki: Active User Intervention in an EC Search. In: *International Conference on Information Sciences (JCIS2000)* (März 2000), S. 995–998

[Tak01] Takagi, Hideyuki: Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation. In: *Proceedings of the IEEE* 89 (September 2001), Nr. 9, S. 1275–1296

[TO07] Takagi, H. ; Ohsaki, M.: Interactive Evolutionary Computation-Based Hearing Aid Fitting. In: *IEEE Transactions on Evolutionary Computation* 11 (2007), Nr. 3, 414–427. `http://dx.doi.org/10.1109/TEVC.2006.883465`. – DOI 10.1109/TEVC.2006.883465. – ISSN 1089–778X

[Wei02] Weicker, Karsten: *Evolutionäre Algorithmen*. 1. Stuttgart u.a : Teubner, 2002 (Leitfäden der Informatik). – ISBN 3–519–00362–7

[WHDY05] Wiese, K.C ; Hendriks, A. ; Deschenes, A. ; Youssef, B.B: P-RnaPredict-a parallel evolutionary algorithm for RNA folding: effects of pseudorandom number quality. In: *IEEE Transactions on NanoBioscience* 4 (2005), Nr. 3, 219–227. `http://dx.doi.org/10.1109/TNB.2005.853656`. – DOI 10.1109/TNB.2005.853656. – ISSN 1536–1241

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit zum Thema

Interaktive Evolution zur Assistenz bei der Einrichtungsplanung

selbstständig und ohne fremde Hilfe angefertigt habe. Es wurden keine außer den im Literaturverzeichnis angegebenen Quellen und Hilfsmitteln verwendet. Die Arbeit hat in dieser oder ähnlicher Form noch keinem anderen Prüfungsausschuss vorgelegen.

Brandenburg, den 9. August 2012

_____

Stephan Dreyer

A. Umfrage zur Planung der Inneneinrichtung

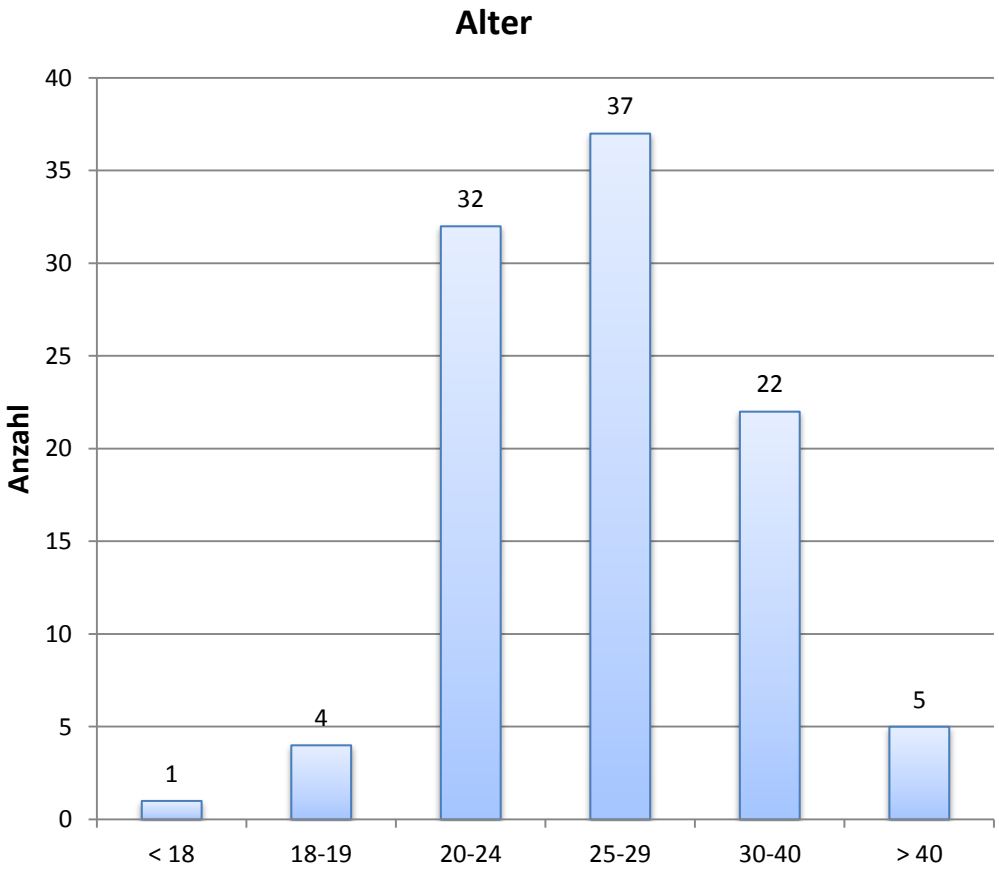
A.1. Befragung

Die Umfrage wurde im Internet über die Plattform www.voycer.de vom 23.07.2011 bis zum 01.03.2012 durchgeführt. Die Anzahl der Teilnehmer belief sich auf 101. Der Zugang zur Umfrage war nicht beschränkt. Sie wurde über ein Forum zur Inneneinrichtung (www.einrichtungsforum.de) sowie Bekannte ersten und zweiten Grades propagiert. Außerdem schlägt Voycer selbst den Besuchern der Webseite die Teilnahme an Umfragen vor.

Das Ziel der Befragung war die Ermittlung von Bedarf und Anforderungen an eine Software zur Inneneinrichtungsplanung. Des Weiteren sollten Gegenargumente zur Nutzung einer solchen Software ermittelt werden.

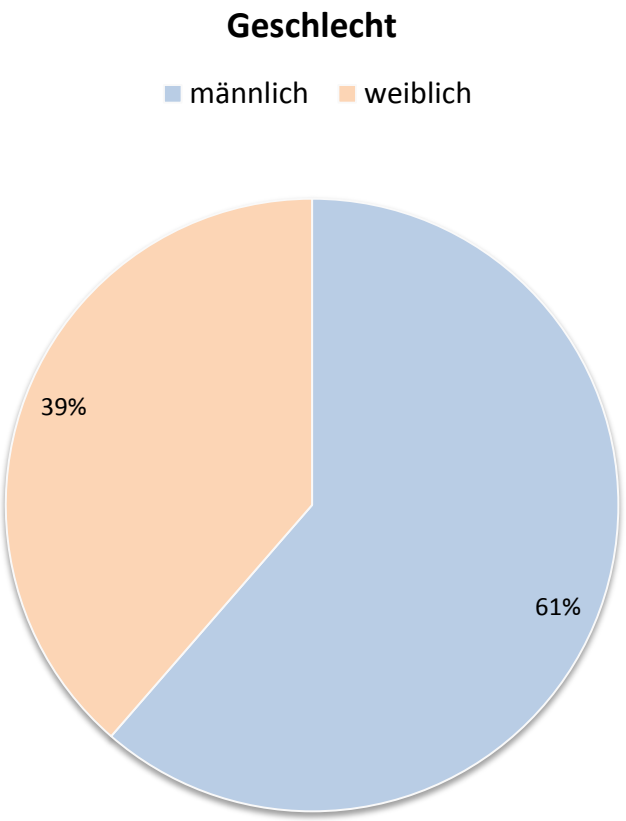
A.2. Ergebnisse der Befragung

Frage 01: „Wie alt sind Sie?“ (n=101)

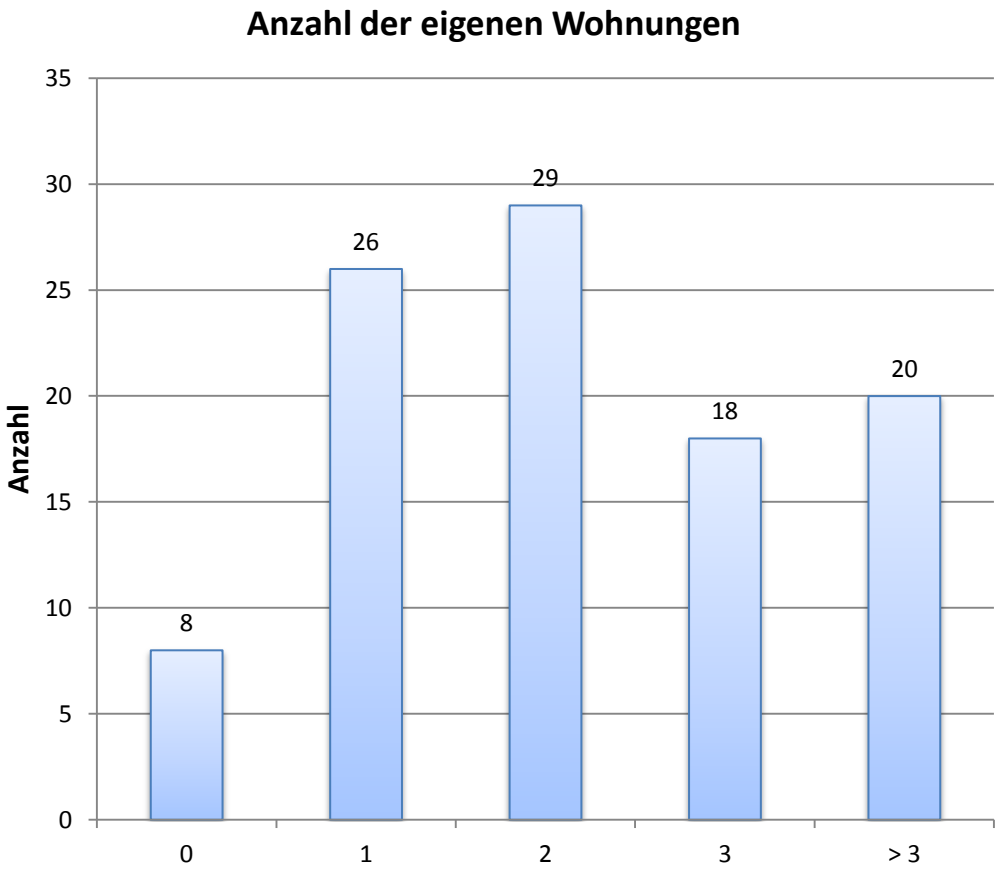


Das Durchschnittsalter der Befragten liegt bei 28,8 Jahren, wenn von einem Intervall von 0 bis 99 Jahren ausgegangen wird.

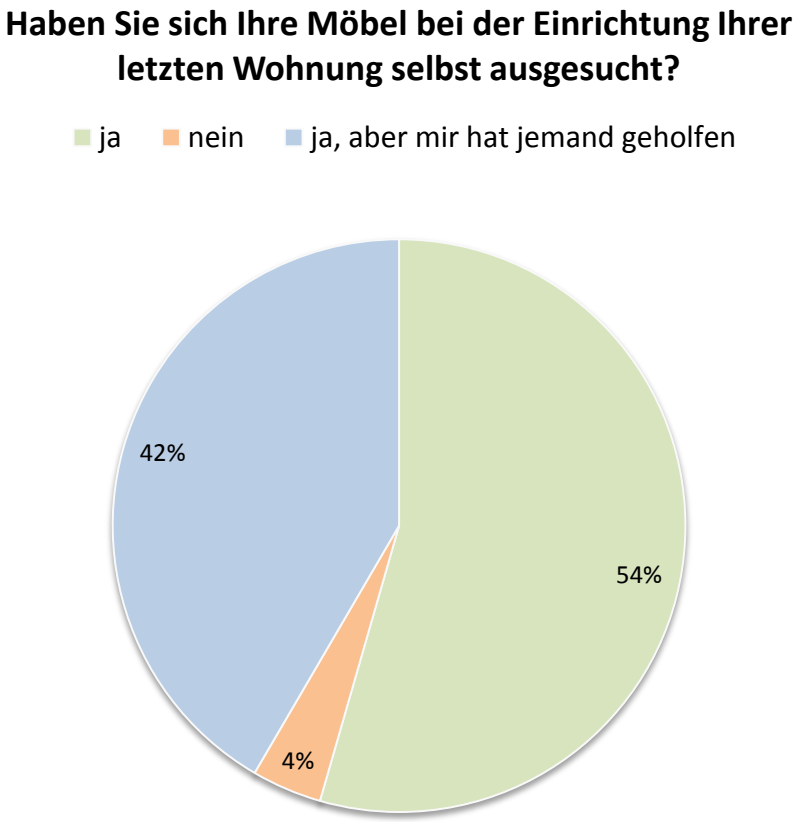
Frage 02: „Sind Sie männlich oder weiblich?“ (n=101)



Frage 03: „Wie viele eigene Wohnungen hatten Sie bisher?“ (n=101)



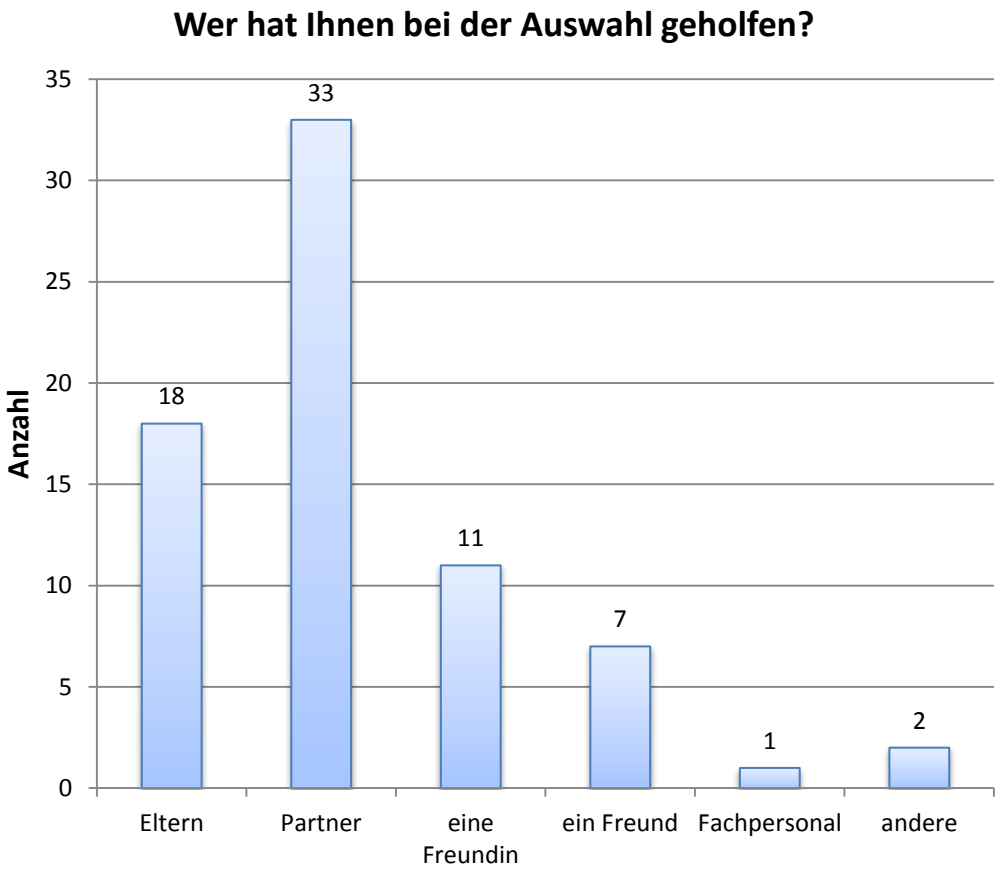
Frage 04: „Haben Sie Ihre Möbel bei der Einrichtung Ihrer letzten Wohnung selbst ausgesucht?“ (n=101)



96,0 % der Befragten haben sich ihre Möbel selbst ausgesucht.

Frage 05: „Wer hat Ihnen bei der Auswahl geholfen?“ (n=42)

Beantwortung war nur möglich, wenn bei Frage 04 die Antwort „ja, aber mir hat jemand geholfen“ ausgewählt wurde. Mehrfache Antworten waren möglich.



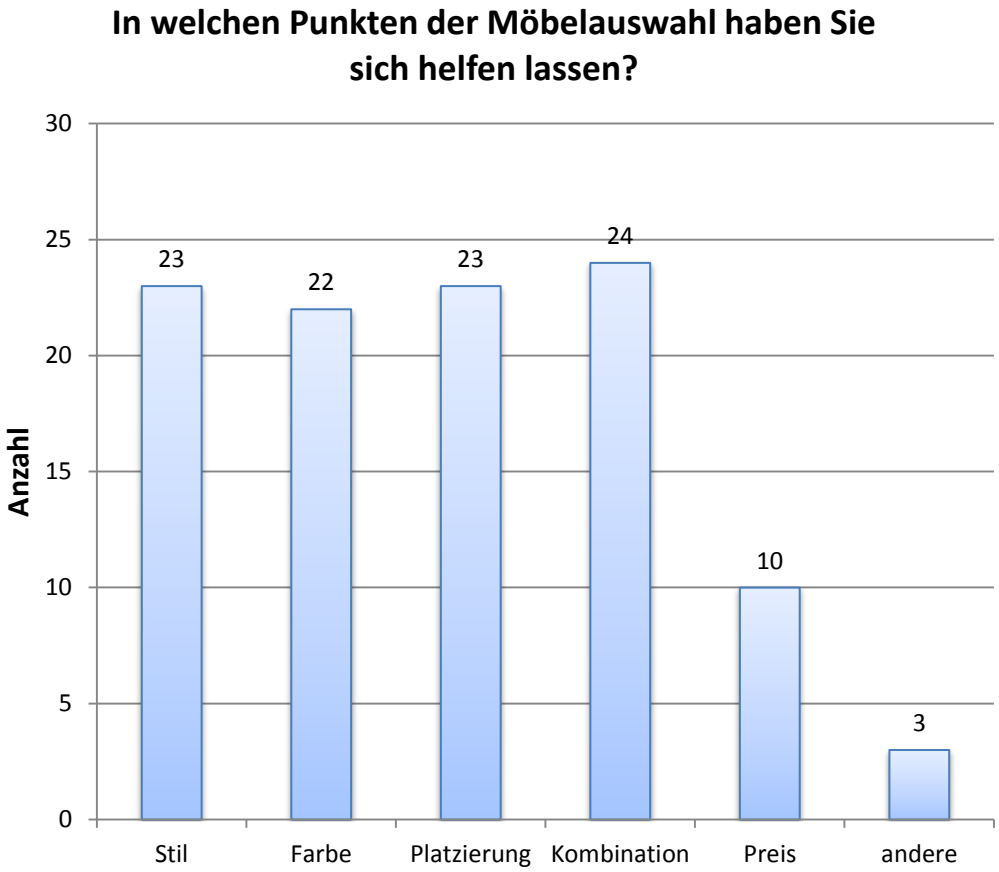
Andere Person:

„Schwester“

32,7 % aller Befragten haben bei der Möbelauswahl die Hilfe des Partners an Anspruch genommen.

Frage 06: „In welchen Punkten der Möbelauswahl haben Sie sich helfen lassen?“ (n=42)

Beantwortung war nur möglich, wenn bei Frage 04 die Antwort „ja, aber mir hat jemand geholfen“ ausgewählt wurde. Mehrfache Antworten waren möglich.

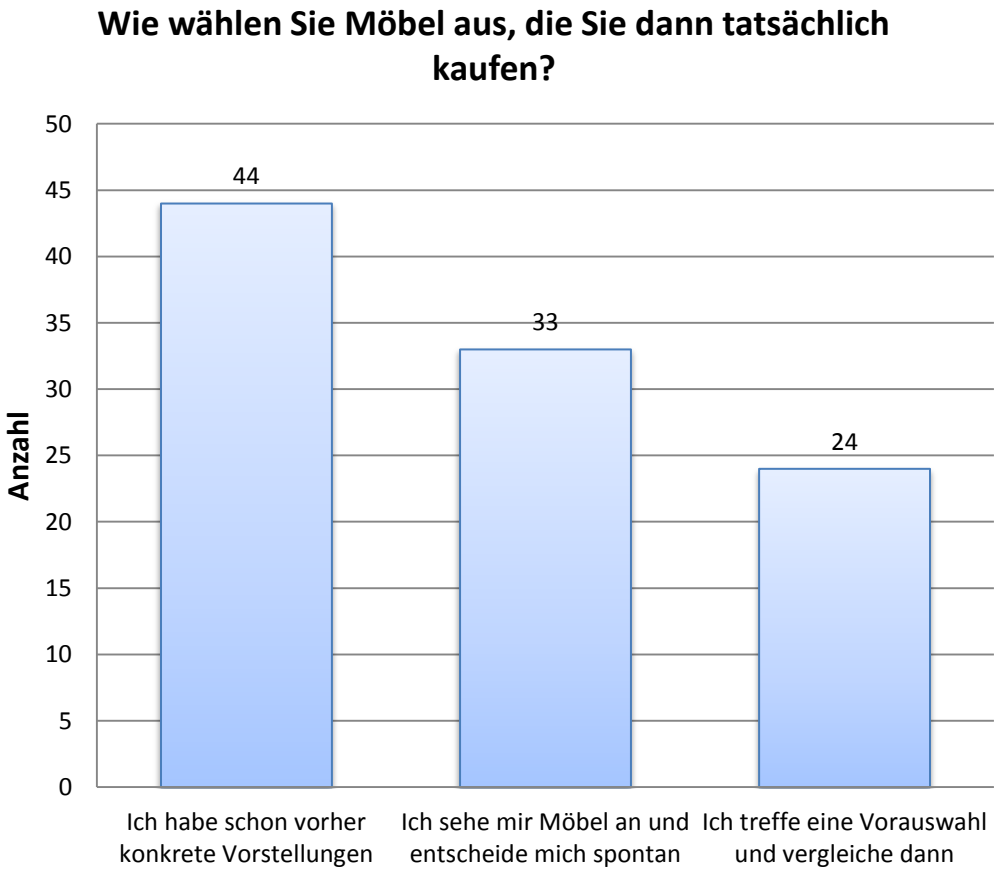


Andere Antworten:

„Sie haben geholfen, Möbel in meiner Vorstellung zu finden“

Die Befragten haben bei Stil, Farbe, Platzierung und Kombination ähnlich oft Hilfe in Anspruch genommen. Nur 9,9 % der Befragten haben sich beim Preis der Möbel beraten lassen.

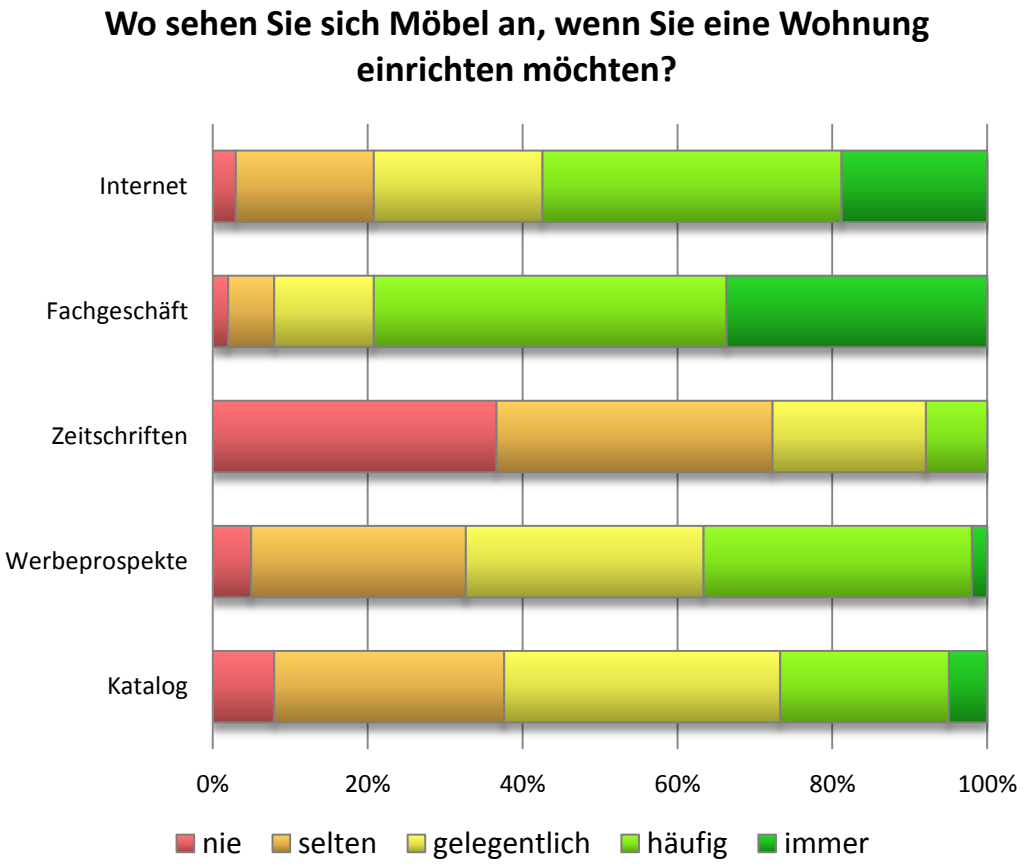
Frage 07: „Wie wählen Sie Möbel aus, die Sie dann tatsächlich kaufen?“ (n=101)



32,7 % der Befragten entscheiden sich spontan für bestimmte Möbel.

Frage 08: “Wo sehen Sie sich Möbel an, wenn Sie eine Wohnung einrichten möchten?“ (n=101)

Zu allen Informationsquellen musste eine Antwort abgegeben werden.



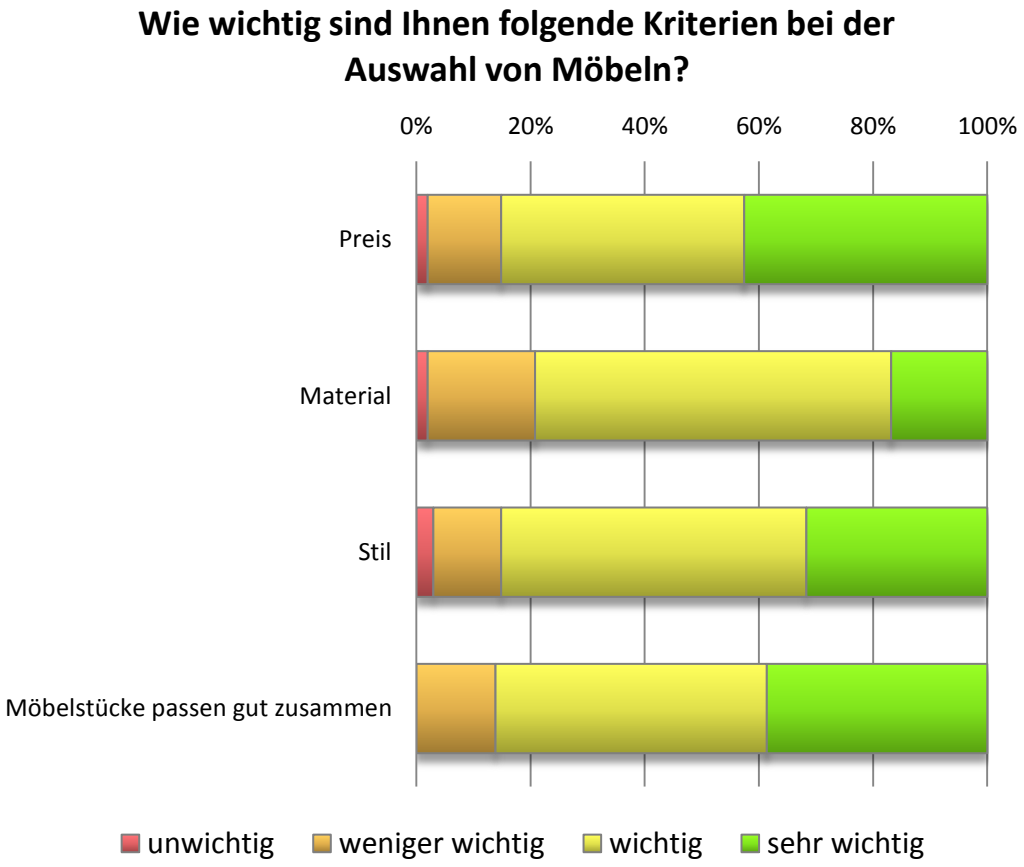
Am häufigsten sehen sich die Befragten Möbel im Fachgeschäft an (siehe Tabelle A.1). Zeitschriften werden nur selten genutzt.

Informationsquelle	**Mittelwert der Häufigkeit**
Fachgeschäft	häufig
Internet	gelegentlich - häufig
Werbeprospekte	gelegentlich
Katalog	gelegentlich
Zeitschrifen	selten

Tabelle A.1.: Informationsquellen für Möbel absteigend geordnet nach Häufigkeit

Frage 09: „Wie wichtig sind Ihnen folgende Kriterien bei der Auswahl von Möbeln?“ (n=101)

Zu allen Kriterien musste eine Antwort abgegeben werden.

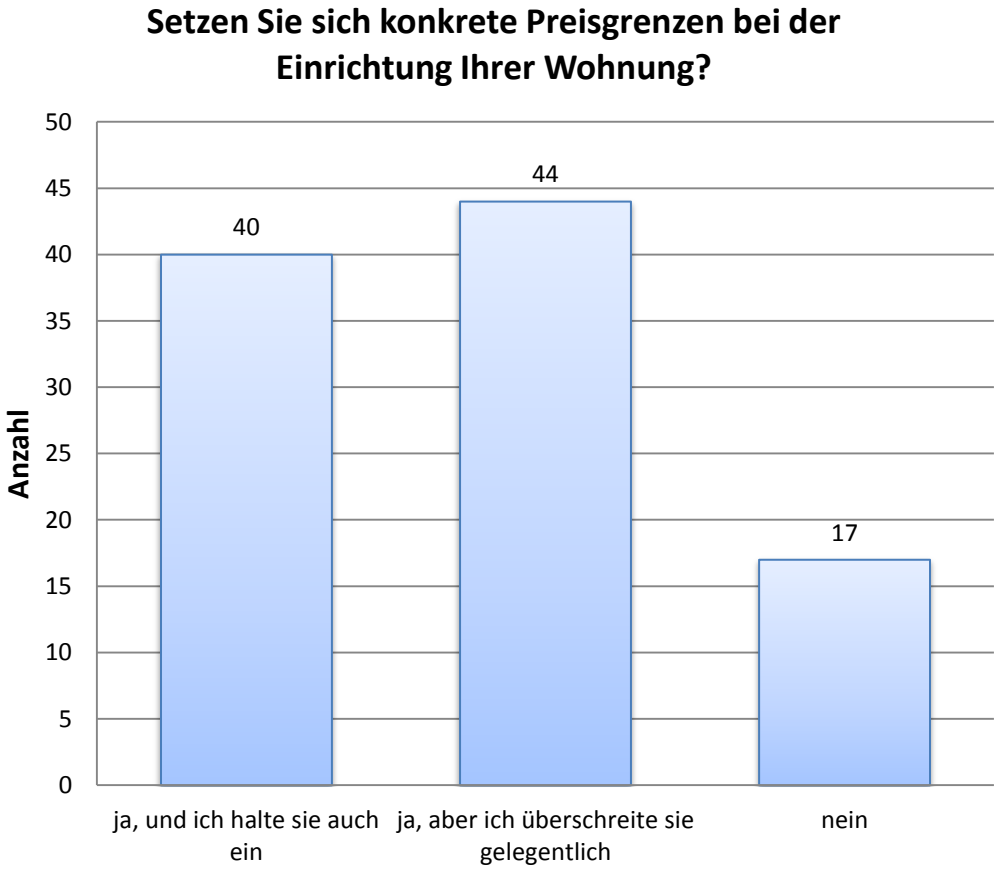


Am wichtigsten ist den Befragten der Preis von Möbelstücken, dicht gefolgt von Zusammenstellung, Stil und Material (siehe Tabelle A.1).

Kriterium	**Mittelwert der Relevanz**
Preis	wichtig - sehr wichtig
Möbelstücke passen gut zusammen	wichtig
Stil	wichtig
Material	wichtig

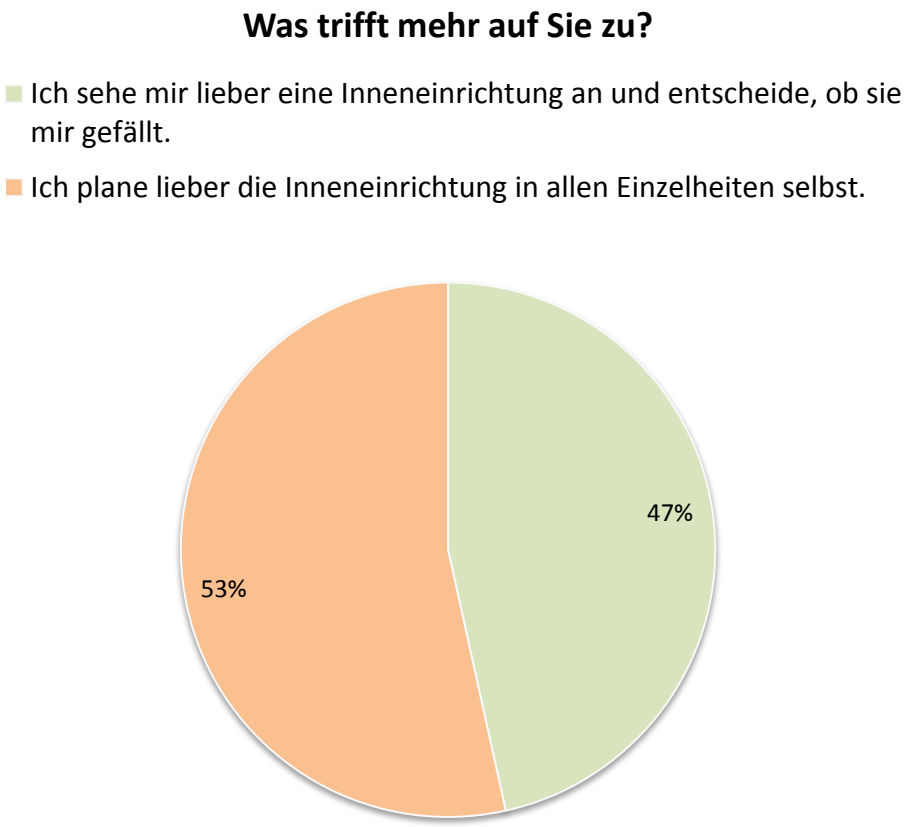
Tabelle A.2.: Kriterien bei der Auswahl von Möbeln absteigend geordnet nach Relevanz

Frage 10: „Setzen Sie sich konkrete Preisgrenzen bei der Einrichtung“ (n=101) Ihrer Wohnung?

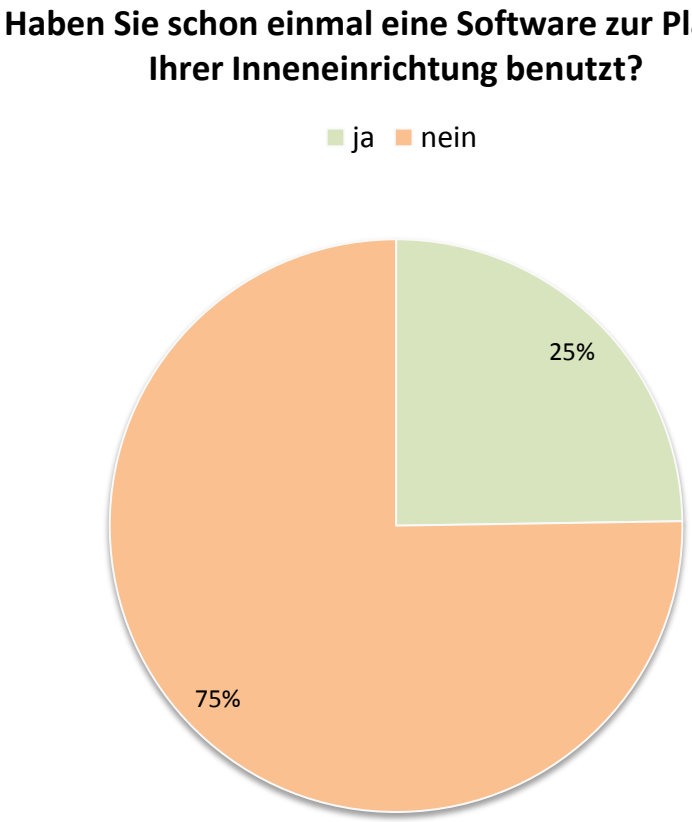


83,2 % der Befragten setzen sich konkrete Preisgrenzen.

Frage 11: „Was trifft mehr auf Sie zu?“ (n=101)



Frage 12: „Haben Sie schon einmal eine Software zur Planung Ihrer Inneneinrichtung benutzt?“ (n=101)



Frage 13: „In wie weit hat Ihnen diese Software geholfen?“ (n=25)

Beantwortung war nur möglich, wenn bei Frage 12 die Antwort „ja“ ausgewählt wurde. Zu allen Zielen musste eine Antwort abgegeben werden.

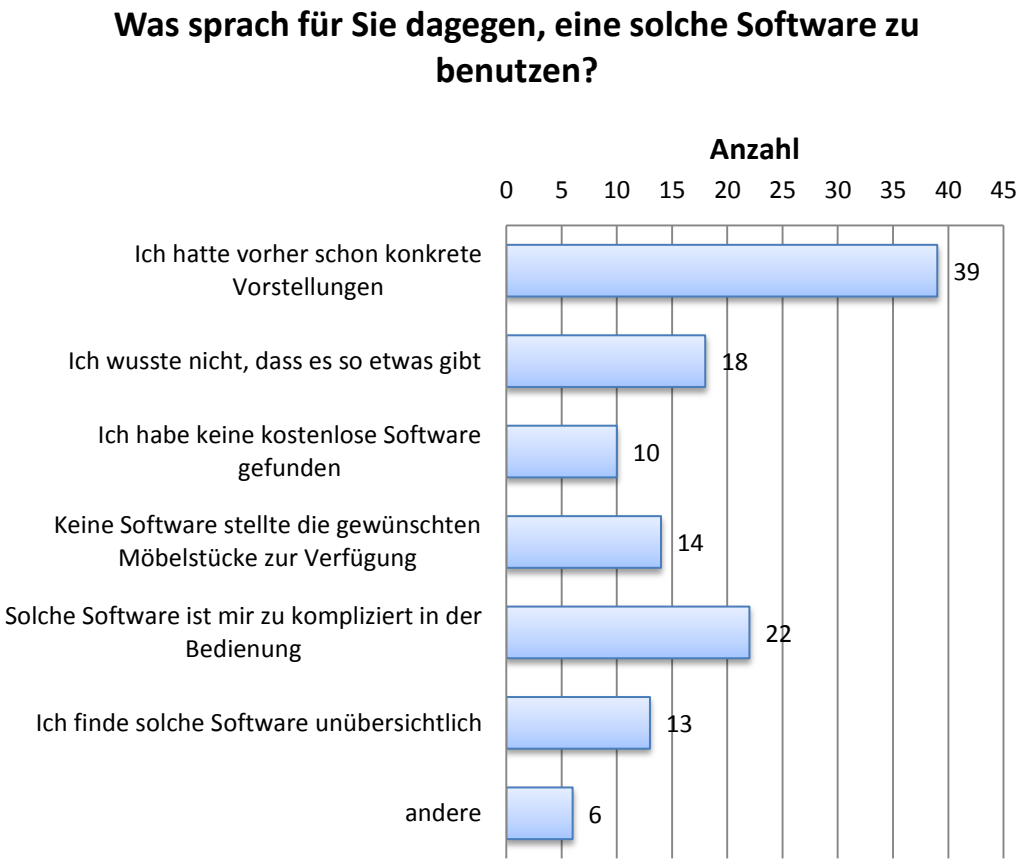
Für die meisten Benutzer einer Software zur Inneneinrichtungsplanung war diese bei der Platzierung der Möbel hilfreich. (siehe Tabelle A.3). Preisliche Vorstellungen konnten mit einer solchen Software nur selten umgesetzt werden.

Ziel	**Mittelwert der Benutzererfahrung**
Platzierung der Möbel	hat geholfen
Auswahl von Möbelstücken	hat ein wenig geholfen
Ideenfindung	hat ein wenig geholfen
Farbfindung	hat ein wenig geholfen
Umsetzung preislicher Vorstellungen	hat überhaupt nicht geholfen - hat ein wenig geholfen

Tabelle A.3.: Ziele bei der Benutzung einer Software zur Inneneinrichtungsplanung absteigend geordnet nach Benutzererfahrung

Frage 14: „Was sprach für Sie dagegen, eine solche Software zu benutzen?“ (n=76)

Beantwortung war nur möglich, wenn bei Frage 12 die Antwort „nein“ ausgewählt wurde. Mehrfache Antworten waren möglich.



Andere Gründe:

„kein interesse“

„brauche so eine software nicht“

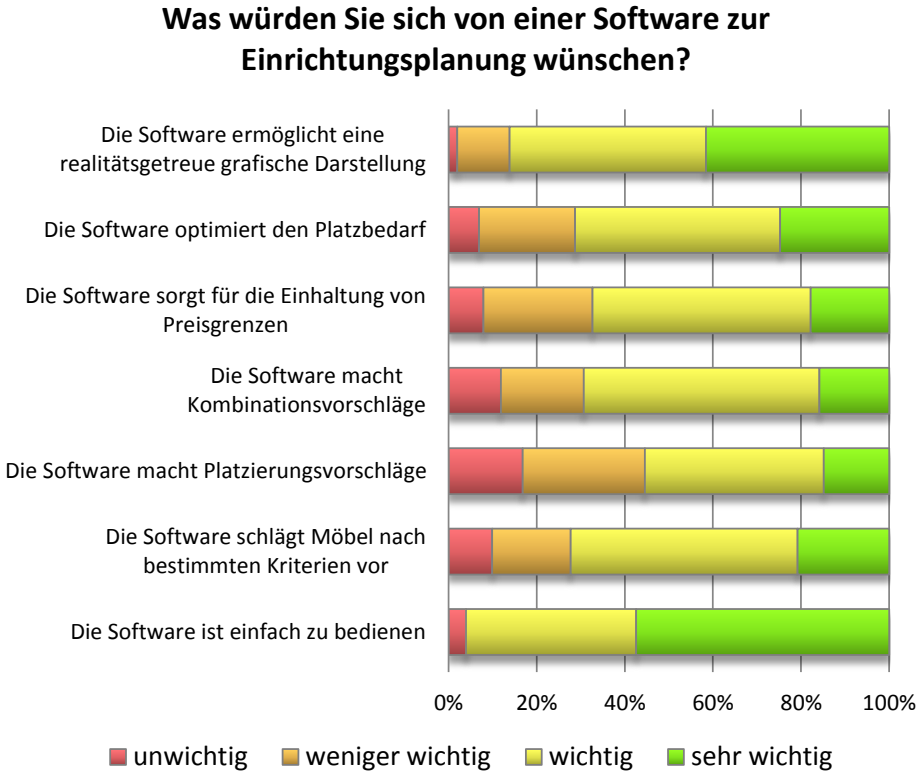
„Zettel & Stift > Plansoftware. Man weiß schon vorher in etwa, wies werden soll ;)“

„es gab weder einen anlass dazu noch hat es sich ergeben“

38,6 % aller Befragten benutzten keine Software zur Inneneinrichtungsplanung, weil sie bereits konkrete Vorstellungen hatten. Für 21,8 % aller Befragten war die komplizierte Bedienung ein Gegenargument zur Benutzung einer solchen Software.

Frage 15: „Was würden Sie sich von einer Software zur Einrichtungsplanung wünschen?“ (n=101)

Zu allen Anforderungen musste eine Antwort abgegeben werden.



Die am häufigsten genannten Anforderungen sind eine einfache Bedienung und eine realitätsgetreue Darstellung (siehe Tabelle A.4).

Anforderung	**Mittelwert**
Einfache Bedienung	wichtig - sehr wichtig
Realitätsgetreue grafische Darstellung	wichtig
Platzbedarfsoptimierung	wichtig
Vorschläge zu Möbelstücken	wichtig
Einhaltung von Preisgrenzen	wichtig
Kombinationsvorschläge	weniger wichtig - wichtig
Platzierungsvorschläge	weniger wichtig - wichtig

Tabelle A.4.: Anforderungen an eine Software zur Inneneinrichtungsplanung absteigend geordnet nach Relevanz

Frage 16: „Haben Sie weitere Anforderungen an eine solche Software, die noch nicht genannt wurden?“ (n=6)

Hier konnten Freitext-Antworten gegeben werden. Die Beantwortung war optional.

1. männlich, 25-29 Jahre, hat noch keine Software zur Inneneinrichtungsplanung benutzt, plant Einrichtung lieber selbst

   „Das Ergebnis drucken zu können und es bestenfalls sogar abspeichern zu können um es nochmals laden zu können wäre toll.“

2. weiblich, 20-24 Jahre, hat schon Software zur Inneneinrichtungsplanung benutzt, plant Einrichtung lieber selbst

   „Die Software führt mich direkt zu einer Bestellung wenn gewünscht“

3. weiblich, 30-40 Jahre, hat schon Software zur Inneneinrichtungsplanung benutzt, plant Einrichtung lieber selbst

   „wenn möglich kostenlos oder preis-leistungsverhältnis ist ensprechend“

4. männlich, 30-40 Jahre, hat schon Software zur Inneneinrichtungsplanung benutzt, plant Einrichtung lieber selbst

   „3D Funktion. Ich möchte durch den Raum laufen können um die räumliche Wirkung zu erleben.“

5. männlich, 25-29 Jahre, hat schon Software zur Inneneinrichtungsplanung benutzt, sieht sich lieber eine Inneneinrichtung an und entscheidet dann

   „Kostenlos“

6. männlich, 20-24 Jahre, hat noch keine Software zur Inneneinrichtungsplanung benutzt, sieht sich lieber eine Inneneinrichtung an und entscheidet dann

   „es sollte ein sehr breites Angebot an Möbeln verschiedenster Stile, Hersteller und Preisklassen geben“

B. XML Schema für die Konfigurationsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.
    example.org/gasettings"
  xmlns:ga="http://www.example.org/gasettings" elementFormDefault="qualified">
  <element name="gasettings" type="ga:settingsType"></element>
  <complexType name="settingsType">
    <sequence>
      <element ref="ga:setting" minOccurs="1" maxOccurs="6"></element>
    </sequence>
  </complexType>
  <element name="setting" type="ga:settingType"></element>
  <complexType name="settingType">
    <all>
      <element ref="ga:values" maxOccurs="1" minOccurs="1"></element>
      <element ref="ga:phenotypeGenerator" maxOccurs="1"
        minOccurs="1"></element>
      <element ref="ga:evaluator" maxOccurs="1" minOccurs="1"></element>
      <element ref="ga:preEvaluationState" maxOccurs="1" minOccurs="0"></element>
      <element ref="ga:postEvaluationState" maxOccurs="1" minOccurs="0"></element>
      <element ref="ga:individualFactory" maxOccurs="1"
        minOccurs="1"></element>
      <element ref="ga:population" maxOccurs="1" minOccurs="1"></element>
      <element ref="ga:selector" maxOccurs="1" minOccurs="1"></element>
      <element ref="ga:validator" maxOccurs="1" minOccurs="0"></element>
      <element ref="ga:mutationOp" maxOccurs="1" minOccurs="1"></element>
      <element ref="ga:crossoverOp" maxOccurs="1" minOccurs="1"></element>
      <element ref="ga:ga" maxOccurs="1" minOccurs="1"></element>
    </all>
  </complexType>
  <element name="values" type="ga:valuesType"></element>
  <complexType name="valuesType">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <choice>
        <element name="simpleValue" type="ga:simpleValueType"
```

```
34          maxOccurs="unbounded" minOccurs="0"></element>
35        <element name="objectValue" type="ga:objectValueType"
36          maxOccurs="unbounded" minOccurs="0"></element>
37      </choice>
38    </sequence>
39  </complexType>
40  <element name="simpleValue" type="ga:simpleValueType"></element>
41  <complexType name="simpleValueType">
42    <attribute name="id" type="string" use="required"></attribute>
43    <attribute name="type" type="string" use="required"></attribute>
44    <attribute name="value" type="string" use="required"></attribute>
45  </complexType>
46  <element name="objectValue" type="ga:objectValueType"></element>
47  <complexType name="objectValueType">
48    <sequence minOccurs="0" maxOccurs="unbounded">
49      <choice>
50        <element ref="ga:simpleParameter" maxOccurs="unbounded"
51          minOccurs="0"></element>
52        <element ref="ga:objectParameter" maxOccurs="unbounded"
53          minOccurs="0"></element>
54      </choice>
55    </sequence>
56    <attribute name="id" type="string" use="required"></attribute>
57    <attribute name="class" type="string" use="required"></attribute>
58  </complexType>
59  <element name="phenotypeGenerator" type="ga:parametersType"></element>
60  <complexType name="parametersType">
61    <sequence minOccurs="0" maxOccurs="unbounded">
62      <choice>
63        <element ref="ga:simpleParameter" maxOccurs="unbounded"
64          minOccurs="0"></element>
65        <element ref="ga:objectParameter" maxOccurs="unbounded"
66          minOccurs="0"></element>
67      </choice>
68    </sequence>
69    <attribute name="class" type="string" use="required"></attribute>
70  </complexType>
71  <element name="evaluator" type="ga:parametersType"></element>
72  <element name="preEvaluationState" type="ga:parametersType"></element>
73  <element name="postEvaluationState" type="ga:parametersType"></element>
74  <element name="individualFactory" type="ga:parametersType"></element>
75  <element name="population" type="ga:parametersType"></element>
76  <element name="selector" type="ga:parametersType"></element>
77  <element name="validator" type="ga:validatorType"></element>
78  <complexType name="validatorType">
79    <sequence minOccurs="0" maxOccurs="unbounded">
80      <choice>
81        <element ref="ga:simpleParameter" maxOccurs="unbounded"
```

```
82          minOccurs="0"></element>
83        <element ref="ga:objectParameter" maxOccurs="unbounded"
84          minOccurs="0"></element>
85        <element ref="ga:rule" maxOccurs="unbounded" minOccurs="0"></element>
86      </choice>
87    </sequence>
88    <attribute name="class" type="string" use="required"></attribute>
89  </complexType>
90  <element name="mutationOp" type="ga:parametersType"></element>
91  <element name="crossoverOp" type="ga:parametersType"></element>
92  <element name="objectParameter" type="ga:objectParameterType"></element>
93  <complexType name="objectParameterType">
94    <attribute name="index" type="integer"></attribute>
95    <attribute name="id" type="string"></attribute>
96  </complexType>
97  <element name="simpleParameter" type="ga:simpleParameterType"></element>
98  <complexType name="simpleParameterType">
99    <attribute name="index" type="integer" use="required"></attribute>
100   <attribute name="id" type="string" use="required"></attribute>
101 </complexType>
102 <element name="rule" type="ga:parametersType"></element>
103 <element name="ga" type="ga:parametersType"></element>
104 </schema>
```

Listing B.1: Schema für die XML-Konfigurationsdatei

C. Bilder der Anwendung

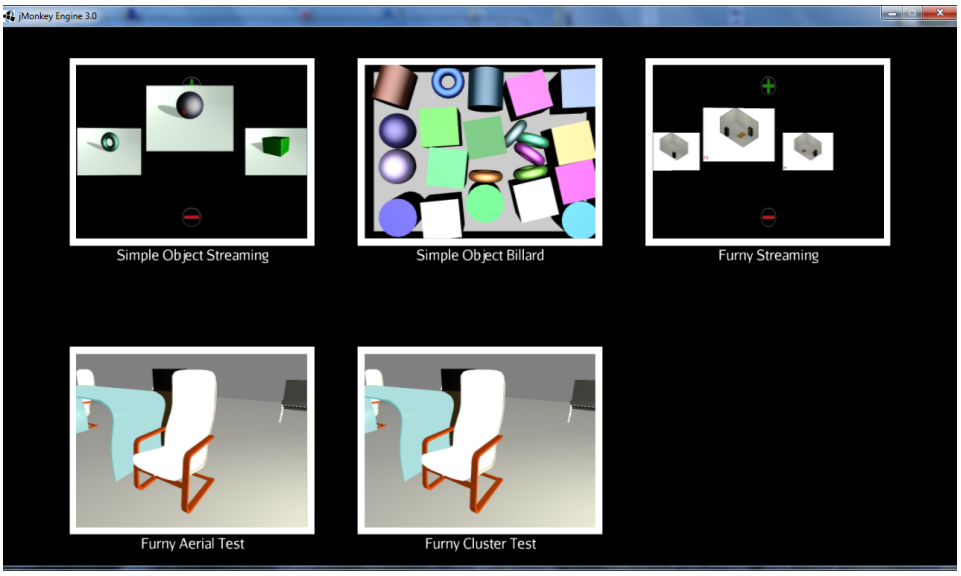


Abbildung C.1.: Auswahlmenü. Alle Konfigurationen werden hier gelistet.

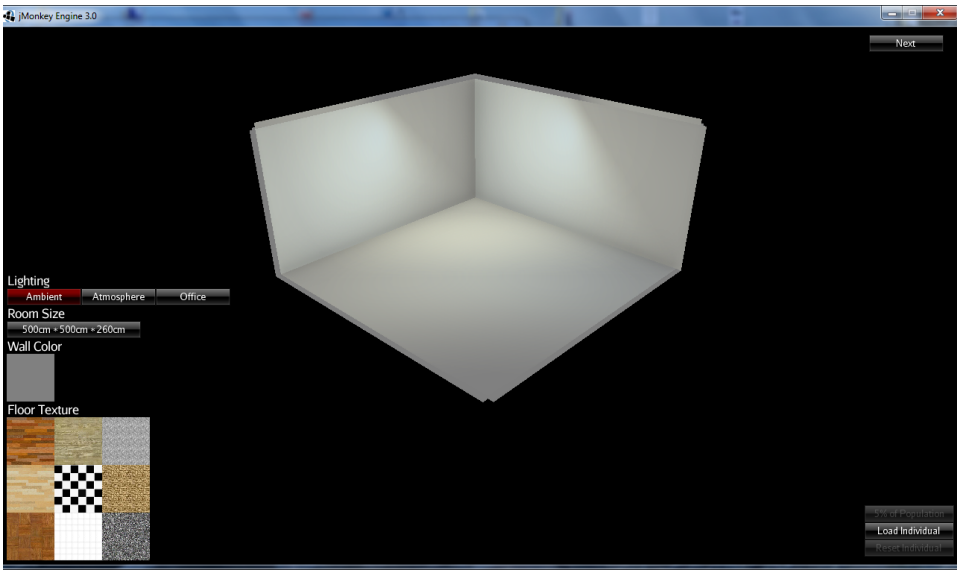


Abbildung C.2.: Konfigurationsansicht. Ein spezieller Shader erzeugt weiche Übergänge an den Kanten der Wände.

Abbildung C.3.: Tisch und Stuhl in der freien Perspektive.

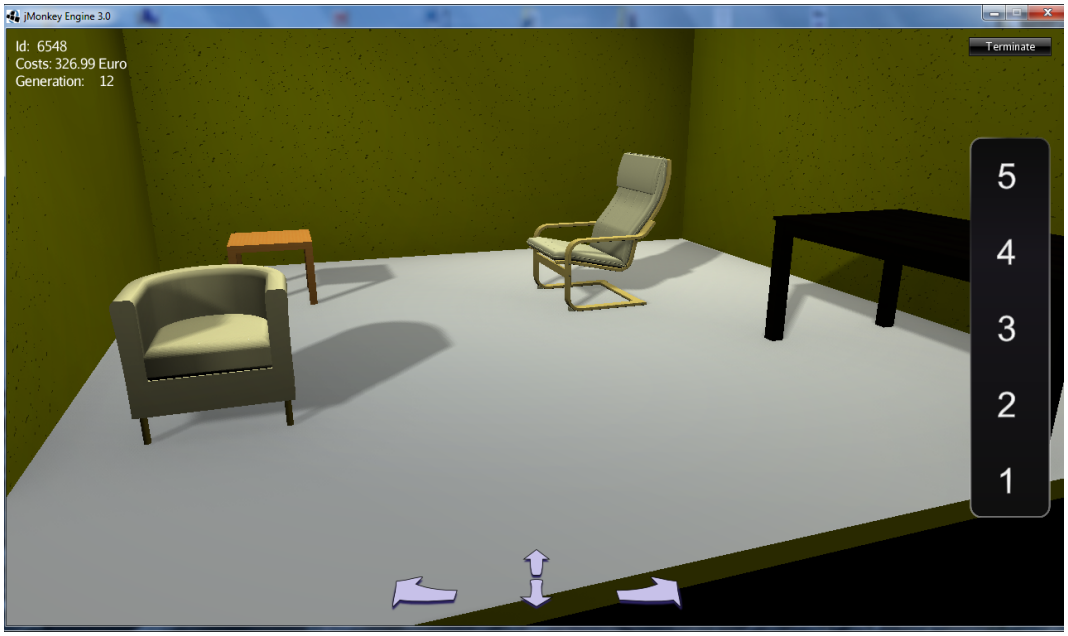


Abbildung C.4.: Mehrere Einrichtungsgegenstände in der freien Perspektive. Hier sind die Schatten der Gegenstände gut erkennbar.

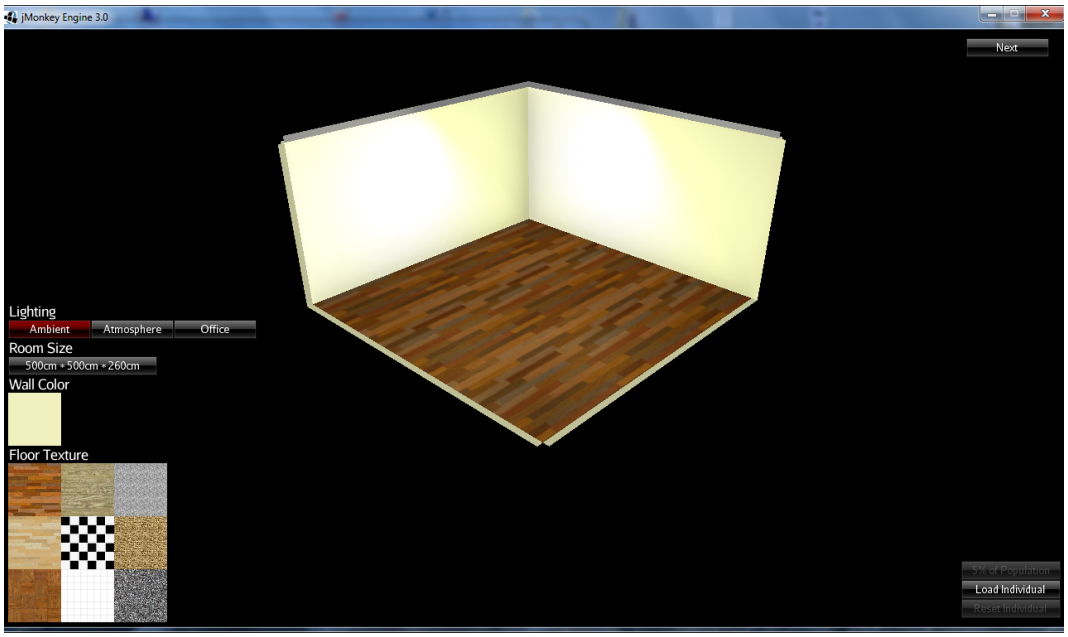


Abbildung C.5.: Wandfarbe und Bodenbelag wurden in der Konfigurationsansicht verändert.

Abbildung C.6.: Eine Einrichtung in der gedrehten Vogelperspektive. Das Bumpmapping der Wände ist links zu erkennen.

Abbildung C.7.: Couch und Tisch in der Vogelperspektive. Das Laminat hat einen leichten Glanz.

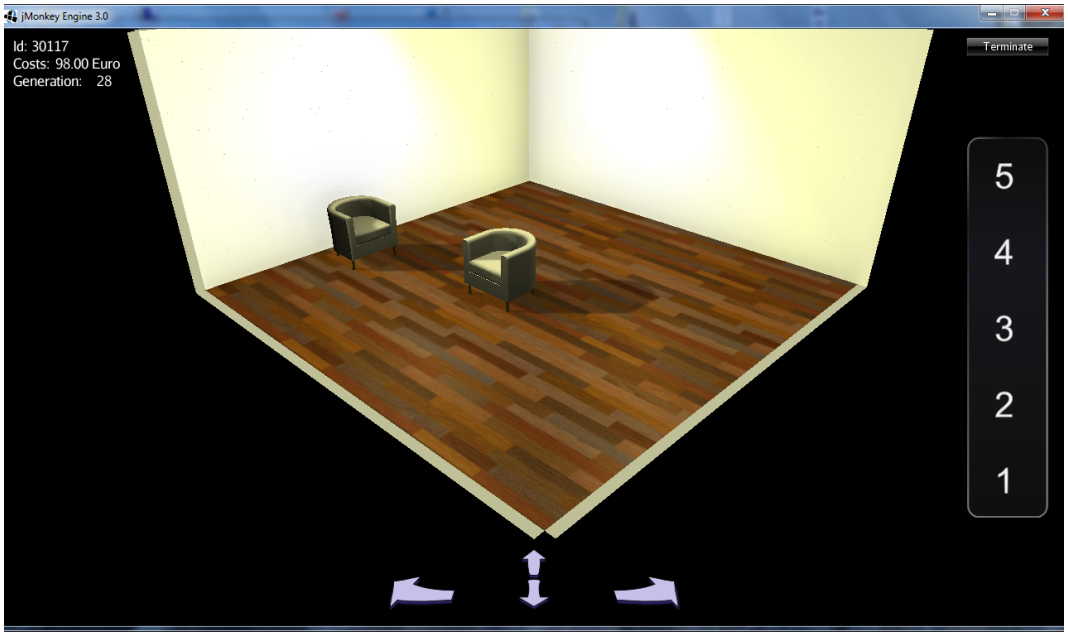


Abbildung C.8.: Zwei Sessel in der Vogelperspektive.

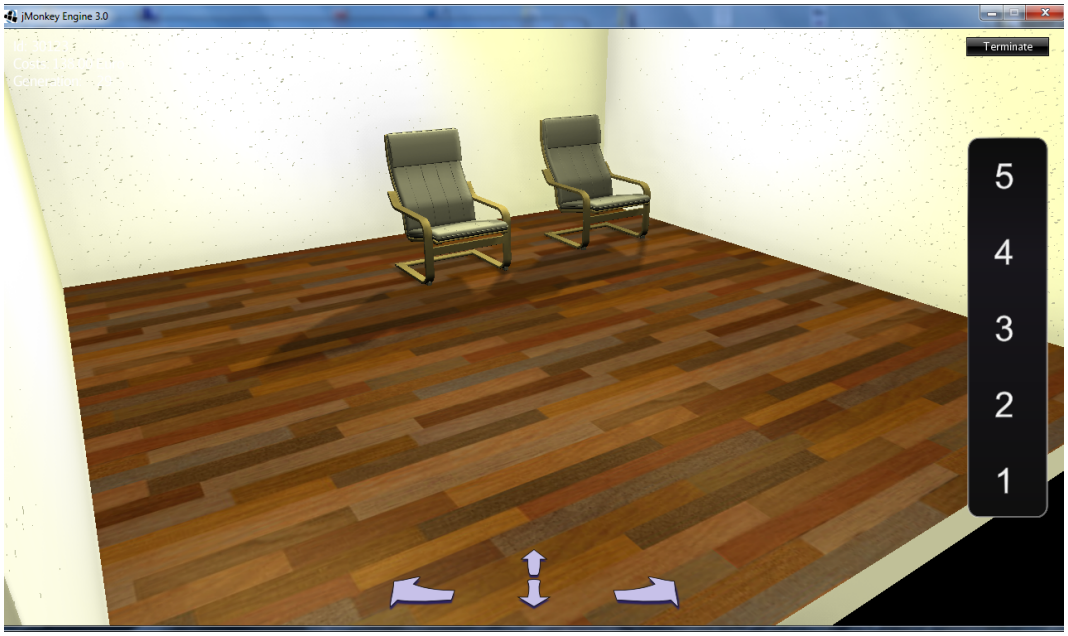


Abbildung C.9.: Zwei Schwingsessel in der freien Perspektive.

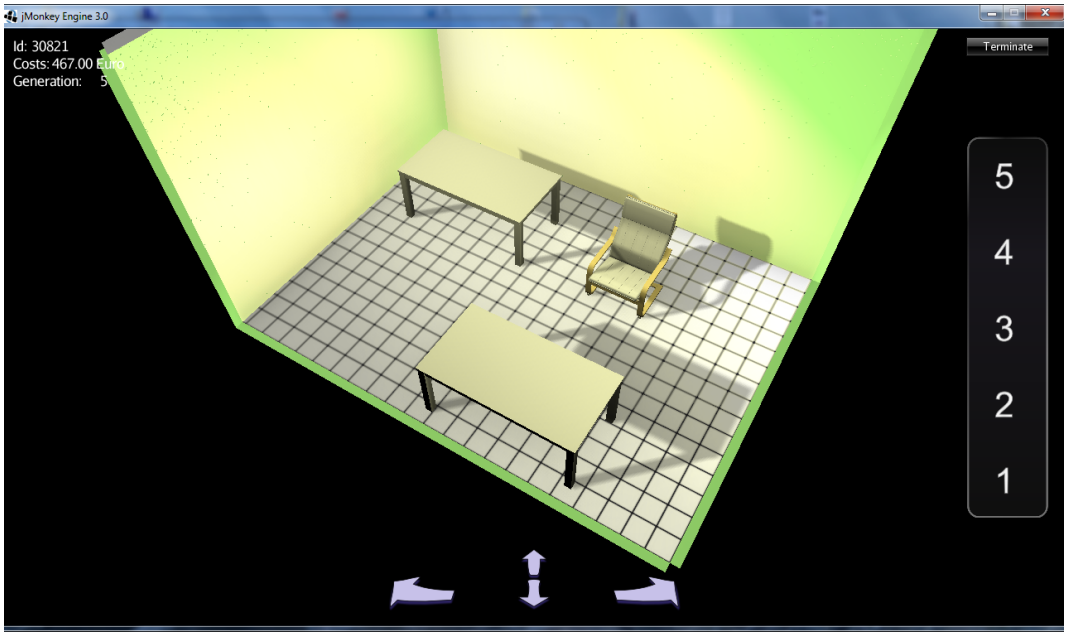


Abbildung C.10.: Ein weiteres Konfigurationsbeispiel. Der Raum ist schmal und hoch, die Wände sind hellgrün und den Boden bedecken weiße Fliesen.

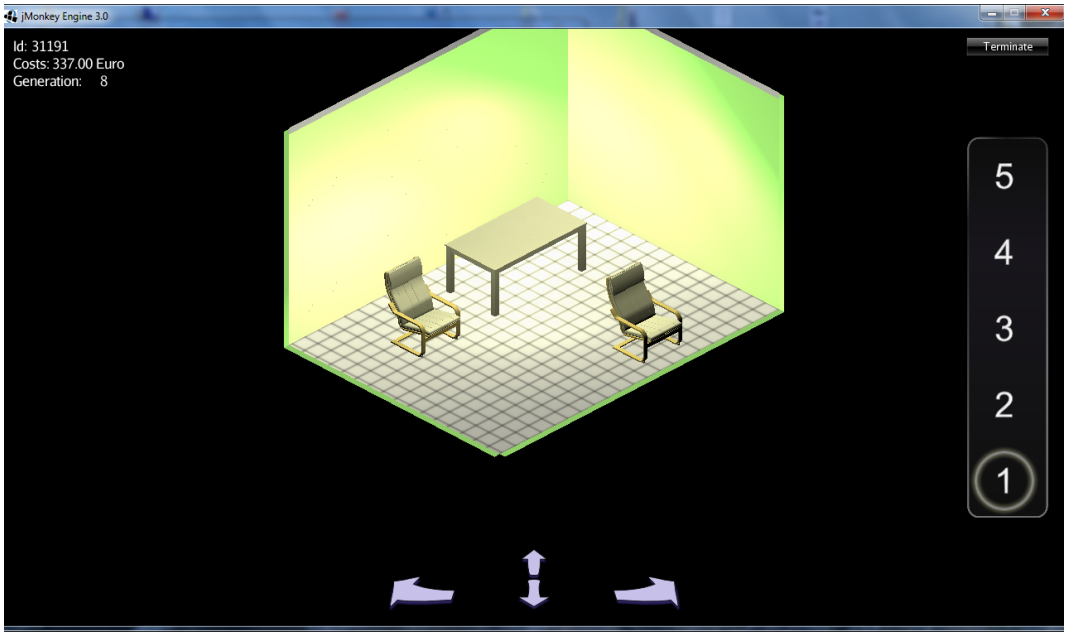


Abbildung C.11.: Parallelprojektion eines Raums. Hier werden keine Schatten angezeigt.

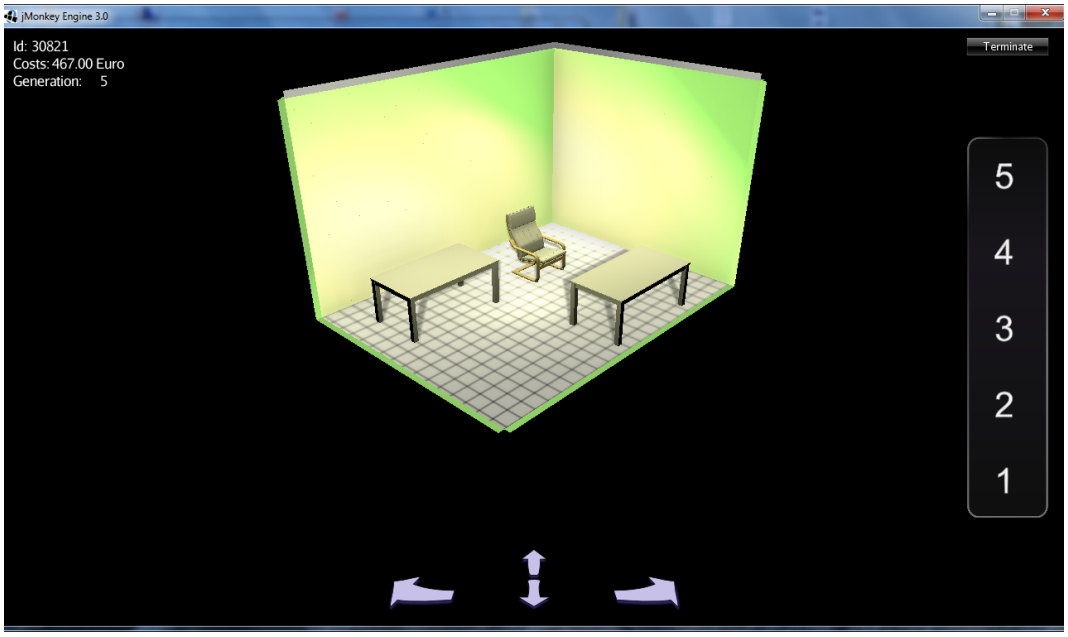


Abbildung C.12.: Raum in der Vogelperspektive.

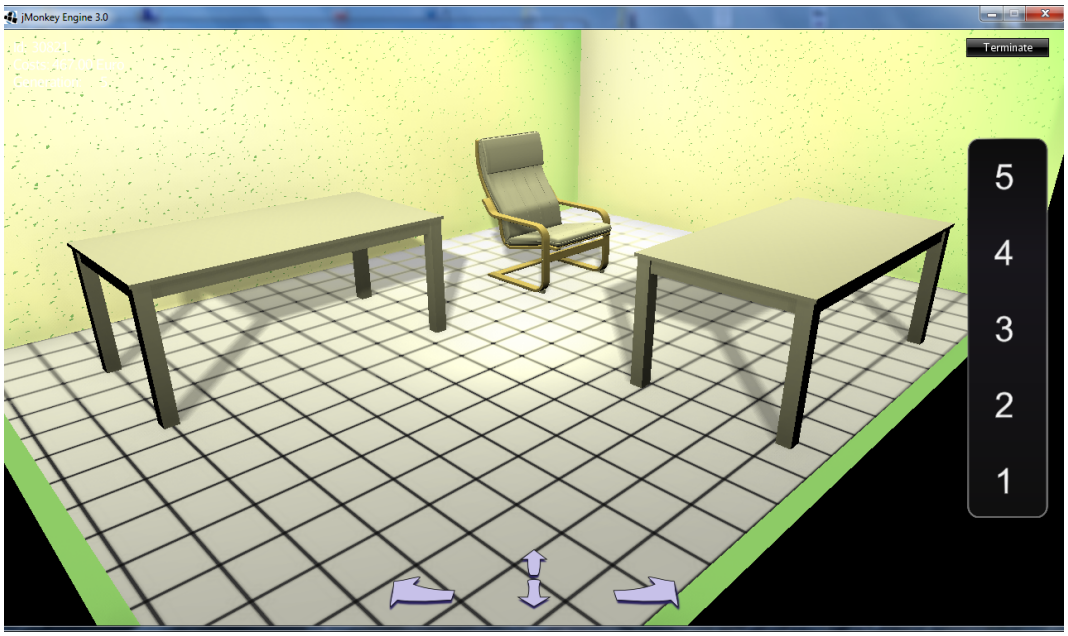


Abbildung C.13.: Der gleiche Raum wie zuvor in der freien Perspektive.

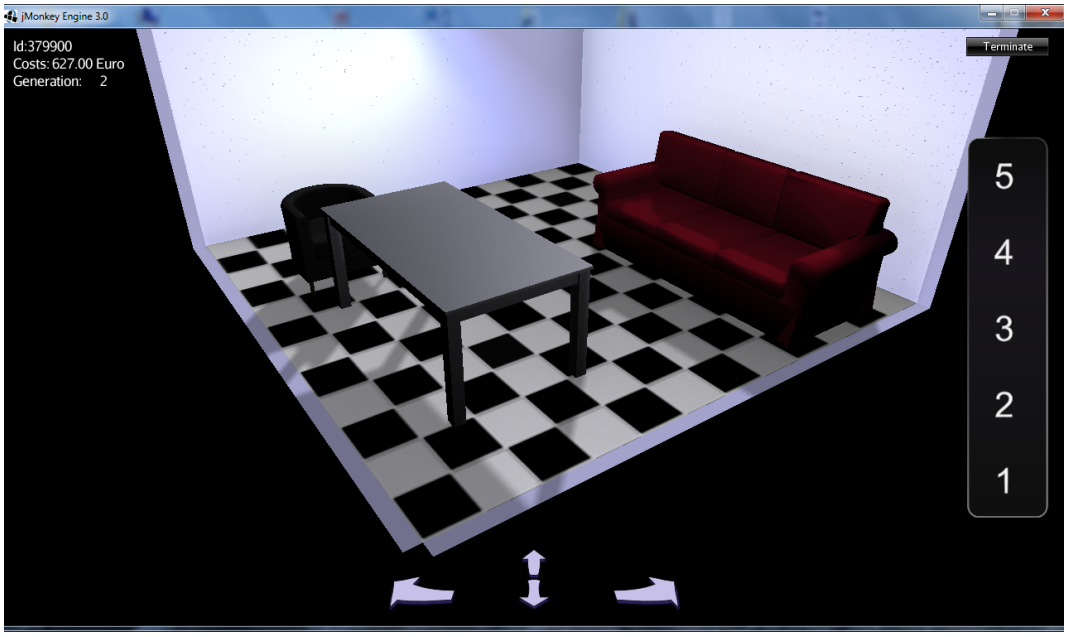


Abbildung C.14.: Fliesen mit Schachbrettmuster und hellblaue Wände.

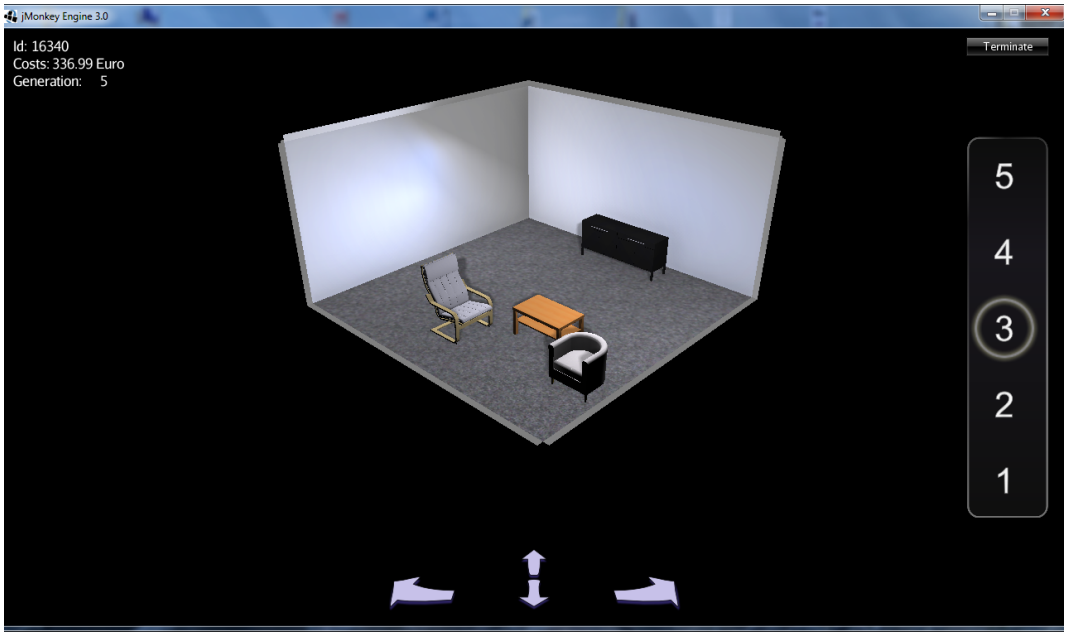


Abbildung C.15.: Ein Raum mit Büroflair. Der Teppich ist grau, die Beleuchtung ist kühl.

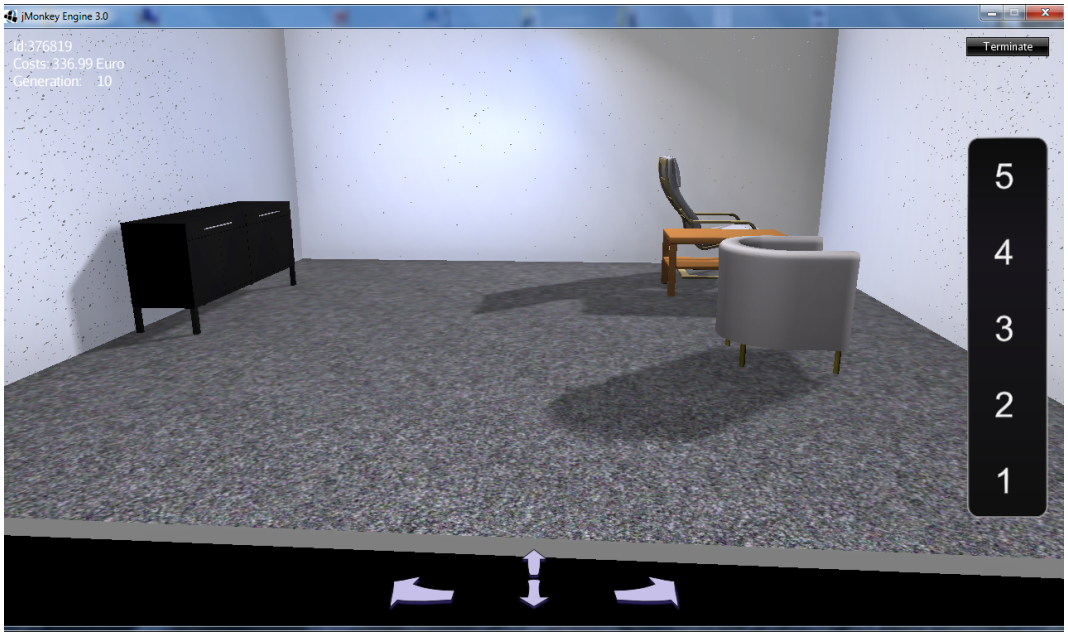


Abbildung C.16.: Der gleiche Raum wie zuvor in der freien Perspektive.

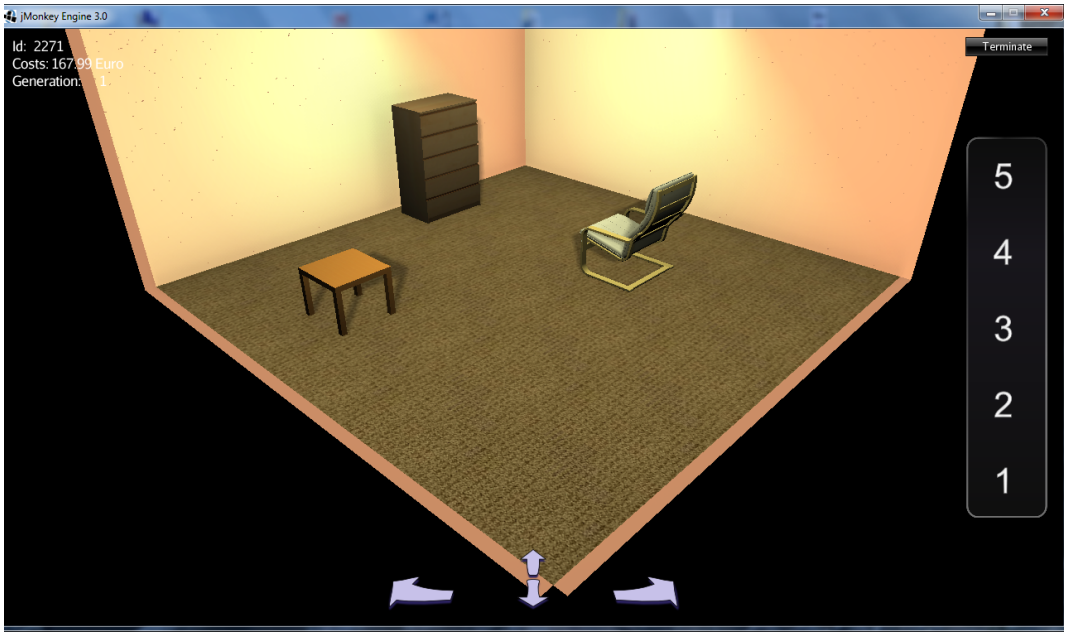


Abbildung C.17.: Warmes Licht und eine warme Wandfarbe mit einem gewebten braunen Teppich.

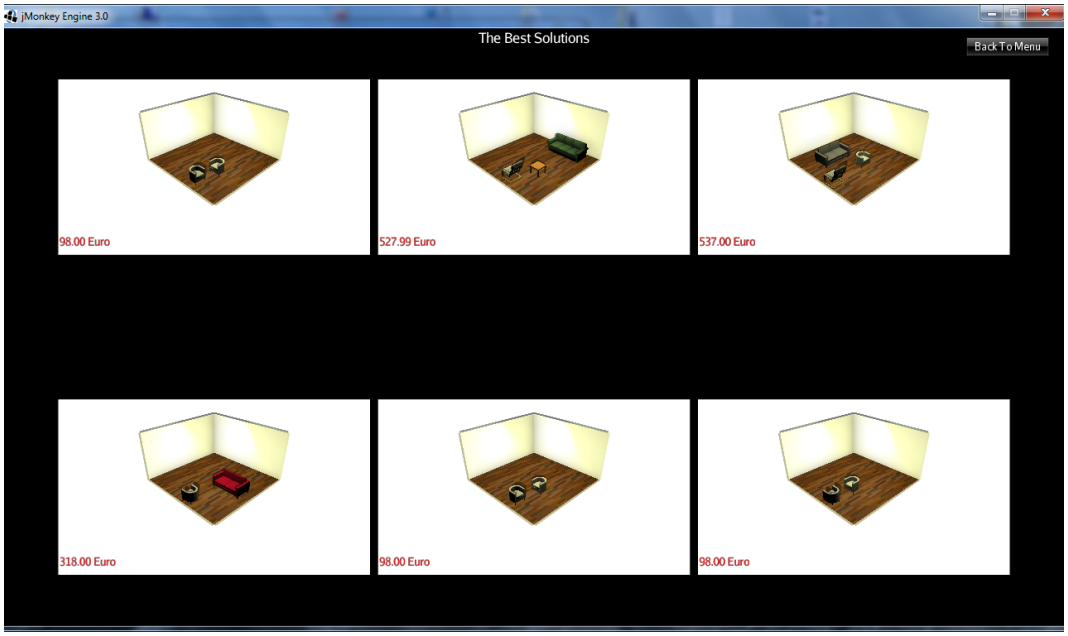


Abbildung C.18.: Die Übersicht über die besten Lösungen nach der Evaluation.

D. Inhalt der CD

Die beiliegende CD enthält die folgenden Verzeichnisse:

Verzeichnis	**Inhalt**
Arbeit	Die vorliegende Arbeit im PDF-Format.
Eclipse-Projekte	Quelltext der Softwarekomponenten gegliedert in einzelne Eclipse-Projekte.
Javadoc	Quelltext-Dokumentation der Software im HTML-Format.
Latex-Quellen	Latex-Projekt dieser Arbeit, importierbar in Eclipse (Texlipse-Plugin erforderlich).
Literatur	Frei verfügbare Literaturquellen dieser Arbeit im PDF-Format.
SketchUp	Die Software Google SketchUp 8 und das zum OgreXML-Export benötigte Plugin.
Software	Lauffähige Version der Software Furny inklusive Startdateien.