Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GUI	**G**raphical **U**ser **I**nterface
JADE	**J**ava **A**gent **D**evelopment **F**ramework
JPEG/JPG	**J**oint **P**hotographic **E**xperts **G**roup
MVC	**M**odel **V**iew **C**ontroller
PC	**P**ersonal **C**omputer
SWT	**S**tandard **W**idget **T**oolkit

Begriffsverzeichnis

Agentensystem	Mehrere Softwareagenten, die zusammen ein Problem lösen
Benutzeroberfläche	Der auf dem Bildschirm sichtbare Teil einer Software
Bilderkennung / Bildanalyse	Aus einem Foto werden technische Daten des abgebildeten Gegenstands ermittelt
Swing	GUI Framework für Java
Softwareagent	Computerprogramm mit selbstständigem Verhalten

Zusammenfassung

Das Gesamtsystem löst ein Legepuzzle mittels Softwareagenten. Den gleichen Softwareagenten wie sie in modernen Automatisierungssystemen Verwendung finden sollen. Ein Puzzle ist ein allgemein verständliches Beispiel das Verhalten dieser Softwareagenten aufzuzeigen, ohne dazu Industrieanlagen genauer kennen zu müssen. Die Aufgabe, einen sich dynamisch anpassenden Lösungsweg in einem komplexen, dezentralen und sich verändernden System mit autonom agierenden, untereinander kommunizierenden und so kooperierenden Elementen, den Softwareagenten, zu finden, ist die Gleiche.

In dieser Studienarbeit ist eine neue Benutzeroberfläche zur Bedienung des Gesamtsystems entstanden. Damit wird auch der Fortschritt des Lösevorgangs und das entstehende Puzzle angezeigt. Die Agentenkommunikation lässt sich veränderbar intensive aufzeigen.

Die einzelnen Puzzleteile werden durch Softwareagenten vertreten, welche miteinander kommunizieren, um die zugehörenden angrenzenden Nachbarpuzzleteile zu finden. Es werden zum einen das entstehende Puzzle und seine Einzelteile auf dem Bildschirm dargestellt, als auch die für das Finden der Nachbarpuzzleteile nötige Unterhaltung der Puzzleteilagenten angezeigt. Die Programmausgaben wurden so gestaltet, dass die Vorgänge auch interessierte Laien und nicht nur im System eingearbeitete Experten verständlich nachvollziehen können. Zum Filtern der Nachrichten war es nötig, diese in Klassen einzuteilen.

Abstract

The overall system solves a puzzle using software agents. The same software agents as they are to be used in modern automation systems. A jigsaw puzzle is a well-understood example to demonstrate the behavior of software agents to without having to know Industieanlagen accurate. The task of finding a dynamically adaptive approach in a complex, distributed and evolving system with autonomous actions, communicating with each other and so cooperating elements, the software agents, is the same.

In this work a new user interface to operate the entire system is created. Thus the progress of the solution process and the resulting puzzle is displayed. The agent communication can be demonstrated changeable intensive.

The puzzle pieces are represented by software agents, which communicate with each other in order to find the belonging neighboring puzzle pieces. It will be shown, the resulting puzzle and its components on the screen for one, as well as the time required for finding the neighboring puzzle pieces puzzle piece of entertainment agents displayed. The program expenditures were designed so that the operations can also reconstruct interested laymen and incorporated not only in the system experts understand. To filter the news, it was necessary to divide them into classes.

1 Einleitung

Das Gesamtsystem löst ein Legepuzzle mittels Softwareagenten. Den gleichen Softwareagenten wie sie in modernen Automatisierungssystemen Verwendung finden sollen. Ein Puzzle ist ein allgemein verständliches Beispiel das Verhalten dieser Softwareagenten aufzuzeigen, ohne dazu Industrieanlagen genauer kennen zu müssen. Die Aufgabe, einen sich dynamisch anpassenden Lösungsweg in einem komplexen, dezentralen und sich verändernden System mit autonom agierenden, untereinander kommunizierenden und so kooperierenden Elementen, den Softwareagenten, zu finden, ist die Gleiche.

In dieser Studienarbeit ist eine neue Benutzeroberfläche zur Bedienung des Gesamtsystems entstanden. Damit wird auch der Fortschritt des Lösevorgangs und das entstehende Puzzle angezeigt. Die Agentenkommunikation lässt sich veränderbar intensive aufzeigen.

Die einzelnen Puzzleteile werden durch Softwareagenten vertreten, welche miteinander kommunizieren, um die zugehörenden angrenzenden Nachbarpuzzleteile zu finden. Es werden zum einen das entstehende Puzzle und seine Einzelteile auf dem Bildschirm dargestellt, als auch die für das Finden der Nachbarpuzzleteile nötige Unterhaltung der Puzzleteilagenten angezeigt. Die Programmausgaben wurden so gestaltet, dass die Vorgänge auch interessierte Laien und nicht nur im System eingearbeitete Experten verständlich nachvollziehen können. Zum Filtern der Nachrichten war es nötig, diese in Klassen einzuteilen.

.

2 Aufgabenstellung

2.1 Zielbestimmung

In dieser Studienarbeit soll die bestehende Benutzeroberfläche des Puzzlers überarbeitet und ergänzt werden. Der Puzzler dient als Demonstrator um das Lösen eines Puzzles mittels Softwareagenten visualisieren zu können. Die einzelnen Puzzleteile sind durch Softwareagenten vertreten, welche miteinander kommunizieren um die zugehörenden angrenzenden Nachbarpuzzleteile zu finden. Dabei geht es weniger um eine optimierte Lösungsstrategie für den Lösevorgang des Puzzles, sondern um ein verständliches Aufzeigen und Darstellen der Einzelschritte in der Agentenkommunikation. Dieses soll auch für interessierte Laien und nicht nur für im System eingearbeitete Experten verständlich nachvollzogen werden können. Es soll zum einen das entstehende Puzzle und seine Einzelteile auf dem Bildschirm angezeigt werden, als auch die für das Finden der Nachbarpuzzleteile nötige Unterhaltung der Puzzleagenten im Einzelschrittablauf und sinnvoll gruppiert anzeigbar sein.

2.1.1 Musskriterien

Nach dem Einlesen und Erkennen der Puzzleteile durch die bestehenden Softwaremodule soll der Lösevorgang gestartet werden können. Danach soll der zeitliche Verlauf des Lösevorgangs transparent für Experten und interessierte Laien dargestellt werden.

2.1.2 Wunschkriterien

Die zur Lösung des Puzzles stattfindende Kommunikation der Agenten und deren Entscheidungsfindung soll in einem Art Schritt-für-Schritt Modus in Einzelschritten mit beliebiger Verzögerung zwischen den Schritten dargestellt werden. Die Lösungsstrategie des Puzzlers darf zur besseren Visualisierung umgebaut werden.

2.1.3 Abgrenzungskriterien

Es geht um die Darstellung eines Agentensystems. Das Lösen des Puzzles ist zweitrangig. Das bestehende System wird vorrangig um eine benutzbare Nutzerschnittstelle erweitert und nicht bestehende Mängel beseitigt.

2.2 Einsatz

2.2.1 Anwendungsbereiche

Der Demonstrator soll dazu dienen, die Fähigkeiten von Agenten, komplexe Probleme zu lösen, anschaulich aufzuzeigen. Im Besonderen soll der Demonstrator bei Publikumsveranstaltungen am Institut eingesetzt werden. Ein Beispiel hierfür ist der Tag der Wissenschaft.

2.2.2 Zielgruppen

Mitarbeiter und Studenten des Instituts sollen den Demonstrator Besuchern vorführen.

2.2.3 Betriebsbedingungen

Der Demonstrator soll auf Computern des Instituts ausgeführt werden. Für die Bilderfassung mittels optischem Gerät und die darauf folgende Bildverarbeitung (siehe [Jung14]) können ideale Bedingungen angenommen werden.

2.3 Umgebung

2.3.1 Software

Der Demonstrator soll auf Computern mit Java Laufzeitumgebung 1.7 ausgeführt werden.

2.3.2 Hardware

Der Demonstrator soll auf aktuellen, üblichen Computern, wie z.B. im PC- Pool des Instituts stehen, ausgeführt werden können. Zur Bilderfassung ist eine geeignete, höher auflösende Kamera oder vergleichbare Hardware zu benutzen.

2.3.3 System-Schnittstellen

Schnittstelle ist zusätzlich zu denen bei aktuellen Computern üblichen wie Monitor, Tastatur und Maus noch ein optisches Gerät zur Bilderfassung wie eine Kamera oder ein Flachbettscanner.

2.4 Funktionale Anforderungen

Die jeweiligen Punkte sind mit ihrer dortigen Reihenfolge an das Lastenheft angelehnt und wurden teilweise ergänzt.

/LFA10/ Jedes reale Puzzleteil wird automatisch generiert einzeln durch eine Instanz eines Softwareagenten vertreten.

/LFA11/ Jedes Puzzleteil muss mit den internen Zuständen des zugehörenden Agenden auf dem Bildschirm visualisiert werden.

/LFA20/ Die Agentenkommunikation soll auch für interessierte Laien verständlich aufgezeigt werden. Dafür können z.B. Gruppen gebildet werden welche sich ein- und ausblenden lassen und die sich optisch unterscheiden.

/LFA30/ Die Einzelschritte des Lösevorgangs sollen aufgezeigt werden, dazu müssen die Parameter (Zustand eines Agenten, Entscheidung eines Agenten) über eine Schnittstelle im Agenten erreichbar sein.

/LFA31/ Plan A: Schritt-für-Schritt- Modus: Dazu benötigt man ein Anhalten der Agenten.

/LFA32/ Plan B: Aufzeichnen des Lösevorgangs "wie einen Film": Dies ermöglicht ein Verlangsamen und Anhalten beim Abspielen der Aufzeichnung zum genaueren Untersuchen der Einzelschritte.

/LFA40/ Beim bestehenden System soll die Benutzeroberfläche überarbeitet werden. Es soll nicht mehr nur das fertige Puzzle angezeigt werden sondern die einzelnen Schritte des Lösevorgangs. Auch soll dieser jederzeit abgebrochen und neu gestartet werden können. Auch soll ein Wechsel zwischen Automatik- und Schritt-für-Schritt- Modus jederzeit möglich werden.

/LFA50/ Die internen Zustände und daraus folgenden Entscheidungen der jeweiligen Agenten sollen nur angezeigt bzw. aufgezeigt werden falls sie relevant für den Lösevorgang des Puzzles sind.

/LFA61/ Schon bestehende Teile des Systems, welche mit der Darstellung des Ablaufs nichts zu tun haben, sollen möglichst weiterverwendet und nicht verändert werden.

2.5 Nichtfunktionale Anforderungen

Die jeweiligen Punkte sind mit ihrer dortigen Reihenfolge an das Lastenheft angelehnt und wurden teilweise ergänzt.

/LNA10/ Da das entstehende System noch nicht das Endprodukt darstellt soll es einfach erweitert werden können.

/LNA11/ Da das entstehende System noch erweitert werden soll muss es sehr gut dokumentiert werden. Auch die Entscheidungsfindung zu den jeweils gewählten Optionen / Lösungen soll sehr gut dokumentiert werden.

/LNA20/ Da das System auch interessierte Laien beobachten werden, soll das Nachvollziehen der Bedienung des Systems auch ohne Einarbeitung zumindest für die Kernelemente möglich sein.

/LNA30/ Da das System auch interessierte Laien beobachtet werden, soll das Nachvollziehen der Visualisierung und Ausgabe der Resultate auch ohne Einarbeitung zumindest für die Kernelemente möglich sein.

/LNA40/ Die Kernelemente müssen zuverlässig und stabil laufen.

/LNA50/ Für die Visualisierung der Kernelemente soll eine Betrachtung auch aus größerem Abstand möglich sein.

2.6 Anforderungen an die Benutzungsschnittstelle

/PBA10/ Das Computersystem muss einen, nach aktuellem Stand der Technik großen, hochauflösenden Farbmonitor haben.

/PBA20/ Da die eigentliche Bilderfassung nicht in diese Arbeit fällt, wird hier nur auf eine geeignete Kamera oder ähnliches geeignete Gerät hingewiesen.

/PBA21/ Die gemachten Aufnahmen der Puzzleteile müssen als Dateien aus einem Verzeichnis gelesen werden können.

/PBA30/ Um den, nicht in das System eingearbeiteten Zuschauer nicht zu verwirren, soll das System unnötige Bedien- und Anzeigeelemente vermeiden.

/PBA40/ Um den, nicht in das System eingearbeiteten Zuschauer nicht zu verwirren, soll das System zusätzliche Bedien- und Anzeigeelemente ausblendbar gestalten

2.7 Qualitäts-Zielbestimmung

Produktqualität	sehr hoch	hoch	normal	nicht relevant
Funktionalität			x	
Richtigkeit			x	
Sicherheit (Security)				x
Interoperabilität			x	
Zuverlässigkeit			x	
Reife			x	
Fehlertoleranz			x	
Wiederherstellbarkeit			x	
Sicherheit (Safety)			x	
Benutzbarkeit	x			
Verständlichkeit	x			
Erlernbarkeit		x		
Bedienbarkeit	x			
Effizienz			x	
Zeitverhalten			x	
Verbrauchsverhalten			x	
Änderbarkeit	x			
Analysierbarkeit	x			
Modifizierbarkeit	x			
Übertragbarkeit	x			

Tabelle 2.1: Qualitäts- Zielbestimmung

2.8 Globale Testszenarien/Testfälle

2.8.1 Puzzle lösen

Bei einem ausgesuchten Testpuzzle muss der Ablauf des Lösevorgangs nachvollzogen werden können.

2.8.2 Programmbedienung

Innerhalb der Programmoberfläche muss der Lösevorgang gestartet und abgebrochen werden können. Das Hauptprogramm muss mit allen Tasks / Threads beendet werden können.

2.9 Entwicklungs-Umgebung

2.9.1 Software

Als integrierte Entwicklungsumgebung wird Eclipse verwendet. Die Programmiersprache ist Java.

2.9.2 Hardware

Die Entwicklung geschieht an Computern im PC- Pool oder vergleichbaren Computern.

2.9.3 Orgware

Als Orgware wird Microsoft Office benutzt.

2.9.4 Entwicklungs-Schnittstellen

Es gibt keine besonderen Hardware- Entwicklungs- Schnittstellen.

2.10 Durchführung

Die Arbeit wird nach dem IAS-Vorgehensmodell (Modell zur Softwareentwicklung) durchgeführt werden.

Der Stand der Arbeit und die Ergebnisse werden in regelmäßigen Abständen (ca. alle 2 Wochen) mit den Betreuern diskutiert.

Bei der Durchführung der Arbeit und der Anfertigung der Ausarbeitung werden die Richtlinien des IAS beachtet.

3 Projektplan

3.1 Projektstrukturplan

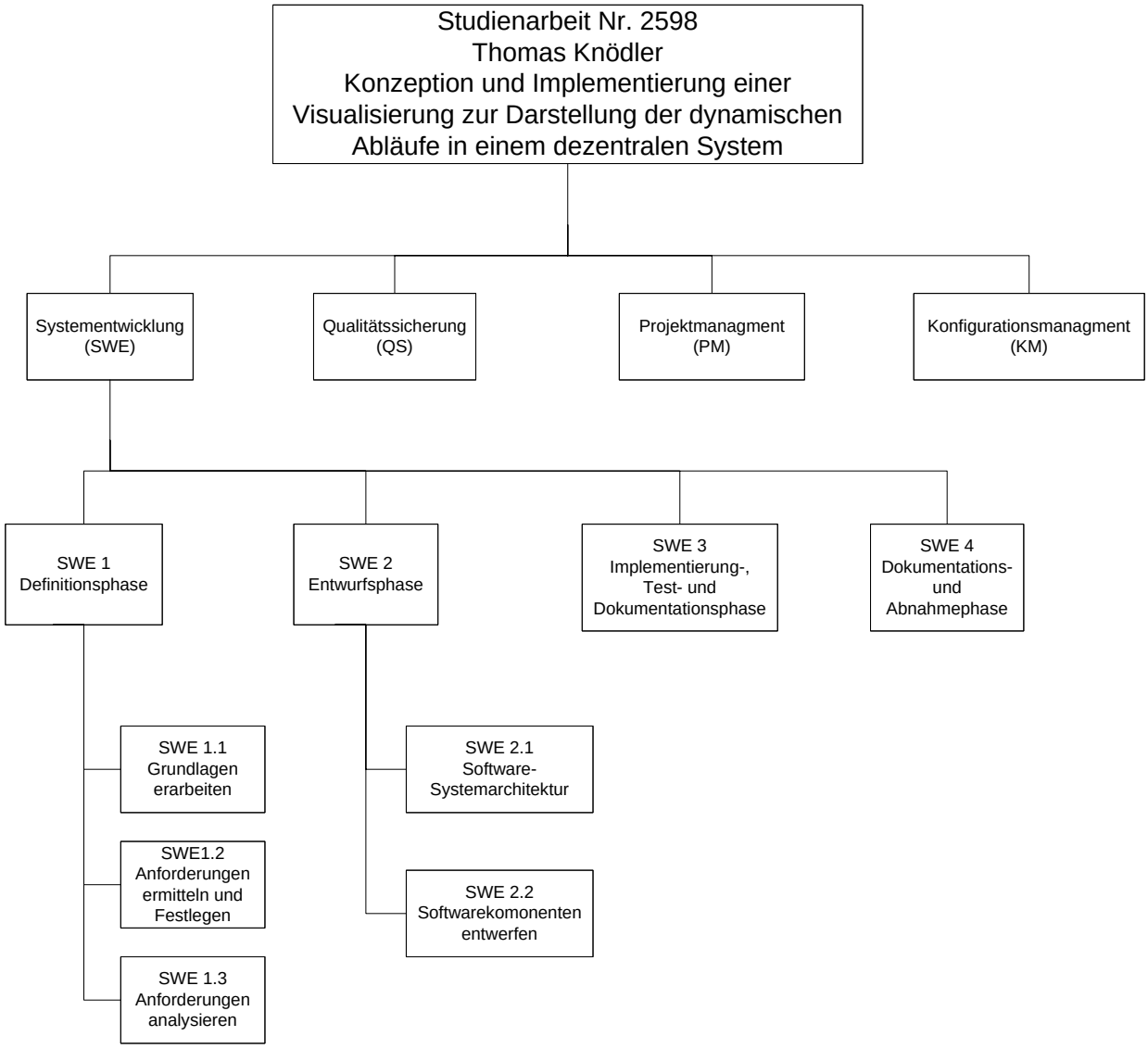

Abbildung 3.1: Use Case Diagramm: System

3.2 Arbeitspakete

3.3 Definitionsphase

3.3.1 Grundlagen erarbeiten

Es soll ein bestehendes System erweitert werden. Dazu wird eine Istzustandsanalyse durchgeführt. Das bestehende System benutzt das Agenten- Framework JADE. Es ist

notwendig, sich in JADE einzuarbeiten. Der Hauptzweck dieser Arbeit ist die Erstellung einer Visualisierung für das bestehende System. Dazu ist es notwendig, sich in die Programmierung des grafischen Benutzerinterfaces und in die Programmierung der Nebenläufigkeit (Multithreading) von Java einzuarbeiten.

3.3.2 Anforderungen ermitteln und festlegen

Die aus dem Lastenheft und durch Gespräche mit dem Betreuer entstandenen Anforderungen werden im Pflichtenheft festgehalten.

3.3.3 Anforderungen analysieren

Mit Hilfe der Analysemethode Use Cases werden die Anforderungen analysiert. Dadurch kann eine Benutzeroberfläche entworfen werden. Das Ergebnis dieses Arbeitsschrittes fließt in das Software- Systemmodell.

3.4 Entwurfsphase

3.4.1 Software- Systemarchitektur

Aus dem Systemmodell wird durch Verfeinerung und Konkretisierung die Systemarchitektur entwickelt in der auch festgelegt wird, wie die Anforderungen umgesetzt werden sollen. Auch gehört in diesen Arbeitsschritt das Festlegen der Schnittstellen der einzelnen Komponenten untereinander.

3.4.2 Software- Komponenten entwerfen

Die Systemarchitektur wird in Komponenten zerlegt und es wird festgelegt, wie diese später unabhängig voneinander umgesetzt werden können. Dabei sind auch die Rand- und Sonderfälle zu beachten und zu notieren, mit denen die späteren Testfälle ergänzt werden.

3.5 Implementierungs-, Test-, und Dokumentationsphase

Es werden die Komponenten aus der Systemarchitektur implementiert, durch die Rand- und Sonderfällen ergänzt, werden die Testfälle festgelegt und anschließend durchgeführt. Dabei wird immer auf eine vollständige Inline- Dokumentation geachtet, auch eventuelle Merkhilfen für die Hauptdokumentation werden notiert.

3.6 Dokumentations- und Abnahmephase

Die Hauptdokumentation wird überarbeitet und um Installations- und Benutzerhilfen ergänzt. Die Installations- und Benutzerhilfen werden an der laufenden Software überprüft. Dieses Gesamtwerk ist das Endprodukt.

4 Grundlegende Funktionen

4.1 Übersicht über die Anwendungsfälle

- Datei(en) auswählen
- Puzzle lösen starten
- Schritt-für-Schritt Modus ändern
- Nächster Schritt
- Puzzlelöser neu starten
- Beenden

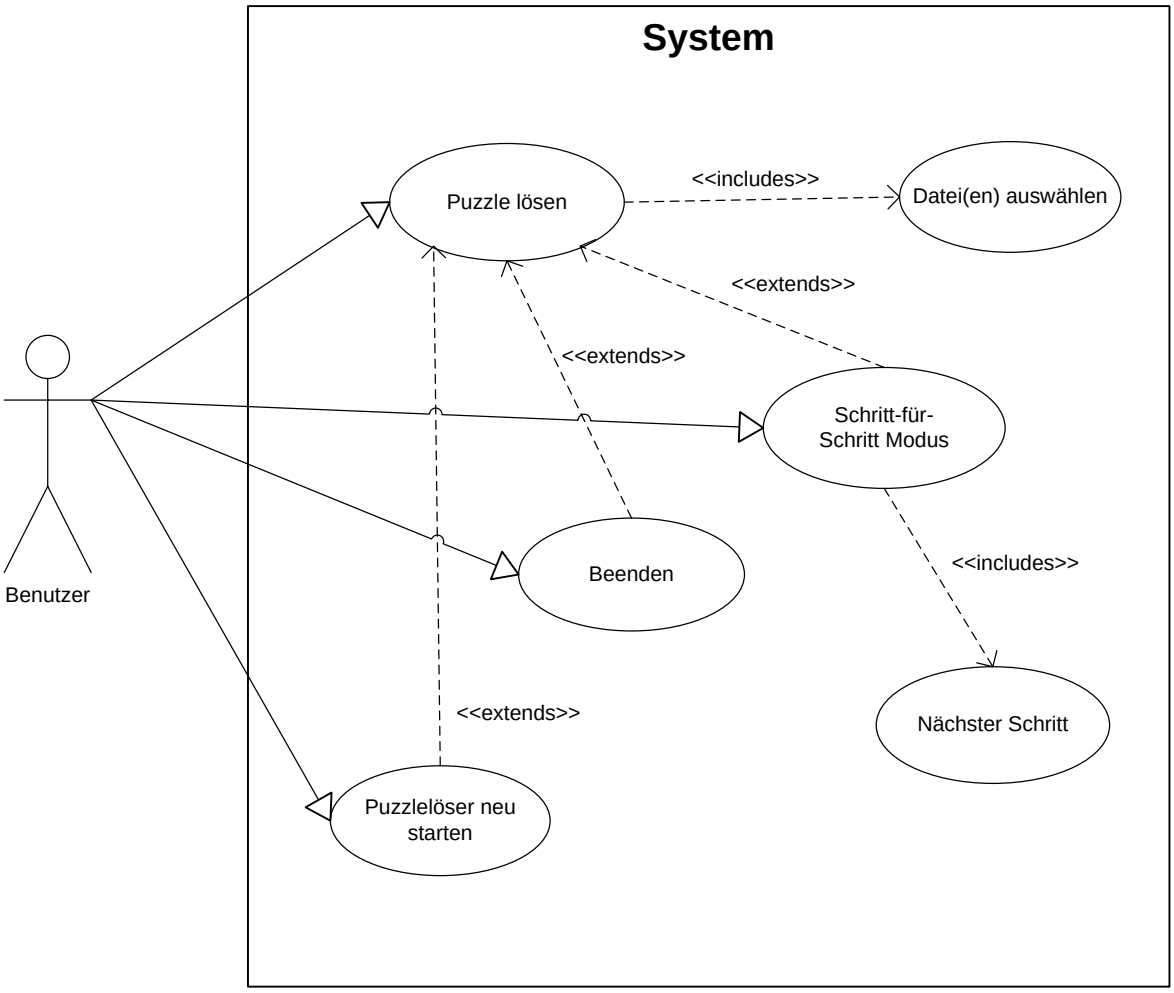

Abbildung 4.1: System

4.2 Beschreibung der Anwendungsfälle

4.2.1 Beschreibung des Anwendungsfalls „Datei(en) auswählen“

Anwendungsfall	Eine oder mehrere Dateien mit Puzzleteilen auswählen.
Ziel	Das Verzeichnis und die Dateien mit den Bildern der Puzzleteile werden ausgewählt.
Kategorie	**primär**
Externe Akteure	Benutzer
Vorbedingung	1. Das Programm muss ordnungsgemäß gestartet sein. 2. optional: „Puzzlelöser neu starten“ wurde gewählt.
Nachbedingung Erfolg	Mindestens eine Datei in einem Verzeichnis wurde zur Liste der Dateien hinzugefügt.
Nachbedingung Fehlschlag	Es wurde keine Datei zur Liste der Dateien hinzugefügt.
Auslösendes Ereignis	Der „Datei auswählen“- Button wurde gedrückt.
Beschreibung	Ein „Datei öffnen Dialog“ öffnet sich und es wird eine Datei in einem Verzeichnis gesucht und ausgewählt. Diese Datei wird in die Liste der Quelldateien hinzugefügt. Dieser Vorgang kann beliebig oft durchlaufen werden. Eine falsch gewählte Datei kann nur mit einem „Puzzlelöser neu starten“ rückgängig gemacht werden.
Erweiterungen	Keine
Alternativen	Keine

Tabelle 4.1: Beschreibung des Anwendungsfalls „Datei(en) auswählen“

4.2.2 Beschreibung des Anwendungsfalls „Puzzle lösen starten“

Anwendungsfall	Der Puzzlelösevorgang wird gestartet.
Ziel	Das Puzzle wird gelöst und die Lösung angezeigt.
Kategorie	**primär**
Externe Akteure	Benutzer
Vorbedingung	Mindestens eine Datei wurde durch „Datei(en) auswählen“ ausgewählt.

Nachbedingung Erfolg	Die Ergebnisse werden angezeigt.
Nachbedingung Fehlschlag	Es wird eine entsprechende Fehlermeldung ausgegeben.
Auslösendes Ereignis	Der "Puzzle lösen"- Button wurde gedrückt.
Beschreibung	Es wird versucht das Puzzle zu lösen. Die Kommunikation der Puzzleagenten wird angezeigt. Die Ergebnisse werden schrittweise angezeigt.
Erweiterungen	Keine
Alternativen	Den "Puzzlelöser neu starten"- Button drücken.

Tabelle 4.2: Beschreibung des Anwendungsfalls „Puzzle lösen starten“

4.2.3 Beschreibung des Anwendungsfalls „Schritt-für-Schritt Modus ändern“

Anwendungsfall	Der Schritt-für-Schritt- Modus wird gewechselt.
Ziel	Der Schritt-für-Schritt Modus wechselt zwischen aktiviert und nicht aktiviert.
Kategorie	**primär**
Externe Akteure	Benutzer
Vorbedingung	Keine
Nachbedingung Erfolg	Der Schritt-für-Schritt Modus wurde gewechselt.
Nachbedingung Fehlschlag	Der Schritt-für-Schritt Modus wurde nicht gewechselt.
Auslösendes Ereignis	Der "Schritt-für-Schritt Modus wechseln"- Button wurde gedrückt.
Beschreibung	Der Schritt-für-Schritt Modus wechselt zwischen aktiviert und nicht aktiviert. Die Beschriftung des Buttons wechselt entsprechend zwischen "Schritt-für-Schritt Modus aktivieren" und "Schritt-für-Schritt Modus deaktivieren".

Erweiterungen	Keine
Alternativen	Keine

Tabelle 4.3: Beschreibung des Anwendungsfalls „Schritt für Schritt Modus ändern“

4.2.4 Beschreibung des Anwendungsfalls „Nächster Schritt“

Anwendungsfall	Im Schritt-für-Schritt Modus wird zum nächsten Schritt geschalten.
Ziel	Den nächsten Schritt anzeigen.
Kategorie	**primär**
Externe Akteure	Benutzer
Vorbedingung	Die Software muss in den Schritt-für-Schritt Modus geschalten sein.
Nachbedingung Erfolg	Der nächste Schritt wird angezeigt.
Nachbedingung Fehlschlag	Der nächste Schritt wird nicht angezeigt.
Auslösendes Ereignis	Der "nächste Schritt"- Button wurde gedrückt.
Beschreibung	Im Schritt-für-Schritt Modus wird einen Schritt weiter ausgeführt und angezeigt.
Erweiterungen	Keine
Alternativen	Den Schritt-für-Schritt Modus beenden.

Tabelle 4.4: Beschreibung des Anwendungsfalls „Nächster Schritt“

4.2.5 Beschreibung des Anwendungsfalls „Puzzlelöser neu starten“

Anwendungsfall	Die Software wird neu Initialisiert.
Ziel	Die Software neu initialisieren.
Kategorie	**sekundär**
Externe Akteure	Benutzer

Vorbedingung	Keine, auch nicht: „Der Lösevorgang muss gestartet worden sein“, damit dadurch ein Reset der Dateiauswahl möglich wird.
Nachbedingung Erfolg	Das Programm ist neu initialisiert.
Nachbedingung Fehlschlag	Das Programm wurde nicht initialisiert.
Auslösendes Ereignis	Der "Puzzlelöser neu starten"- Button wurde gedrückt.
Beschreibung	Die Software soll in den Status wie direkt nach dem Start gehen, also noch ohne Benutzerinteraktion. Dies ist z.B. bei einer Fehlbedienung oder einer Fehllauf des Lösealgorithmus nötig.
Erweiterungen	Keine
Alternativen	Keine

Tabelle 4.5: Beschreibung des Anwengungsfalls „Puzzlelöser neu starten“

4.2.6 Beschreibung des Anwendungsfalls „Beenden“

Anwendungsfall	Die Software soll beendet werden.
Ziel	Die Software mit allen Threads soll beendet werden.
Kategorie	**primär**
Externe Akteure	Benutzer
Vorbedingung	Keine
Nachbedingung Erfolg	Die Software mit allen Threads ist beendet.
Nachbedingung Fehlschlag	Keine
Auslösendes Ereignis	Der "Beenden"- Button wurde gedrückt.
Beschreibung	Die Software soll mit allen Threads beendet werden.
Erweiterungen	Keine

Alternativen	Keine

Tabelle 4.6: Beschreibung des Anwendungsfalls „Beenden“

4.3 GUI-Konzept

4.3.1 Prinzipielle Bedienkonzepte

Um auch vor Publikum gezeigt werden zu können, soll der Demonstator möglichst einfach gehalten werden und auch zu fein auflösende Darstellungen vermeiden. Da aber der interessierte Laie auch nicht gelangweilt werden sollte, ist trotzdem auf eine lückenlos nachvollziehbare Lösung des Puzzles zu achten. Deshalb sind großflächige Symbole dem tatsächlichen Statuscode vorzuziehen. Die Anzeige ist auf den normalen Durchlauf der Lösung zu optimieren, sollte aber bei Sonderfällen nicht an Übersicht und Verständlichkeit verlieren. Im Notfall besser den Vorgang mit "Puzzle nicht lösbar" abbrechen als Chaos anrichten.

4.3.2 Elemente der Benutzungsschnittstelle

Der Lösevorgang wird mit dem "Start"- Button ausgelöst, das Puzzle wird in Einzelschritten gelöst und dies angezeigt. Der "Schritt-für-Schritt"- Button wechselt den Anzeigemodus zwischen einfacher Anzeige und manuell weiterschaltbaren Einzelschritten. Einen Button für eine noch tiefergehende Auflösung als in Einzelschritten wie z.B. das Anzeigen des Extrahierens der Puzzleteilmerkmale, gibt es nicht. Auch gibt es nur Schritt-für-Schritt Modus (manuelle Einzelschritte) und keine Button für schnelles, langsames oder normales Durchlaufen. Um den nächsten Schritt auslösen gibt es einen "nächster Schritt"- Button. Die Wirkung dieses Buttons soll im nicht Dedugmodus keinen Einfluss haben. Durch den "Datei(en) auswählen"- Button werden eine oder mehrere Dateien, welche Fotos von Puzzleteilen enthalten, ausgewählt. Erst ab mindestens einer ausgewählten Datei soll der Lösevorgang durch den Start Button ausgelöst werden können. Nach dem Auslösen sollen keine weitere Dateien mehr ausgewählt werden können. Beide Verriegelungen sollen durch entsprechende Hinweise auch mitgeteilt werden. Der Gesamtvorgang lässt sich durch den "Neustart"- Button reseten und damit neu beginnen. Dabei geht nur die Auswahl der Dateien aber nicht die Dateien selber verloren und alle Infomeldungen werden gelöscht. Durch den Log- Level Auswahlschalter kann das Filtern der Log- Meldungen eingestellt werden.

4.4 Fensterstruktur und –gestaltung

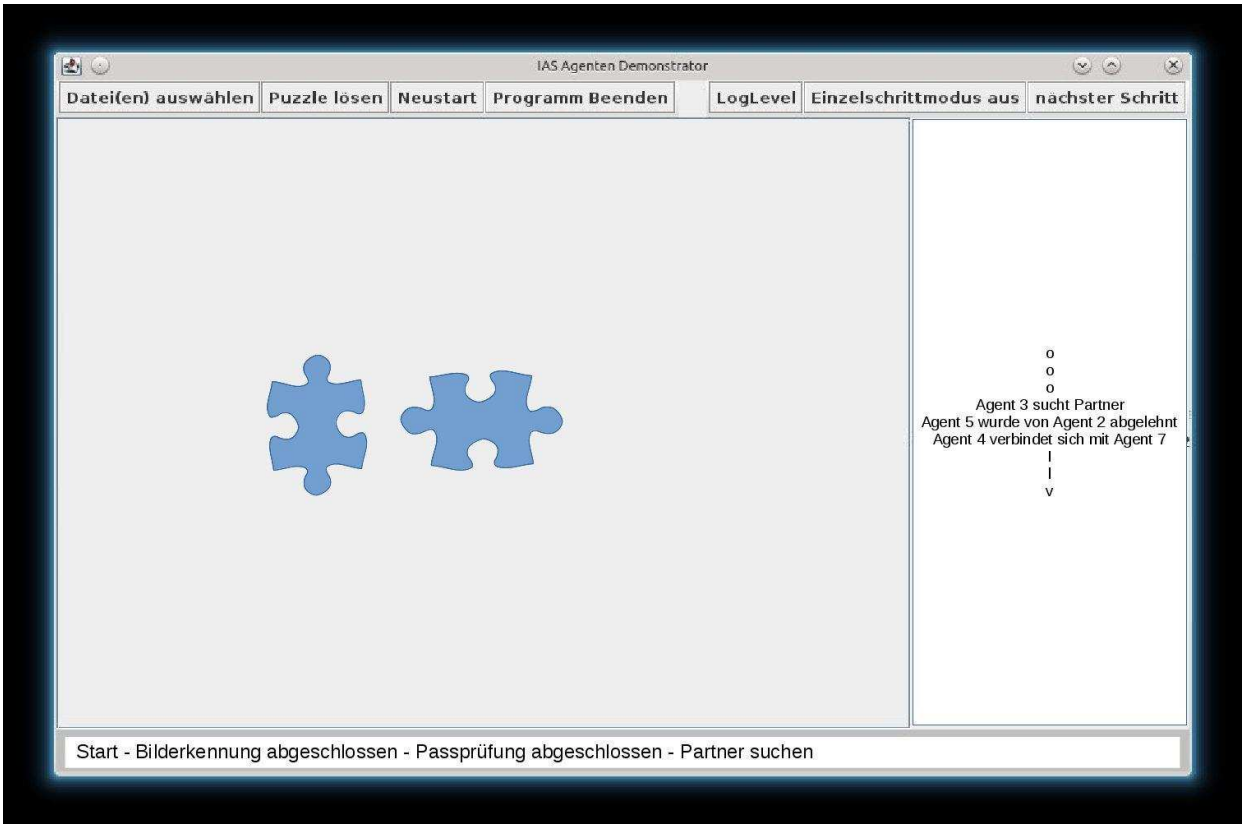

Abbildung 4.2: Hauptfenster

Abbildung 4.2 zeigt das Hauptfenster. Die Darstellung des entstehenden Puzzles erfolgt auf der linken, großen Fläche. Die einzelnen Meldungen der Puzzleteilkommunikation werden im rechten Streifen angezeigt. Die Bedienelemente werden in einer Leiste am oberen Fensterrand angeordnet. Dabei sind die Elemente für den Schritt-für-Schritt Modus rechtsbündig, die anderen linksbündig zu sortieren. Am unteren Rand wird der Fortschritt des Vorgangs durch eine Status- Zeile sichtbar gemacht. Es werden die einzelnen Teilschritte in ihrer Reihenfolge dargestellt und der aktuelle Teilschritt hervorgehoben.

Weitere Ausgabefenster als das Hauptfenster soll es, zumindest für die Publikumsvorführung nicht geben. Für interne Zwecke ist eine Ausgabe von Meldungen auf der Konsole oder in eine Logdatei denkbar.

4.5 Dialoggestaltung

Als Dateiauswahldialog wird der Standarddialog der verwendeten Programmieroberfläche benutzt. Die Fehlermeldungen dieses Dialogs werden von Popups angezeigt.

Die Agentenkommunikation erfolgt als Auflistung zeitlich sortiert von oben (älter) nach unten (jünger) in der Form:

Puzzleteil 1 <--- Meldung ---> Puzzleteil 2

Für die Puzzleteilinseln wird die gleiche Form benutzt. Als mögliche Meldungen sind folgende Text (Beispiele) denkbar: nehmen Kontakt auf; überprüfen zusammenpassen; passt nicht, beenden Kommunikation; passen zusammen, bilden Insel.

Beispiel für einen Ausschnitt der Agentenkommunikation:

P3 <-- nimmt Kontakt auf --> P8

P3 <-- überprüft zusammenpassen --> P8

P3 <-- passt nicht, beenden Kommunikation --> P8

P5 <-- nimmt Kontakt auf --> P14

P5 <-- überprüft zusammenpassen --> P14

P5 <-- passen zusammen, bilden Insel --> P14

Farbliche gekennzeichnet kann entweder der Zyklus (Verbindung aufnehmen, Test, Verbindung beenden) einheitlich werden und somit eine Unterscheidung der Zyklen bewirken, oder jeder Aufgabenteil immer gleich gefärbt werden und somit ein Muster der Tätigkeiten entstehen. Die Wirkung beider Möglichkeiten auf den Betrachter soll durch Tests ermittelt werden.

Die möglichen Log- Meldungen werden mit einer Priorität versehen (Log- Level). Damit ist eine Filterung und dadurch eine Veränderung des Meldungsumfangs zwischen wenig Meldungen zur Übersichtserhöhung oder mehr Meldungen für detailreichere Auskunft möglich

5 Entwurf

5.1 Umgebungs- und Randbedingungen

Der Demonstrator soll vor Publikum das Lösen eines Puzzles mit Hilfe eines Agentensystems aufzeigen. Die noch zu entwickelnde grafische Benutzeroberfläche soll um möglichst viel aus den schon vorhandenen Projekten "Bilderkennung" und "Prototyp des Demonstrator" ergänzt werden, um das Gesamtsystem zu erhalten. Im entstehenden Demonstrator soll das entstehende Puzzle aber auch die Agentenkommunikation dargestellt werden.

5.2 Grundlegende Entwurfsentscheidungen

Die Fotos der Puzzleteile werden in einem Verzeichnis zur Verfügung gestellt. Das Aufnehmen der Puzzleteilfotos mit einer an den Demonstratorcomputer angeschlossenen Hardware und deren Steuerung wird erst für eine der nächsten Versionen angedacht. Die Software wird in Java, die grafische Oberfläche in Java/Swing entwickelt. Die in [Emri13] verwendeten SWT Komponenten sind, so weit möglich, ganz zu entfernen, damit das Gesamtsystem ohne Plattformabhängigkeiten auskommt. Bei dieser Art Software bietet es sich an, die einzelnen Komponenten aus der GUI- Komponente heraus aufzurufen und zu steuern, welche somit eine zentrale Rolle einnimmt. Der klassische MVC Entwurf wird also in einen GUI - Logic - Datenhaltung geändert. Die Logic Komponente teilt sich noch einmal in Logic, Helper, Recognition und Solver. Die beiden Letzten beinhalten Wrapper zu den jeweiligen Vorprojekten bzw. deren Entsprechung. Es wird eine Komponente RecognitionSimSer geben welche mit einer Simulation eine Serialisierung des Vorgangs ermöglicht. Die Serialisierung dient dazu, die Einzelvorgänge nicht gleichzeitig sondern getrennt und hintereinander ablaufen lassen zu können und somit für den Zuschauer verständlicher zu machen. Da der Demonstrator für beobachten durch interessierte Laien entwickelt wird soll das System sich nicht in Details verlieren sondern ein gewisses Maß an Vereinfachung in der Darstellung tolerieren.

5.3 Diagramm der Software-Systemarchitektur

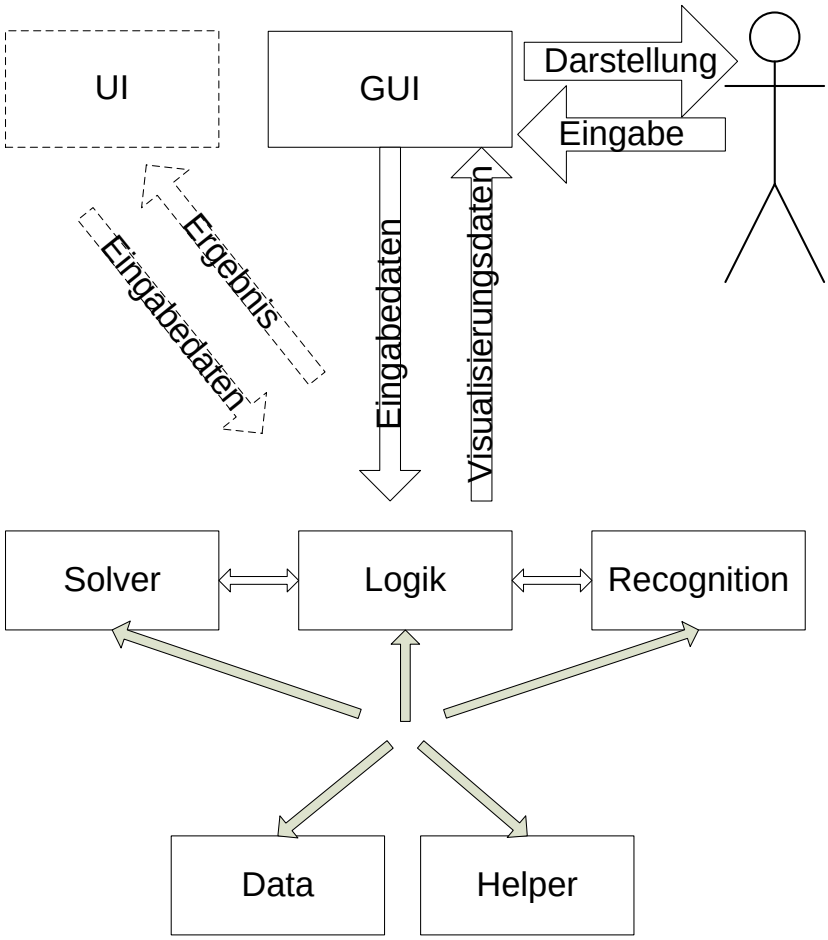

Abbildung 5.1: Systemarchiterktur

Abbildung 5.1 zeigt den Datenfluss zwischen den einzelnen Komponenten. Der Benutzer macht seine Eingaben in der GUI. Diese leitet die Wünsche an die Logic Komponente weiter, welche die einzelnen Schritte ausführt. Die Ergebnisse werden wieder an die GUI weiter gegeben und dort angezeigt. Die Logic- Komponente lädt die Fotos, führt die nötigen Teilschritte zum Lösen des Puzzles durch und speichert die entstehenden Daten in der Daten- Komponente zwischen.

5.4 Softwarekomponenten

5.4.1 GUI

Diese Komponente beinhaltet die Kommunikation mit dem Benutzer. Er kann hier den Ablauf steuern und bekommt die Ergebnisse angezeigt. Da diese Komponente so zentral ist übernimmt sie auch den Rest der prinzipiellen Ablaufsteuerung, verteilt aber dazu Details an Unterkomponenten im Block Logic. Bei der Aufteilung soll darauf geachtet werden, dass das Puzzle-lösen auch ohne grafische Komponente, nur mit einer gedachten Komponente UI (also nur mit Ablaufsteuerung) möglich wäre, z.B. für einen Batch- Betrieb. Die Komponente UI wird nicht zwischen GUI und Logik geschoben um den Kommunikationsaufwand der Komponenten nicht zu erhöhen, der Schritt-für-Schritt Modus ist z.B. in der UI somit nicht relevant.

5.4.2 Logic

Aufgeteilt in Logic, Recognition, Solver und Unterstützt durch den Helper sind hier die einzelnen Schritte zum Lösen des Puzzles enthalten.

Es gibt im Modul Logic drei mögliche Arten das Puzzle lösen zu veranlassen. Zuerst wird im Schritt 1 eine grafische Oberfläche entworfen welche die geforderten Eigenschaften aus dem Systemmodel bietet. Darin werden die Teile aus dem Vorprojekt [Jung14] integriert welche die Puzzle Stücke charakterisieren. Zusätzlich wird noch der Algorithmus zur Passprüfung aus dem Agentenvorprojekt [Emri13] integriert. Damit ist eine Serialisierung des Puzzle lösens durch eine 100% igen Simulation möglich geworden. Dies erlaubt einen Schritt-für-Schritt Modus zum schrittweisen Begutachten der Fortschritte des Lösungsvorgangs. Dies ist das oben erwähnte Modul RecognitionSimSer.

Schritt 2 integriert die Agenten aus dem Vorprojekt. In dieser Stufe wird aber nur ein Start und noch kein Schritt-für-Schritt Modus mit echten Agenten möglich sein.

In Schritt 3 wird versucht in dem für die Agentensteuerung verwendeten JADE ein Pausieren der Kommunikation und somit das Anhalten des Lösevorgangs für den Schritt-für-Schritt Modus zu ermöglichen. Eine Alternative dazu wäre das Umprogrammieren der Puzzleteilagenten, so dass diese bei einem Systemagenten vor dem nächsten Teilschritt um Erlaubnis fragen. Falls dies nicht möglich ist kann an eine Aufzeichnung der auflaufenden Log-Meldungen gedacht werden um somit ein Schritt für Schritt Ablauf am Bildschirm darzustellen.

5.4.3 Helper

Hilfskomponenten, welche nichts mit dem Extrahieren der Puzzleteile und dem Extrahieren deren Charakteristika und das Lösen des Puzzles zu tun haben sondern für den Ablauf und die GUI gebraucht werden, aber größeren Umfang haben, sodass sich eine Auslagerung in den Helper angeboten hat.

5.4.4 Recognition

Komponenten, welche mit dem Extrahieren der Puzzleteile und dem Extrahieren deren Charakteristika zu tun haben. Es sind teilweise auch Wrapper zu bestehenden Komponenten.

5.4.5 Solver

Komponenten, welche mit dem eigentlichen Lösen des Puzzles zu tun haben. Es sind teilweise auch Wrapper zu bestehenden Komponenten.

5.4.6 Data

Durch Seter und Geter gekapselte Datenhaltung, es sind teilweise auch Wrapper zu bestehenden Komponenten. Entgegen der klassischen Vorgehensweise in der objektorientierten Programmierung, die Daten und Methoden zu deren Bearbeitung in der gleichen Klasse zu implementieren, werden die Methoden zur Bearbeitung der Daten, welche das Extrahieren der Puzzleteile und das Extrahieren deren Charakteristika und das Lösen des Puzzles betreffend in

der Komponente Logic abgetrennt. Aufgrund der Deklaration als "static" der Data-Klassen können diese dort aber wie in der Klasse befindend, benutzt werden. Diese Vorgehensweise wurde aus den Vorprojekten (vor allem [Emri13]) so übernommen und hat den Vorteil, die Hauptkomponenten des Puzzlelösens hervorzuheben und austauschbarer zu machen. In [Jung14] werden die Methoden und Daten meistens in der gleichen Klasse gehalten. Dies wird versucht so beizubehalten.

In [Jung14] ist vorgesehen, alle Puzzlestücke in ein Foto zu packen. Jetzt sollen die Püzzlestücke auch auf mehrere Fotos verteilt werden können. Dadurch ist in einigen Bereichen eine doppelte Datenhaltung (Vorprojekt – aktuelles Projekt) erforderlich.

5.5 Schnittstellendefinitionen

5.5.1 Schnittstelle „Foto laden“

Die Fotos werden aus dem Dateisystem des Computers geladen.

5.5.2 Schnittstellen zu den Vorprojekten

Die benötigten Module der Vorprojekte werden durch Wrapper gekapselt um die Schnittstellen anzugleichen. Evtl. dadurch verursachte Mehrfachumwandlungen von Daten werden erst in folgenden Versionen der Software vermieden.

5.5.3 Schnittstelle zum Benutzer

Die, bei auf dem Markt üblichen Computern, vorhandene Tastatur, Maus und Monitor-Kombination. Der Monitor sollte für die Darstellung des Puzzles und der Agentenkommunikation höher auflösend sein und dadurch auch den Platz für die Inforation bieten.

6 Spezifikation der Systemkomponenten

6.1 Komponente data

Die Datenhaltung des Programms. Der Zugriff erfolgt über entsprechende Seter- und Geter-Methoden. Aufgrund der Deklaration als „static“ ist der Zugriff aus dem gesamten Programm heraus direkt möglich. Hier sind auch die Klassen welche als Vorlage für die Objekte der Datenelemente dienen untergebracht. Teilweise aus dem Vorprojekt [Emri13] entnommen.

6.2 Klasse FileList

Diese Klasse speichert die Liste der Dateien vom Benutzer gemachten Fotos des Puzzle.

6.2.1 Attribute

Attribut	Typ	Beschreibung
fileList	LinkedList<File>	Die Liste aller Dateien mit Fotos des Puzzle.

6.2.2 Methoden

Methode	Beschreibung
getFileList()	Gibt die Liste der Dateien zurück.
getFileListAsString()	Gibt die Liste der Dateien zurück.
getFile()	Gibt eine bestimmte Datei zurück.
getSize()	Gibt die Anzahl der Dateien zurück.
addFileList()	Fügt Dateien zur Liste hinzu.
emptyList()	Leer die Liste.
printDebug()	Gibt die Liste auf der Konsole aus.

6.3 Klasse Kante

Hier wird eine Kante eines Puzzlestücks definiert.

(Quelle: [Emri13])

6.4 Klasse ListeDerPuzzleTeil

In dieser Klasse wird die Liste der Puzzleteile abgespeichert.

(Quelle: [Emri13])

6.5 Klasse LogGroup

Die möglichen Bezeichnungen der Loglevel werden hier als Konstanten definiert.

6.5.1 Attribute

Attribut	Typ	Beschreibung
logGroup	enum	Die möglichen Bezeichnungen der Loglevel.

6.5.2 Methoden

keine

6.6 Klasse LogItem

Hier wird das Aussehen eines Logeintrages definiert.

6.6.1 Attribute

Attribut	Typ	Beschreibung
text	String	Hier steht der Text des Logeintrags.
group	LogGroup	Hier steht die zugehörige Gruppe des Logeintrags.

6.6.2 Methoden

Methode	Beschreibung
logItem()	Konstruktor der Klasse.
getLogItemText()	Gibt den Text des Logeintrags zurück.
getLogItemGroup()	Gibt die Loggruppe des Logeintrags zurück.

6.7 Klasse LogItemDB

In dieser Klasse wird die Liste der festgelegten Logeintrage gespeichert.

6.7.1 Attribute

Attribut	Typ	Beschreibung
arraySize	int	Hier wird die Größe des Array festgelegt.
logItemList	LogItem[]	Die Liste aller möglichen Logeinträge.

6.7.2 Methoden

Methode	Beschreibung
existItem()	Prüft auf die Existens eines Eintrags in der Liste.
getText()	Gibt den Text eines bestimmten Eintrags zurück.
getGroup()	Gibt die Loggruppe eines bestimmten Eintrags zurück.
inProgram()	Überprüft ob sich ein Logeintrag in der Loggruppe Programm befindet.
inPicture()	Überprüft ob sich ein Logeintrag in der Loggruppe Bildverarbeitung befindet.
inAgent()	Überprüft ob sich ein Logeintrag in der Loggruppe Agent befindet.
printAllDebug()	Gibt die ganze Liste auf der Konsole aus.
printSortDebug()	Gibt die nach Loggruppen sortierte Liste auf der Konsole aus.

6.8 Klasse LogMessage

Hier wird das Aussehen einer Lognachricht definiert.

6.8.1 Attribute

Attribut	Typ	Beschreibung
itemNo	int	Hier steht die Nummer des Logeintrags.
pieceStart	int	Hier steht die Nummer des

		sendenden Agenten.
pieceTarget	int	Hier steht die Nummer des empfangenen Agenten.

6.8.2 Methoden

Methode	Beschreibung
logMessage()	Konstruktor der Klasse.
getItemNo()	Gibt die Nummer des Logeintrags der Lognachricht zurück.
getparamPieceStart()	Gibt die Nummer des sendenden Agentens der Lognachricht zurück.
getparamPieceTarget()	Gibt die Nummer des empfangenen Agentens der Lognachricht zurück.
getLogMessagesAsString()	Gibt alle drei Attribute zurück.

6.9 Klasse LogMessagesList

In dieser Klasse wird die Liste der entstandenen Lognachrichten abgespeichert.

6.9.1 Attribute

Attribut	Typ	Beschreibung
logMessagesList	LinkedList<LogMessage>	Die Liste aller pber die Zeit aufgelaufenen Lognachrichten.

6.9.2 Methoden

Methode	Beschreibung
addLogMessage()	Fügt eine Lognachricht zur Liste hinzu.
getFilteredLogMessagesList()	Gibt die gefilterte Liste zurück.
emptyList()	Leert die Liste.
getAllLogMessages()	Gibt die ganze Liste zurück.
getAllLogMessagesAsString()	Gibt die ganze Liste zurück.

6.10 Klasse ProgramDefaults

Für das Programm hilfreiche Festlegungen werden in diese Klasse abgespeichert.

6.10.1 Attribute

Attribut	Typ	Beschreibung
currentProgramFilterLevel	LogGroup	Legt den aktuellen Level des Programmfilters fest.
currentPictureFilterLevel	LogGroup	Legt den aktuellen Level des Bildverarbeitungsfilters fest.
currentAgentFilterLevel	LogGroup	Legt den aktuellen Level des Agentenfilters fest.
selectedMode	int	Legt die aktuelle Wahl des Modus fdest.
redMark	int	Legt die aktuelle Wahl der einzufärbenden Nachrichten fest.
max	int	Legt die Größe des Ergebnisarray fest.
writeLog	boolean	Legt fest ob eine Datei der Liste der Lognachrichten geschrieben werden soll.

6.10.2 Methoden

Methode	Beschreibung
getImageDir()	Gibt das Verzeichnis der Quellbilder zurück.
getLogFileName()	Gibt den Dateinamen und das Verzeichnis der Logdatei zurück.

6.11 Klasse PuzzleTeil

Hier wird ein Puzzlestück definiert.

(Quelle: [Emri13])

6.12 Klasse ResultArray

In dieser Klasse wird so weit wie gelöst das zusammengesetzte Puzzle abgespeichert.

(Quelle: Sebastian Abele)

6.13 Komponente gui

Diese Komponente beinhaltet die Kommunikation mit dem Benutzer. Er kann hier den Ablauf steuern und bekommt die Ergebnisse angezeigt. Da diese Komponente so zentral ist übernimmt sie auch den Rest der prinzipiellen Ablaufsteuerung, verteilt aber dazu Details an Unterkomponenten in der Komponente Logic.

6.14 Klasse LogArea

Diese Klasse sorgt für das Ausgeben des Logbereichs.

6.14.1 Attribute

Attribut	Typ	Beschreibung
targetScrollPane	JScrollPane	Die JScrollPane des Logbereichs aus dem Hauptfdenster.
targetPanel	JPanel	Die JPanel des Logbereichs aus dem targetScrollPane.
logLabel	JLabel	Die JLabel der „init init init ….“ Zeile.
lastPrintedNo	int	Die Nummer der zuletzt ausgegebenen Lognachricht in LogMessagesList.

6.14.2 Methoden

Methode	Beschreibung
setLogArea()	Damit kann aus dem MainWindow der Logbereich übergeben werden.
refreshInit()	Setzt den Logbereich zurück.
refreshDisplay()	Erneuert den Logbereich um in LogMessageList neu hinzugekommene Zeilen.

6.15 Klasse Main

Enthält „die“ main- Methode zum Anwerfen des Programms.

6.15.1 Attribute

keine

6.15.2 Methoden

Methode	Beschreibung
main()	„die“ main Methode zum Anwerfen des Programms.

6.16 Klasse MainWindow

In dieser Klasse ist das Hauptfenser untergebracht.

6.16.1 Attribute

Attribut	Typ	Beschreibung
serialVersionUID	long	Nötig für swing.
scrollPanePuzzle	JScrollPane	Die JScrollPane im Bildbereich.
buttonJFileChooser	JButton	Dient zum Auswählen der Fotos im Hauptfenster.
buttonStart	JButton	Dient zum Starten im Hauptfenster.
buttonNewFiles	JButton	Dient zum alles Reseten im Hauptfenster.
buttonReset	JButton	Dient zum Reseten der Puzzlestückliste im Hauptfenster.
buttonEnde	JButton	Dient zum Beenden im Hauptfenster.
buttonLogSelection	JButton	Dient zum Starten der Auswahl des Loglevels im Hauptfenster.
buttonSelectMode	JButton	Dient zum Starten der Auswahl des Modus im Hauptfenster.
buttonMark	JButton	Dient zum Starten der

		Auswahl des zu markierenden Agenten im Hauptfenster.
toolBar	JToolBar	Die JToolBar im Hauptfenster.
dialogLogSelection	SelectLog	Fenster zur Auswahl des Loglevels.
dialogSelectMode	SelectMode	Fenster zur Auswahl des Modus.
dialogSelectMark	SelectMark	Fenster zur Auswahl des zu markierenden Agenten.
scrollPaneLog	JScrollPane	Die JScrollPane im Logbereich.
panelLog	JPanel	Das JPanel für die Überschrift im Logbereich.
labelLog	JPanel	Das Jlabel für die Überschrift im Logbereich.
scrollPaneStatus	JScrollPane	Die JScrollPane im Statusbereich.
fileSelected	FlipFlop	Ob Dateien ausgewählt worden sind abspeichern.
runStatus	FlipFlop	Ob Puzzle lösen schon ausgewählt worden ist abspeichern.
pictureProcess	PictureProcessing	Damit wird die Bildverarbeitung gestartet.

6.16.2 Methoden

Methode	Beschreibung
initMainWindow()	Initialisiert das Hauptfenster.
buttonFileChooserActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
buttonNewFilesActionPerformed()	Legt die Aktion für diesen Button fest

	(Button siehe oben).
buttonStartActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
buttonResetActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
buttonEndeActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
buttonLogSelectionActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
buttonSelectModeActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
buttonMarkActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).

6.17 Klasse PictureArea

Diese Klasse sorgt für das Ausgeben des Bildbereichs.

6.17.1 Attribute

Attribut	Typ	Beschreibung
targetScrollPane	JScrollPane	Die JscrollPane des Bildbereichs aus dem Hauptfdenster.
targetPanel	JPanel	Das JPanel des Bildbereichs aus dem targetScrollPane.
panelFoto	JPanel	Das JPanel für die Fotos.
panelIconFoto	JPanel	Das Jpanel für die eigentlichen Fotos im Bildbereich.
labelFoto	JLabel	Das JLabel für die Überschrift über die Fotos.
panelImage	JPanel	Das JPanel für die Images.
panelIconImage	JPanel	Das JPanel für die eigentlichen Images im

		Bildbereich.
labelImage	JLabel	Das JLabel für die Überschrift über die Images.
panelBuild	JPanel	Das JPanel für das zusammengesetzte Puzzle.
panelIconBuild	JPanel	Das Jpanel für die eigentlichen Images im Bildbereich.
labelBuild	JLabel	Das Jlabel für die Überschrift über das zusammengesetzte Puzzle.
outputDimensionX	int	Anzahl Spalten im bereits zusammengesetzten Puzzle.
outputDimensionY	int	Anzahl Zeilen im bereits zusammengesetzten Puzzle.
outputArray	JLabel	Die JLabel des bereits zusammengesetzten Puzzles.
alreadyDisplayed	boolean	Information welches Puzzlestück bereits angezeigt wurde.
wasIncreased	boolean	Information ob Ausgabebereich des bereits zusammengesetzten Puzzles seit dem letzten Neuzeichnen vergrößert worden ist.

6.17.2 Methoden

Methode	Beschreibung
setPictureArea()	Damit kann aus dem MainWindow der Bildbereich übergeben werden.
displayInit()	Initialisiert den Bildbereich.
displayChangeFoto()	Erneuert den Bereich der Fotos.
displayImageReady()	Erneuert den Beriech der Images.

displayPuzzleBuildFirst()	Zeigt den Bereich des zusammengesetzten Puzzles im Fall erstes Stück anzeigen an.
displayPuzzleBuildMorePiece()	Erneuert den Bereich des zusammengesetzten Puzzles.
putOutputArrayInPanelIconBuild()	Baut den Outputarray in das Jpanel des Ausgabebereichs und beachtet dabei outputDimensionX und outputDimensionY.
fillOutputArray()	Fügt neu gefundene Puzzlestücke in den Outputarray an der richtigen Stelle hinzu.
fillJLabel()	Erzeugt ein neues Jlabel mit einem neu gefundenem Puzzlestück für den Outputarray.
increaseOutputArray()	Vergrößert den Outputarray durch einen größeren neu anlegen und alten hinein kopieren.

6.18 Klasse SelectLog

In dieser Klasse wird das Auswahlfenster zum Festlegen der anzuzeigenden Nachrichten festgelegt.

6.18.1 Attribute

Attribut	Typ	Beschreibung
serialVersionUID	long	Nötig für swing.
radioButtonProgramImportant	JRadioButton	JRadioButton für ProgramImportant.
radioButtonProgramBasic	JRadioButton	JRadioButton für ProgramBasic.
radioButtonProgramInfo	JRadioButton	JRadioButton für ProgramInfo.
radioButtonPictureImportant	JRadioButton	JRadioButton für PictureImportant.
radioButtonPictureBasic	JRadioButton	JRadioButton für PictureBasic.
radioButtonPictureInfo	JRadioButton	JRadioButton für PictureInfo.

radioButtonAgentImportant	JRadioButton	JRadioButton für AgentImportant.
radioButtonAgentBasic	JRadioButton	JRadioButton für AgentBasic.
radioButtonAgentInfo	JRadioButton	JRadioButton für AgentInfo.
buttonOK	JButton	JButton für OK.
buttonAbort	JButton	JButton für Abbruch.
groupProgram	ButtonGroup	ButtonGroup für Programmgruppe.
groupPicture	ButtonGroup	ButtonGroup für Bilderkennungsgruppe.
groupAgent	ButtonGroup	ButtonGroup für Agentengruppe.
radioButtonPanel	JPanel	JPanel zum Aufnehmen der JradioButtons.

6.18.2 Methoden

Methode	Beschreibung
selectLog()	Konstruktor der Klasse.
init()	Erzeugt das Fenster.
blank()	Dummy damit Eclipse kein unbenutzte Deklaration anzeigt.
buttonAbortActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
buttonOKActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).

6.19 Klasse SelectMark

In dieser Klasse wird das Auswahlfenster zum Festlegen der einzufärbenden Nachrichten festgelegt.

6.19.1 Attribute

Attribut	Typ	Beschreibung
serialVersionUID	long	Nötig für swing.
buttonOK	JButton	JButton für OK.
buttonAbort	JButton	JButton für Abbruch.
labelRed	JLabel	JLabel für den Text zum Eingabefeld.
inputRed	JTextField	Das Eingabefeld zum Eingeben welcher Agent markiert werden soll.
labelText1	JLabel	Beschreibender Text im Fenster.
labelText2	JLabel	Noch ein beschreibender Text im Fenster.
buttonPanel	JPanel	Das JPanel zum Zusammenfassen der Jbuttons.
inputPanel	JPanel	Das JPanel zum Aufnehmen der Eingabeelemente.

6.19.2 Methoden

Methode	Beschreibung
selectMark()	Konstruktor der Klasse.
init()	Erzeugt das Fenster.
blank()	Dummy damit Eclipse kein unbenutzte Deklaration anzeigt.
buttonAbortActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
buttonOKActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).

6.20 Klasse SelectMode

In dieser Klasse wird das Auswahlfenster zum Festlegen des ausgewählten Modus festgelegt.

6.20.1 Attribute

Attribut	Typ	Beschreibung
serialVersionUID	long	Nötig für swing.
radioButtonSimSer	JRadioButton	JButton für die Auswahl des simulierten Lösens.
radioButtonAgenten	JRadioButton	JButton für die Auswahl des Lösens über Agenten.
radioButtonPanel	JPanel	JPanel zum Aufnehmen der JRadioButtons.
groupRadioButton	ButtonGroup	ButtonGroup für Auswahlgruppe.

6.20.2 Methoden

Methode	Beschreibung
selectMode()	Konstruktor der Klasse.
init()	Erzeugt das Fenster.
blank()	Dummy damit Eclipse kein unbenutzte Deklaration anzeigt.
radioButtonSimSerActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).
radioButtonAgentenActionPerformed()	Legt die Aktion für diesen Button fest (Button siehe oben).

6.21 Klasse StatusArea

Diese Klasse sorgt für das Ausgeben des Statusbereichs.

6.21.1 Attribute

Attribut	Typ	Beschreibung
targetScrollPane	JScrollPane	Die JScrollPane des Statusbereichs aus dem

		Hauptfdenster.
targetPanel	JPanel	Das JPanel des Bildbereichs aus dem targetScrollPane.
labelInit	JLabel	Das JLabel zum Anzeigen der Init- Information.
labelStart	JLabel	Das JLabel zum Anzeigen der Start- Information.
labelRecognition	JLabel	Das JLabel zum Anzeigen der Bilderkennunsphase.
labelSolve	JLabel	Das JLabel zum Anzeigen der Information Lösephase ist hat begonnen.
labelLocalFinish	JLabel	Das JLabel zum Anzeigen, dass die lokalen Lösungen vorliegen.
labelFalse	JLabel	Das JLabel zum Anzeigen, dass keine Lösung möglich ist.
labelFinish	JLabel	Das JLabel zum Anzeigen der Puzzle ist gelöst Information.
labelUnexpected	JLabel	Das JLabel zum Anzeigen, dass es Widersprüche in der Lösung gibt.

6.21.2 Methoden

Methode	Beschreibung
setStatusArea()	Damit kann aus dem MainWindow der Statusbereich übergeben werden.
displayInit()	Initialisiert den Statusbereich.
displayStart()	Zeigt das entsprechende JLabel an (Inhalt des JLabel siehe oben).
displayRecognition()	Zeigt das entsprechende JLabel an (Inhalt des

	JLabel siehe oben).
displaySolve()	Zeigt das entsprechende JLabel an (Inhalt des JLabel siehe oben).
displayLocalFinish()	Zeigt das entsprechende JLabel an (Inhalt des JLabel siehe oben).
displayFalse()	Zeigt das entsprechende JLabel an (Inhalt des JLabel siehe oben).
displayFinish()	Zeigt das entsprechende JLabel an (Inhalt des JLabel siehe oben).
displayUnexpected()	Zeigt das entsprechende JLabel an (Inhalt des JLabel siehe oben).

6.22 Komponente helper

Hilfskomponenten, welche nichts mit dem Extrahieren der Puzzleteile, dem Extrahieren deren Charakteristika oder dem Lösen und Anzeigen des Puzzles zu tun haben sondern für den Ablauf und die GUI gebraucht werden, aber größeren Umfang haben, sodass sich ein Auslagerung in eine eigene Komponente helper angeboten hat.

6.23 Klasse FlipFlop

Diese Klasse stellt die Funktionalität eines Flipflops bereicht. Damit kann ein Status festgehalten und dieser mit den Methoden der Klasse einfach gewechselt werden.

6.23.1 Attribute

Attribut	Typ	Beschreibung
status	boolean	Beinhaltet den Status des Flipflips.

6.23.2 Methoden

Methode	Beschreibung
flipFlop()	Konstruktor der Klasse.
getStatus()	Gibt aktuellen Status zurück.
setStatus()	Setzt den Status neu.
changeStatus()	Wechselt den Status.

6.24 Klasse FotoLoader

Mit dieser Klasse kann ein Bild vom Datenträger geladen werden.

6.24.1 Attribute

keine

6.24.2 Methoden

Methode	Beschreibung
loadPuzzleFoto()	Gibt das geladene Foto zurück.

6.25 Klasse Substitute

Sorgt für das Umsetzen der Lognachrichten in Text und das Substituieren der Platzhalter durch die richtigen Zahlen bzw. Zeichenfolgen.

6.25.1 Attribute

Attribut	Typ	Beschreibung
makeRed	boolean	Information, ob die Zeile eingefärbt werden soll.

6.25.2 Methoden

Methode	Beschreibung
makeText()	Erzeugt die Zeile mit den richtigen Ersetzungen.
isItRed()	Gibt Information, ob diese Zeile gefärbt werden soll, zurück.

6.26 Klasse WriteFile

Schreibt die Liste der Lognachrichten in eine Datei.

6.26.1 Attribute

keine

6.26.2 Methoden

Methode	Beschreibung
writeOut()	Schreibt den Inhalt von LogMessagesList in ein Datei.

6.27 Komponente libs

Externe Programmbibliotheken, zur Zeit nur das Agentenframework jade.jar.

6.28 Komponente logic

Diese Komponente beinhaltet Teile der Steuerung des Programmablaufs, welche nichts direkt mit der Bilderkennung oder dem Lösen des Puzzles zu tun haben, aber groß genug sind um nicht in der Komponente gui gelassen werden zu können.

6.29 Klasse EmitMessage

Diese Klasse bietet die Möglichkeit von beliebiger Stelle im Programm eine neue Lognachricht in die Liste einzutragen.

6.29.1 Attribute

Attribut	Typ	Beschreibung
resultArray	int[][]	Der bereits gelöste Puzzlebereich.
outFile	WriteFile	Zum Schreiben in eine Datei.

6.29.2 Methoden

Methode	Beschreibung
emitLogMessage()	Wird aufgerufen um eine Lognachricht in die Liste hinzuzufügen.
changeStatusMessage()	Werten die neu hinzugekommene Nachricht aus und benachrichtigt ggf. den Statusbereich.
changePictureArea()	Werten die neu hinzugekommene Nachricht aus und benachrichtigt ggf. den Bildbereich.

6.30 Klasse PictureProcessing

Mit dieser Klasse kann die Bildverarbeitung durchgeführt werden.

6.30.1 Attribute

keine

6.30.2 Methoden

Methode	Beschreibung

makeRecognition()	Führt die Bildverarbeitung durch.

6.31 Komponente recognition

Komponenten der Bildverarbeitung zur Extrahierung der Puzzleteile aus den Fotos und dem Extrahieren der Charakteristika dieser. Aus dem Vorprojekt [Jung14] entnommen.

6.32 Klasse CharacteristicsRecognition

In dieser Klasse ist das Extrahiert aus einem Image mit genau einem Puzzlestück darin zu der Characteristika dieses Stücks umgesetzt (Quelle: [Jung14]).

6.33 Klasse ImageRecognition

In dieser Klasse ist das Aufteilen des Quellbildes in einzelne Puzzlestücke umgesetzt (Quelle: [Jung14]).

6.34 Komponente solver

Komponenten zum eigentlichen Lösen des Puzzles. Zur Zeit entweder durch ein von Sebastian Abele bereit gestelltes Agentensystem, oder nur für den prinzipiellen Ablauf simulierendes Scheinlösen ohne Agenten.

6.35 Klasse Agenten

Diese Klasse kann das Agentensystem starten.

6.35.1 Attribute

Attribut	Typ	Beschreibung
agentControl	AgentController	Die Agenten zum Lösen des Puzzles.
container	AgentContainer	Der Container für die Agenten.

6.35.2 Methoden

Methode	Beschreibung
makeAgenten()	Erstellt die Agenten.
clearAgenten()	Löscht die Agenten.

6.36 Klasse PuzzleTeilAgent

In dieser Klasse ist ein Puzzleteil- Agent untergebracht.

(Quelle: Sebastian Abele)

6.37 Klasse SimSer

Mit dieser Klasse ist die Simulation des Programmablauf ohne Agenten möglich.

6.37.1 Attribute

keine

6.37.2 Methoden

Methode	Beschreibung
makeSimSer()	Löst die Bildverarbeitung aus. Simuliert den Programmablauf durch das Aussenden der Emitnachrichten und das Basteln eines fertigen Puzzles.

6.38 Klasse SystemAgent

In dieser Klasse in der System- Agent untergebracht (Quelle: Sebastian Abele).

6.39 Komponente tests

Hier sind Programme zum Testen von Teilaufgaben oder Komponenten der eigentlichen Software.

7 Komplexitätsanalyse

Der Gesamtvorgang unterteilt sich in die Abschnitte Bildanalyse und Lösevorgang.

7.1 Bildanalyse

Vor dem eigentlichen Lösevorgang steht die Bildanalyse. Es müssen alle Puzzlestücke in den Ausgangsfotos gefunden und danach jedes einzelne Puzzlestück noch für sich analysiert werden. Da dazu jedes Puzzlestück zwei mal angefasst wird ergibt sich ein Aufwand von O(n).

Bei den im Test verwendeten Puzzle handelt es sich um kleinere Puzzle mit ca 9 bis 12 Puzzlestücken. Diese konnten in einer Fotoaufnahme zusammen gefasst werden. Die vorliegende Software beherrscht schon das Aufteilen der Puzzlestücke auf mehrere Fotoaufnahmen. Damit bietet sich eine parallele Analyse der Ausgangsfotos um die Laufzeit des Suchens der Puzzlestücke in den Fotos zu beschleunigen an. Die Analyse der entstandenen Einzelbereiche der Ausgangsfotos, welche jeweils genau ein Puzzlestücke enthalten, könnte man zum Extrahieren der Charakteristika gleich an die Puzzleteilagenten verteilen um so die schon bestehende Nebenläufigkeit des Agentensystems auszunutzen und damit ebenso eine Beschleunigung erzielen. Trotz all dieser Bemühungen bleibt die Komplexität der Bildanalsye bei O(n), man muss eben jedes Puzzlestück einmal „anfassen“. Für den Zuschauer würde allerdings ein Vierteln der Analysezeit mit aktuellen Quadcore Prozessoren eine spürbare Verkürzung der Wartezeit bedeuten, in der für den Zuschauer eigentlich nichts passiert. Der Aufwand der Parallelisierung würde sich also lohnen.

7.2 Lösevorgang

Im Prinzip muss zum Lösen eines Puzzles jede Kante eines Puzzlestücks mit jeder Kante aller anderen Püzzlestücke verglichen werden. Also ohne Sonderfälle zu berücksichtigen jedes Puzzlestück mit jedem anderen Puzzlestück multipliziert mit vier Kanten. Aufgrund der mechanischen Randbedingung eines starren Puzzlestücks erübrigt sich die Überprüfung mit den eigenen Kanten. Dies kann bei der Aufwandsanalyse noch berücksichtigt werden. Bei einem flachen Puzzle, welches eben auf einem Tisch zusammen gesetzt werden kann, hat nicht jedes Puzzlestück vier Kontaktkanten. Wie viele Außenkanten bei einem realen Puzzle vorhanden sind hängt allerdings von dessen Geometrie ab. Ein Beachten oder nicht Beachten der Außenkanten hat keine Änderung der Anzahl der Puzzlestücke zur Folge und somit keinen Einfluss auf die Komplexitätsklasse und bleibt deshalb unberücksichtigt. Es wird also vereinfacht die Anzahl der Vergleiche Puzzlestück zu Puzzlestück ermittelt und dieser Wert mit vier multipliziert. In Abbildung 7.1: Komplexität des Lösevorgangs

sind die nötigen Vergleiche bildlich angedeutet. Damit lässt sich das Ergebnis $O(n^2)$ schon erahnen.

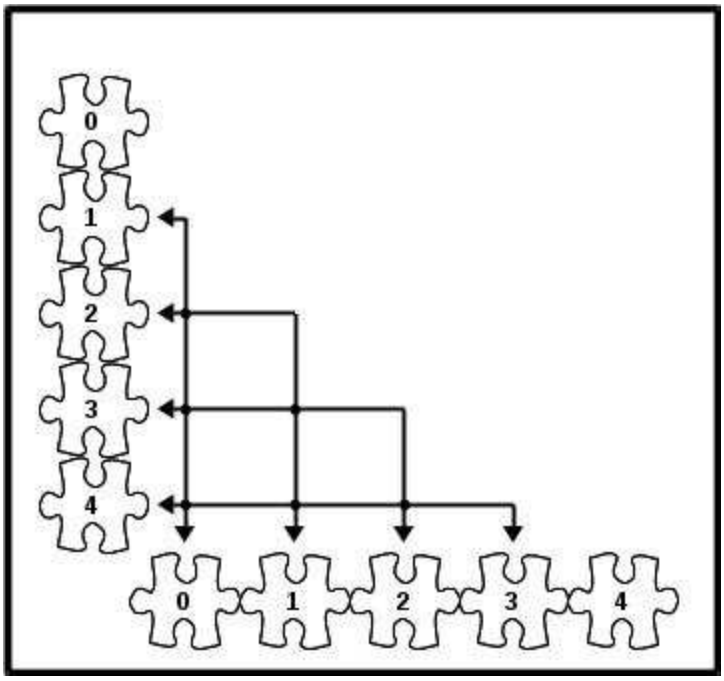

Abbildung 7.1: Komplexität des Lösevorgangs

Ein Vergleich jedes mit jedem, wobei der Vergleich eines Puzzlestücks mit demselben Puzzlestück ungezählt bleibt (Reflexivität) und ein Vergleich dieses Puzzlestücks mit einem Anderen das gleiche Ergebnis wie ein Vergleich des Anderen mit diesem Puzzlestück ergibt (Symmetrie) und somit nur ein mal ermittelt werden muss, ergibt bei N Puzzlestücken folgende Aufsummierung: Das Erste mit allen Restlichen ergibt N-1 Vergleiche, das Zweite mit den nun übrig gebliebenen Restlichen ergibt N-2, und mit diesem Muster weiter bis der übrig gebliebene Rest auf Nichts schrumpft, also das Vorletzte mit dem Letzten N – (N - 1) und das Letzte mit keinem mehr ergibt N – N. Zusammengefasst ist dies N/2 * (N – 1), der n-1-ten Dreieckszahl, was sich auch aus der Kombinatorik als Ergebnis des Binomialkoeffizienten N über 2 ergibt.

Dies liegt in $O(n^2)$. Die Art der Ermittlung dieser Vergleiche erinnert sehr Stark an einfache Sortierverfahren. Dort wie hier lässt sich durch keinen einfachen Trick die Komplexitätsklasse zur nächst besseren wechseln.

Das Agentensystem arbeitet schon Nebenläufigkeit, eine Veränderung um einen konstanten Faktor ist aber durch die quadratische Laufzeit ab einer bestimmten Anzahl Puzzlestücke nicht mehr spürbar.

7.3 Gesamtvorgang

Da der Lösevorgang nur mit kompletter und erfolgreicher Bildanalyse starten kann addieren sich die beiden Laufzeiten $O(n) + O(n^2)$. Wegen $O(n) \in O(n^2)$ ergibt sich damit $O(n^2)$ für den Gesamtvorgang, der bestimmende Teil ist somit der Lösevorgang.

8 Prüfung der Software

8.1 Prüfanforderungen

Die Aufgabe dieser Studienarbeit ist das Verbinden der bestehenden Komponenten Bildverarbeitung und Agentensystem zu einem benutzbaren Ganzen und die Darstellung des Ablaufs und des Ergebnisses. Dies soll durch eine grafische Benutzeroberfläche geschehen. Geprüft wird die Bedienung der Oberfläche und das Anzeigen des Ergebnisses. Daher die Bildverarbeitung und das Agentensystem nicht Inhalt dieser Studienarbeit ist, werden sie als solche auch nicht geprüft und deshalb Testpuzzle benützt, die bei der Prüfung dieser Komponenten schon erfolgreich eingesetzt worden sind. Dadurch, dass sich das Puzzle lösende Agentensystem noch im Prototypenstatus befindet kann nicht sicher erkannt werden ob das Lösen des Puzzle erfolgreich war. Auch muss davon ausgegangen werden , dass alle Puzzleteile immer richtig erkannt worden sind.

8.2 Methoden der Prüfung

Die in den Prüffällen aufgeführten möglichen Durchläufe und deren Sollverhalten werden durch Funktionsabdeckung überprüft. Dabei wird folgendes notiert:

- Testfall

- Datum

- Version der getesteten Software

- Sollverhalten erreicht

- Abweichungen (falls vorhanden)

Die Prüffälle sollen nicht nur den Normalfall beinhalten, sondern auch mögliche Fehlbedienungen und Fehlbedingungen beachten. Dazu sind vom gleichen Grundvorgang (wie Programm lässt sich beenden) auch die Variationen (gleich nach Start, nach Auswahl der Fotos aber ohne Lösevorgang, ...) zu bestimmen. Es werden in solchen Fällen allerdings nicht alle aus der Kombinatorik dadurch entstehenden Prüffälle einzeln aufgelistet sondern nur die Besonderheit, welche intuitive um restliche Abläufe ergänzt wird.

8.3 Prüfkriterien

Falls das beobachtete Verhalten dem Sollverhalten entspricht ist die Prüfung bestanden.

9 Benutzungsanleitung

9.1 Installation

9.1.1 Systemvoraussetzungen

Hardware: Ein handelsüblicher PC mit mind. 4GB Hauptspeicher und großem Monitor.

Betriebssoftware: Das Programm arbeitet auf der Java PC Plattform der Version 1.7.

9.1.2 Installation

Eine Installation ist Aufgrund der Verwendung der Java Plattform nicht nötig. Es wird nur die anschließend aufgelistete Datei benötigt, in deren Verzeichnis Lese-, Schreib- und je nach verwendetem Betriebssystem auch Ausführungsrechte benötigt werden. In diesem Verzeichnis wird das Agentenframwork Jade Dateien anlegen. Auch wird empfohlen in diesem Verzeichnis das Unterverzeichnis „PuzzleBilder“ für die Aufnahme der Puzzlefotos anzulegen, da dort der Standardpfad dafür ist.

9.1.3 Benötigte Dateien

- puzzler.jar

9.2 Inbetriebnahme

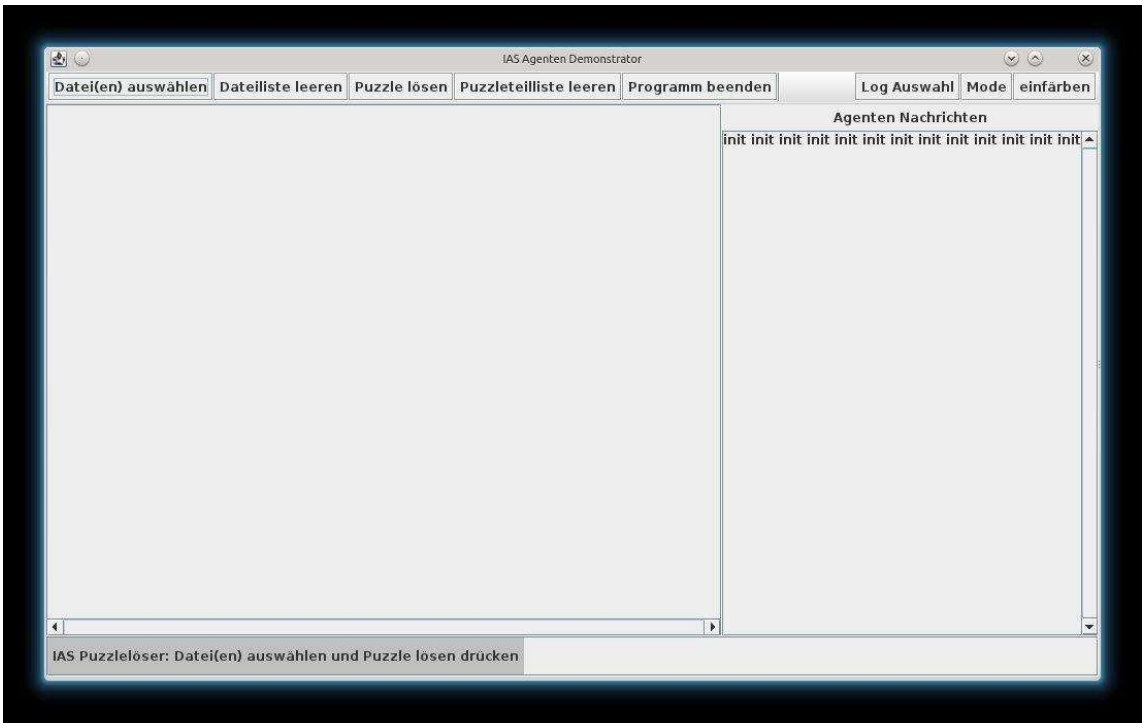

Abbildung 9.1: Startbildschrim

Das Programm wird in der Java Plattform mit

java -jar puzzler.jar

gestartet. Sollte der zugeteilte Speicherplatz bei anderen als den getesteten Puzzlefotos nicht ausreichen, kann dieser mit folgenden Parametern

-Xmsn: initiale Größe des allokierten Speichers.

-Xmxn: maximale Größe des allokierten Speichers.

java -Xms256M -Xmx2G -jar puzzler.jar (Zahlenwerte als Beispiel)

herauf gesetzt werden.

9.3 Übersicht

Abbildung 9.2: Teilbereiche farblich markiert

9.3.1 Bedienleiste

Über die Bedienelemente in der Bedienleiste werden die Funktionen des Programms gesteuert.

9.3.2 Bildbereich

Die ausgewählten Puzzlefotos, die darin erkannten Puzzlestücke und das zusammen gesetzte und somit gelöste Puzzle wird hier bildlich angezeigt.

9.3.3 Logbereich

Die Programmmeldungen und die Agentenkommunikation während aller Phasen des Programmablaufs werden hier, teilweise gefiltert, angezeigt.

9.3.4 Statusbereich

Hier wird angezeigt in welchem logischen Abschnitt der Programmausführung sich das Programm befindet.

9.4 Funktionen

Die Bedienung des Programms erfolgt ausschließlich über die Bedienleiste. Die verschiedenen Bedienelemente und deren Funktion werden hier vorgestellt.

9.4.1 Datei(en) auswählen

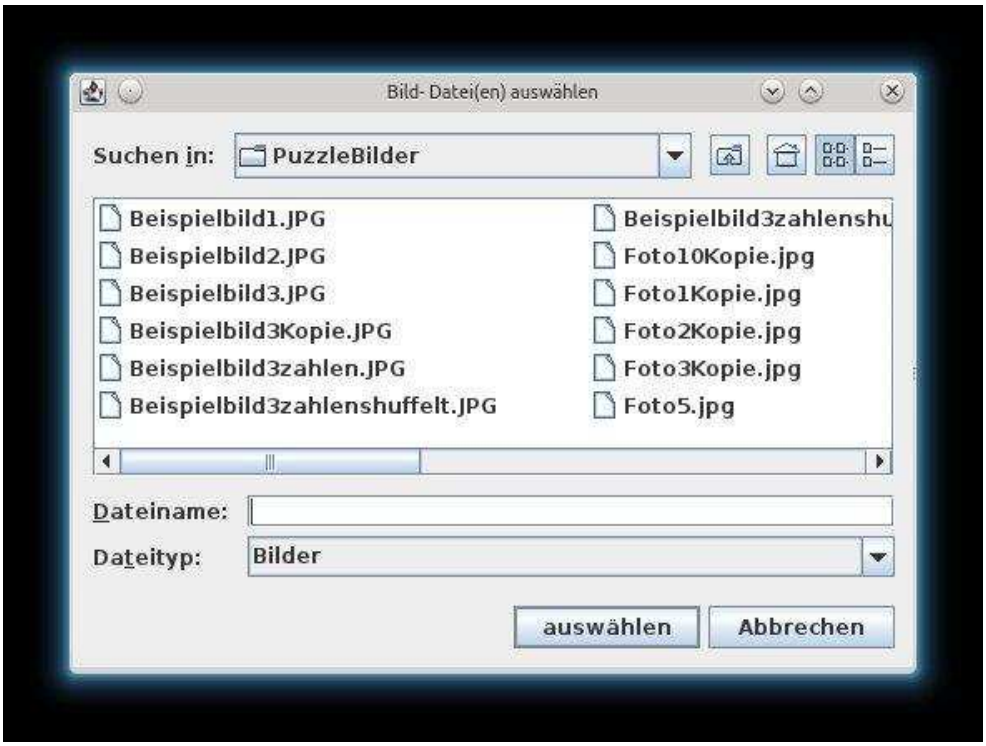

Abbildung 9.3: Dialog Fotoauswahl

Mit dem Bedienelement „Datei(en) auswählen“ wird der Dialog zur Auswahl der im Dateisystem vorhandenen Dateien mit Fotos zu lösender Puzzle aufgerufen. Es können mehrere Dateien ausgewählt werden. Auch kann der Dialog mehrfach aufgerufen werden um weitere Dateien in die programminterne Liste hinzuzufügen, dabei wird die Auswahl vorheriger Aufrufe nicht angezeigt. Die Möglichkeit der Dateiauswahl wird durch das Starten des Lösevorgangs begrenzt. Es werden nur Dateien mit der Endung „jpg“ angezeigt.

9.4.2 Dateiliste leeren

Die über das Bedienelement „Datei(en) auswählen“ entstandene Liste der Auswahl an Dateien von Fotos kann hierüber geleert werden. Dabei wird auch das Agentensystem zurückgesetzt und die Liste der erkannten und charakterisierten Puzzlestücke geleert. Es ist nach einem Lösevorgang möglich wieder Dateien von Fotos auszuwählen. Diese Funktion wird benötigt um ein anderes Puzzle zu lösen ohne das Programm beenden und neu starten zu müssen, oder nach falsch ausgewählten Fotos die Liste einfach nur zu leeren.

9.4.3 Puzzle lösen

Der Lösevorgang auf der ausgewählten Dateiliste wird gestartet. Im Statusbereich wird angezeigt in welchem logischen Abschnitt sich das Programm befindet, die einzelnen Programmmeldungen und die Agentenkommunikation können im Logbereich mitverfolgt werden.

9.4.4 Puzzleteilliste leeren

Dieses Bedienelement ist eine abgeschwächte Version des Bedienelements „Dateiliste leeren“. Es wird nur die Liste der erkannten und charakterisierten Puzzlestücke geleert und das Agentensystem zurückgesetzt, aber nicht die Dateiliste geleert. Es ist damit möglich das gleiche Puzzle erneut zu lösen, also nach dem Ende des Lösevorgangs den nächsten Lösevorgang auf der gleichen, unveränderten Dateiauswahl noch einmal zu starten. Die aufgelaufenen Meldungen im Logbereich werden für Vergleiche unter den verschiedenen Läufen nicht gelöscht sondern nur durch eine „init init init ...“ Zeile getrennt.

9.4.5 Programm beenden

Hiermit wird das Programm beendet, alle, außer den abgespeicherten Informationen (siehe 9.5.2.5 Log ausschreiben), gehen verloren.

9.4.6 Log Auswahl

Abbildung 9.4: Dialog Logauswahl

Die Stufen der Filterung einzelner Meldungen sind nach Bereichen getrennt. Es gibt die Bereiche: Programm: hier sind Meldungen des Gesamtprogramms, Bild: Meldungen die Bildverarbeitung betreffend, und Agent: beinhaltet die Agentenkommunikation. Die Stufe der Filterung lässt sich pro Bereich getrennt regeln. Es gibt die Stufen: wichtig, normal und info, welche in dieser Reihenfolge zunehmend mehr Meldungen im entsprechenden Bereich durchlassen.

9.4.7 Mode

Um verschiedene Lösungsverfahren und Strategien wechseln zu können.

9.4.8 Einfärben

Abbildung 9.5: Dialog Agent markieren

Es ist möglich einen Agenten farblich in den Meldungen im Logbereich hervorzuheben. Dies wird hier eingestellt. Es soll damit eine bessere Nachverfolgbarkeit einzelner Agenten ermöglicht werden. Mögliche Nummern sind die Nummern der Agenten (0 bis Anzahl minus 1), für Meldungen des Systemagenten Neo ist die Nummer -1 und für Meldungen des Programms die Nummer -2 vorgesehen. Daher vor der Bildverarbeitung die Anzahl der erkannten Puzzlestücke nicht bekannt ist, wird auf die Plausibilitätsprüfung der Eingabe verzichtet. Falls ein nicht existierender Agent ausgewählt wird, wird nichts farblich hervorgehoben.

9.5 Durchführung spezieller Operationen

9.5.1 Ein Puzzle lösen

Vor oder nach der Auswahl der Fotos (siehe 9.4.1 Datei(en) auswählen) sollten gewünschte Änderungen im Filter (siehe 9.4.6 Log Auswahl) und der Markierung (siehe 9.4.8 Einfärben) vorgenommen werden. Die ausgewählten Fotos werden im Bildbereich angezeigt (Abbildung 9.6).

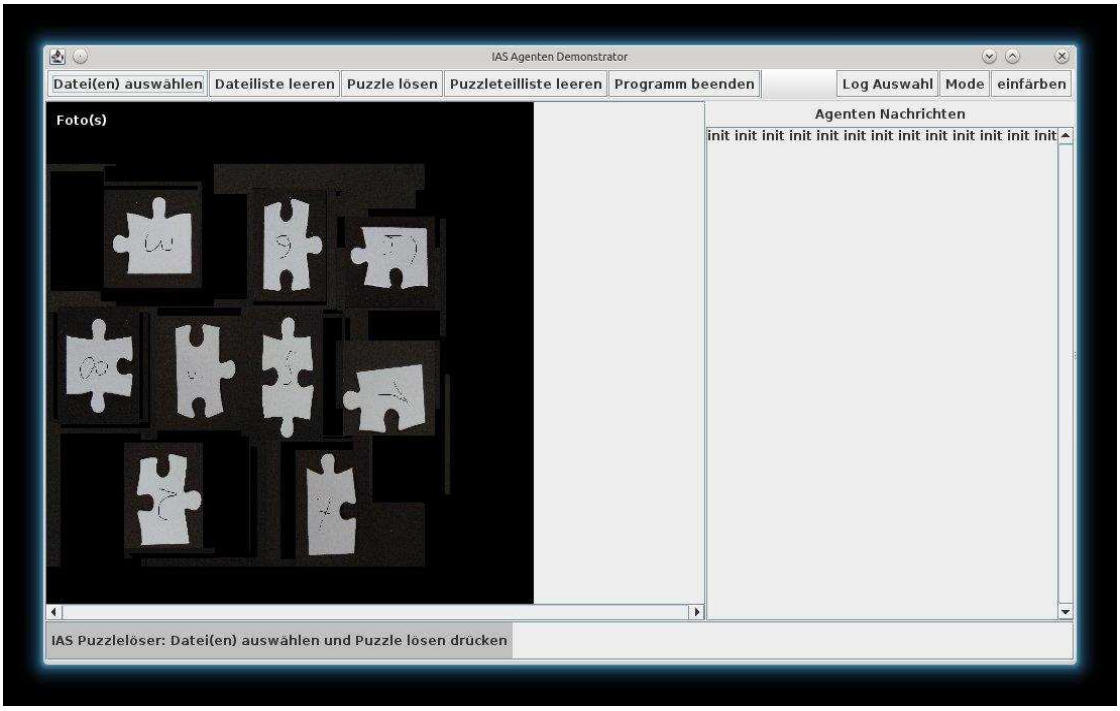

Abbildung 9.6: Ansicht nach Dateiauswahl

Nach dem Start des Lösevorgangs (siehe 9.4.3 Puzzle lösen) wird zuerst die Bilderkennung durchlaufen.

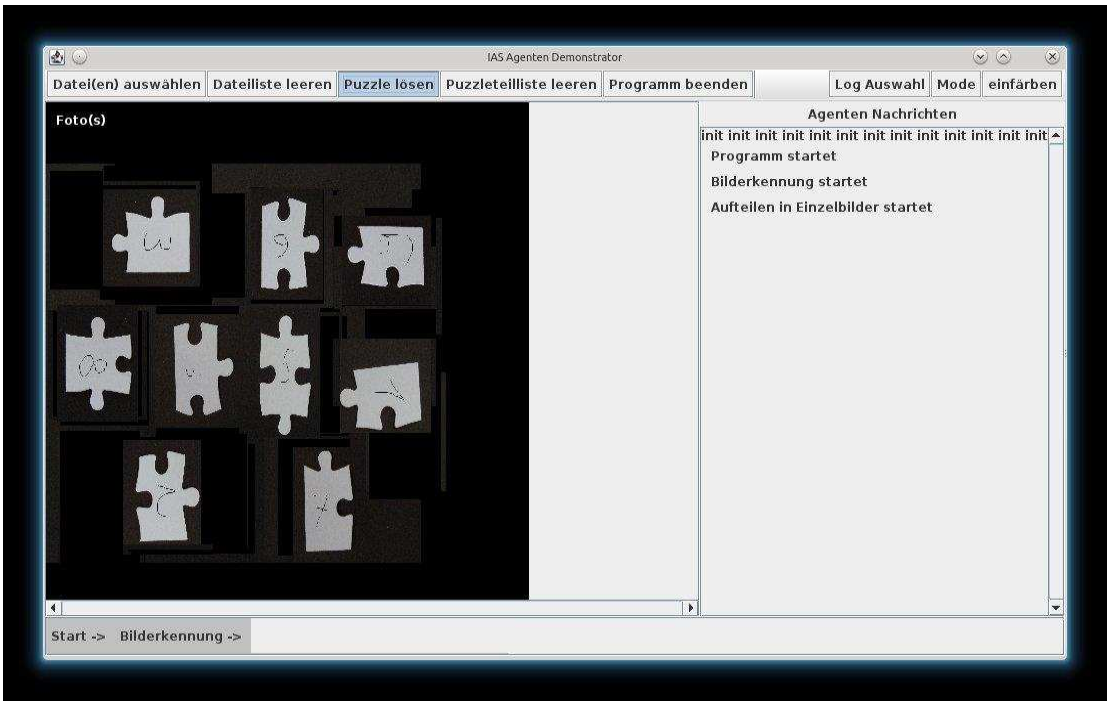

Abbildung 9.7: Bilderkennung beginnt

Die Bilderkennung muss als erstes die Puzzlestücke in den Fotos erkennen. Dies kann sehr lange dauern. Alle erkannten Puzzlestücke werden nummeriert im Bildbereich angezeigt und die Gesamtzahl im Logbereich ausgegeben (Abbildung 9.8).

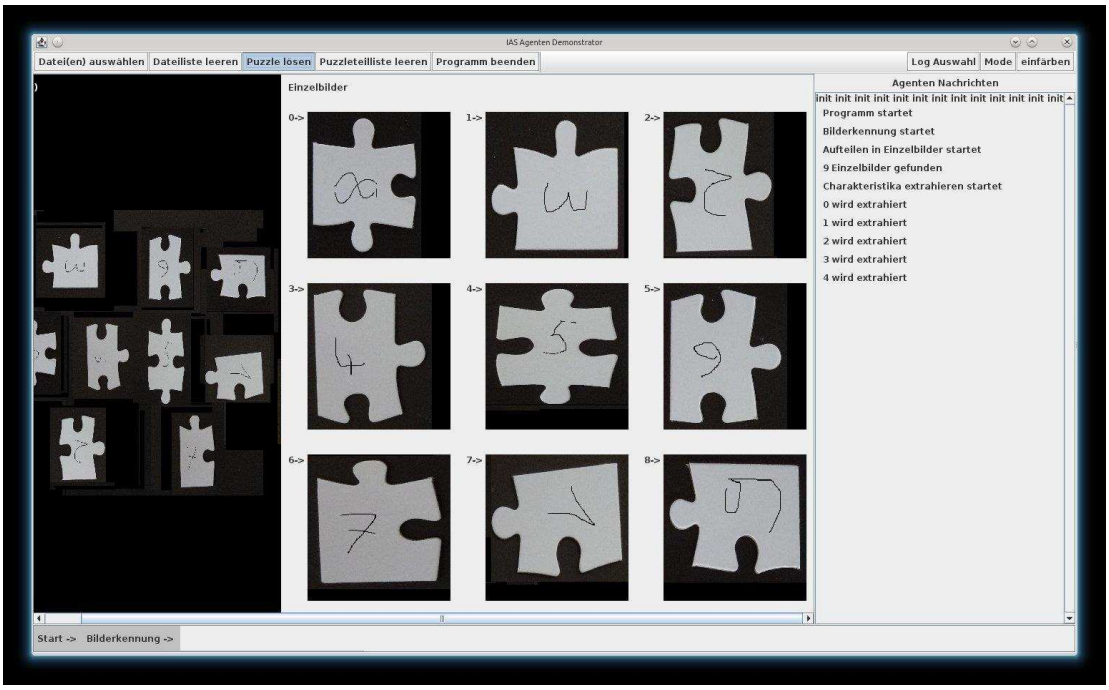

Abbildung 9.8: Anzeige gefundener Puzzleteile und erste Extraktion der Charakteristika

Danach erfolgt die Extraktion der Charakteristika der einzelnen Puzzlestücke (Abbildung 8). Auch dieser Vorgang kann sehr lange dauern. Im Anschluss wird das Puzzle gelöst. Das Ergebnis wird angezeigt (Abbildung 9.9).

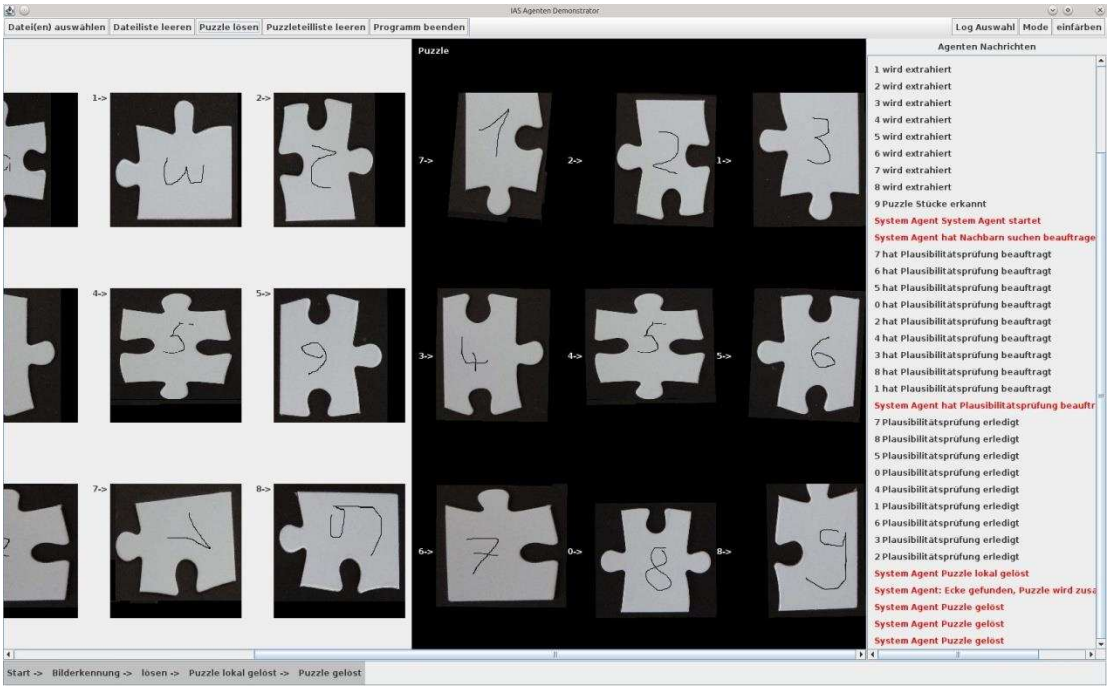

Abbildung 9.9: gelöstes Puzzle

Sollte sich das Puzzle nicht lösen lassen wird dies im Statusbereich angezeigt und die genaueren Umstände im Logbereich ausgegeben.

Jetzt kann das gleiche Puzzle noch einmal gelöst werden (siehe 9.4.4 Puzzleteilliste leeren), oder ein anderes (siehe 9.4.2 Dateiliste leeren), oder das Programm beendet werden (siehe 9.4.5 Programm beenden).

9.5.2 Spezielle Operationen

In der Klasse ProgrammDefaults.java können Standardwerte für das Programm geändert werden.

```
public class ProgramDefaults
{
    // Default für Anfangsbelegung der Filter
    public static LogGroup currentProgramFilterLevel = LogGroup.PROGRAM_IMPORTANT;
    public static LogGroup currentPictureFilterLevel = LogGroup.PICTURE_INFO;
    public static LogGroup currentAgentFilterLevel = LogGroup.AGENT_IMPORTANT;

    // Default für Anfangsauswahl welcher ProgrammCode zum Lösen benutzt wird
    // 0: SimSer
    // 1: Agenten
    public static int selectedMode = 1;

    // Default für Anfangsauswahl welcher Agent eingefärbt werden soll
    // -2: Programm
    // -1: SystemAgent
    // 0..n: PuzzleTeilAgent
    public static int redMark = -1;

    // Größe ResultArray (Spalten, Zeilen)
    public static int max = 100;

    // ob überhaupt eine LogDatei auf Festplatte geschrieben wird
    public static boolean writeLog = true;
    // Ort der LogDatei
    public static String getLogFileName()
    {
        String name = System.getProperty("user.dir") + System.getProperty("file.separator") + "PuzzleBilder" + System.getProperty("file.separator") + "agenten_log.txt";

        //System.out.println("LogDatei: " + name);
        return name;
    }

    // DefaultVerzeichnis des "Datei(en) auswählen" Dialog
    public static String getImageDir()
    {
        //user.home -> User home directory
        //user.dir -> User working directory
        String dir = System.getProperty("user.dir") + System.getProperty("file.separator") + "PuzzleBilder" + System.getProperty("file.separator");

        //System.out.println("BilderVerzeichnis: " + dir);
        return dir;
    }

}
```

Abbildung 9.10: Programmdefaults

Zum einfacheren Auffinden entsprechen die Überschriften den Kommentarzeilen im Code.

9.5.2.1 „// Default für Anfangsbelegung der Filter“

Hier kann geändert werden mit welcher Filterstufe welcher Bereich bei Programmstart beginnt, diese Vorgabe kann während des Programmlaufs über den Vorgang „Log Auswahl“ geändert werden.

9.5.2.2 „// Default für Anfangsauswahl welcher ProgrammCode zum Lösen benutzte wird“

Selbsterklärend, diese Vorgabe kann während des Programmlaufs über den Vorgang „Mode“ geändert werden.

9.5.2.3 „// Default für Anfangsauswahl welcher Agent eingefärbt werden soll“

Selbsterklärend, diese Vorgabe kann während des Programmlaufs über den Vorgang „einfärben“ geändert werden.

9.5.2.4 „// Größe ResultArray (Spalten, Zeilen)“

Das Agentensystem (aus dem Vorprojekt) speichert das Ergebnispuzzle in einem Array dessen Größe hier je nach Puzzle geändert werden kann.

9.5.2.5 Log ausschreiben

9.5.2.6 „// ob überhaupt eine LogDatei auf Festplatte geschrieben wird“

9.5.2.7 „// Ort der LogDatei“

Das Programm kann am Ende des Lösevorgangs die Daten aus dem Logbereich in eine Datei schreiben. Ob und wohin dies geschieht wird hier festgelegt.

9.5.2.8 „// DefaultVerzeichnis des "Datei(en) auswählen" Dialog“

Selbsterklärend, natürlich kann dieses Verzeichnis in dem „Datei(en) auswählen“ Dialog während des Programmlaufs noch geändert werden.

9.6 Einschränkungen

Es wurde in der vorliegenden Arbeit zwei Vorarbeiten verbunden welche sich selber noch im Prototypenstatus befinden. Daher ist ein aufmerksames beobachten des Bedieners erforderlich. Situationen, welche in einem der Vorarbeiten schon zu Problemen, Deadlocks, Livelocks oder Abstürzen führten, tun dies immer noch, weil an den Vorprojekten nichts verändert wurde. Die Information über diese Situationen werden nicht immer aus den Vorprojekten weiter geleitet und können so nicht immer in diesem Programm erkannt und berücksichtigt werden. Ein aufmerksamer Bediener wird diese Situationen aber erkennen und entsprechend handeln.

9.6.1 Einschränkungen der Fotografie und der Puzzleteile

Um nur eine Anleitung zu haben wird hier kurz das Wesentliche aus dem Vorprojekt Bilderkennung (Quelle [Jung14], Benutzeranleitung Punkt 5) wiedergegeben. Die Puzzleteile müssen auf einem schwarzen, nicht spiegelnden Hintergrund liegen und die Umgebung sollte möglichst gleichmäßig beleuchtet sein. Es sind nur rein weiße Puzzleteile ohne Motiv möglich. Die Grundform der Puzzleteile sollte möglichst rechteckig sein mit nicht abgerundeten 90° Ecken. Die inneren Kanten müssen entweder weiblich oder männlich sein, die Außenkanten des Puzzles gerade. Verteilt werden die einzelnen Puzzlestücke auf der schwarzen Fläche in einem gedachten Schachbrett mit deutlichen Trennstreifen zwischen den Schachbrettfeldern.

9.6.2 Einschränkungen des Agentensystems

Das Agentensystem kann zur Zeit nicht immer erkennen ob „Sackgassen“ oder „im Kreis laufen“ vorkommt. Dies kann somit auch von der Oberfläche nicht immer weiter verarbeitet oder angezeigt werden. Somit ist ein aufmerksames Beobachten des Programms nötig.

9.7 Fehlerbehebung

Daher nur Puzzleaufnahmen zugelassen wurden welche das Vorprojekt [Jung14] erfolgreich durchlaufen haben kann der Fall einer schlechten Fotografie oder eines nicht charakterisierten Puzzlestückes nicht vorkommen.

Bei Problemen mit der Bilderkennung oder dem Agentensystem wird empfohlen den Vorgang mit der Funktion „Puzzleteilliste leeren“ zu unterbrechen. Dies kann aber aufgrund der Nebenläufigkeit der Agenten, z.B. bei „im Kreis“ laufender Agenten, auch bis zu 30 Runden dauern. Gleiche Verzögerungen gelten für die Funktionen „Dateiliste leeren“ und „Programm beenden“.

10 Ausblick

Während des Testens dieser Software wurde bewusst, dass die implementierte Lösungsstrategie sehr oft in Deadlocks bzw. Livelocks gerät. Dies liegt an der linearen Lösung des Puzzles. Es wird beginnend mit einem Puzzlestück ein angrenzendes Puzzlestück gesucht, das erste passende Puzzletück gewählt, und so fort, bis keine mehr übrig sind. Ein mechanisch passendes Puzzlestück muss aber nicht logisch an diesen Platz gehören. Diesen Fehler bemerkt man falls über die Ecke kein weiteres Puzzlestück mehr passt. Jetzt sollte die Software einen oder mehrere Schritte zurück nehmen und dort aus der Menge der anderen noch passenden Puzzlestücke ein anderes wählen und mit diesem den Vorgang fort setzen. Dieses Verhalten wäre für eine weitere Arbeit an diesem Projekt eine sinnvolle Ergänzung der Lösestrategie.

Eine Funktion im Framework JADE den Vorgang sinnvoll anzuhalten wurde nicht gefunden. Die Möglichkeiten in JADE welche vorhanden wären haben leider Seiteneffekte welche auf die Darstellung vor Publikum störende Auswirkungen haben. Ein Ausweg wäre das Einführen eines weiteren Agententyps. Eine Art „Bearbeitungsstation“ bei der sich genau zwei Puzzleteilagenten melden und welche die Passprüfung vornimmt. Für den Fall nur einer vorhandenen Vergleichstation ergäbe sich die gesuchte Linearisierung für den Schritt für Schritt Modus, falls mehr als ein Agent als Vergleichstation gestartet wird hätte man das bisherige nebenläufige Verhalten. Es wäre aber eine Veränderung des bestehenden Agentensystems und das Einführen weiterer Agententypen nötig gewesen, was in dieser Arbeit nicht vorgesehen gewesen war.

11 Schlusswort

Das vorrangige Ziel, die beiden Vorprojekte unter einer Oberfläche zusammen zu bringen und dabei den Lösevorgang darzustellen, wurde erreicht. Die Programmmeldungen wurden für Zuschauer aufbereitet. Zukünftige Erweiterungen der Puzzleagenten können in die Oberfläche eingebracht werden.

Literaturverzeichnis

[Göhn13] Göhner, P.: Skript zur Vorlesung Softwaretechnik I WS 13/14, IAS, 2013

[KrHa11] Krüger, G.; Hansen H.: Handbuch der Java-Programmierung, Addision-Wesley, 2011

[Ulle10] Ullenboom, Ch.: Java ist auch eine Insel, Galileo Press, 2010

[Emri13] Emrich, E.: Ausarbeitung der Diplomarbeit, IAS, 2013

[Jung14] Jung, T.: Ausarbeitung der Bachelorarbeit, IAS, 2014

[Inte01] Internet001: http://docs.oracle.com/javase/7/docs/index.html, Stand 03.07.2014

[Inte02] Internet002: http://stackoverflow.com/questions/17646076/how-to-start-jade-gui-within-another-gui, Stand 03.07.2014

[Inte03] Internet003: http://www.dpunkt.de/java/index.html, Stand 03.07.2014

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.