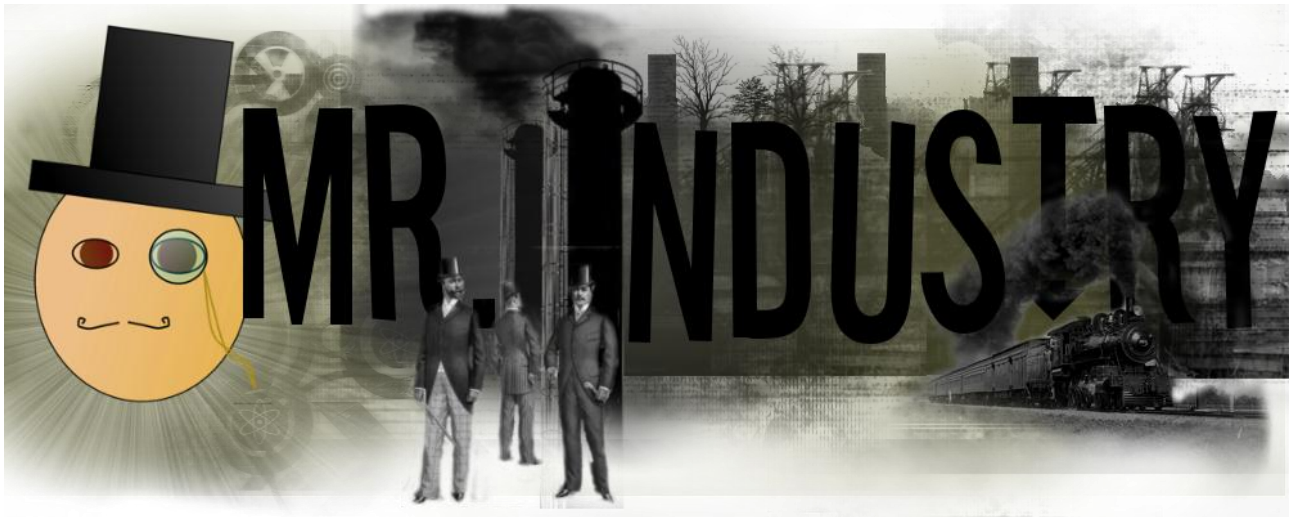


Realisierungsbericht



Status	In Arbeit
Projektname	Mr. Industry
Projektleiter	Oliver Schmidhauser
Auftraggeber	Herr Frieden / GIBB
Autoren	Luca Herzog / Oliver Schmidhauser

Änderungskontrolle, Prüfung, Genehmigung

Version	Datum	Beschreibung, Bemerkung	Name oder Rolle
0.1	25.4.2011	Ersterstellung	Luca Herzog
0.2	28.4.2011	Überarbeitung	Luca Herzog
0.3	2.5.2011	Überarbeitung	Oliver Schmidhauser
0.4	5.5.2011	Überarbeitung	Luca Herzog
0.5	7.5.2011	Überarbeitung	Oliver Schmidhauser
1.0	10.5.2011	Fertigstellung	Luca Herzog/Oliver Schmidhauser

Definitionen und Abkürzungen

Begriff / Abkürzung	Bedeutung
Tick	Das Abziehen oder Hinzufügen von Rohstoffen über eine vorbestimmte Zeit
GUI	Graphical User Interface: Die grafische Benutzeroberfläche eines Programms wo Eingaben getätigt werden können.

Referenzen

Referenz	Titel / Quelle
[1]	http://code.google.com/p/mrindustry/issues/list
[2]	http://code.google.com/p/mrindustry/source/browse/

Inhaltsverzeichnis:

1	Zweck des Dokuments	4
2	Technische Detailspezifikation	4
2.1	Innere Struktur	4
2.1.1	Struktur des Systemdesigns	4
2.1.2	Struktur des Systemdesigns	5
2.1.3	Beschreibung der Elemente	9
2.2	Schnittstellendefinitionen	10
2.3	Datenmodell	10
2.4	Sicherheit	10
3	Anforderungszuordnung	11
4	Systemdokumentation	14
4.1	Inline-Dokumentation	14
4.2	Benutzerhandbuch	14
4.2.1	Systemübersicht	14
4.2.2	Anwenderfunktionalität	18
4.3	Supporthandbuch	18
4.3.1	Massnahmen bei Benutzerproblemen	18
4.3.2	Massnahmen bei technischen Problemen	18
4.3.3	Anhang zum Supporthandbuch	18
5	Systemtest	19
5.1	Testspezifikation	19
5.1.1	Kritikalität der Funktionseinheit	19
5.1.2	Testanforderungen	19
5.1.3	Testverfahren	19
5.1.4	Testkriterien	19
5.1.5	Testfälle	20
5.2	Testprozedur	21
5.2.1	Vorbereitung	21
5.2.2	Durchführung	22
5.2.3	Nachbearbeitung	22
5.3	Testprotokoll	22
5.3.1	Testauswertung	23
6	Mittelbedarf	23
7	Planung und Organisation	24
8	Wirtschaftlichkeit	24
9	Konsequenzen	25
10	Antrag	25
11	Abbildungsverzeichnis	25
12	Anhang	26
	MrIndustry.java	26
	Baumenü.java	45
	Spielkarte.java	54
	SpielObjekt.java	65
	LeistungSteigerungsObjekt.java	71
	PrivatesObjekt.java	72
	Rohstoffgewinnung.java	72
	Umgebung.java	73
	Verarbeitung.java	77
	Zentrallager.java	77
	Autofabrik.java	79
	WohnHausMittelKlasse.java	79
	WohnHausOberKlasse.java	80
	WohnHausUnterKlasse.java	80
	Bergwerk.java	81
	Eisenmine.java	81

Farm.java	82
Försterei.java	82
Baeckerei.java	83
Werkzeugfabrik.java	84



1 Zweck des Dokuments

Zusammenfassung der Ergebnisse der Phase „Realisierung“.

2 Technische Detailspezifikation

2.1 Innere Struktur

2.1.1 Struktur des Systemdesigns

Unser Programm läuft unter Java, wofür man auf dem Gerät auf dem man es ausführen möchte eine funktionierende Java-Umgebung benötigt. Ansonsten werden keine zusätzlichen Programme oder Plugins benötigt. Eine lauffähige Umgebung



2.1.2 Struktur des Systemdesigns

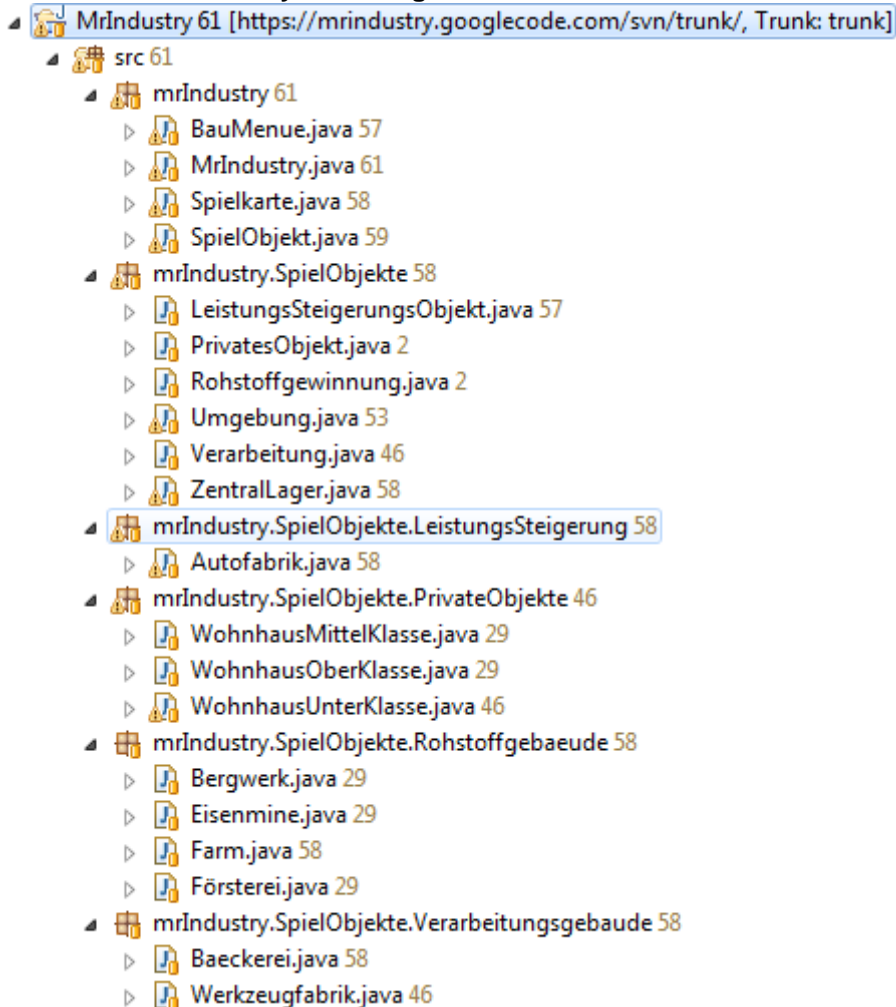


Abbildung 1 - Ordnerhierarchie unserer Applikation

Beschreibung der Pakete und Ordnern:

Im mrIndustry-Paket befinden sich die Klassen welche die hauptsächliche Spiellogik beinhalten. Die Gebäude die errichtet werden sind kategorisiert in den jeweiligen Unterordnern im SpielObjekte-Paket. In diesen Paketen befinden sich Klassen welche die Eigenschaften der verschiedenen Objekte beschreiben welche man im Spiel bauen kann. Die genauen Aufgaben der jeweiligen Klassen wird im späteren Verlauf des Dokuments beschrieben. Die Texturen welche das Spiel benötigt befinden sich alle im src Ordner damit der Dateipfad nicht zu kompliziert wird. Die Zahlen hinter den jeweiligen Dateien bedeuten sozusagen die Version der Datei, also wieviele Male die Datei schon geändert wurde. Diese Zahlen kommen vom SVN-Plugin welches wir zum Code-Hosting benutzen."

Auf den nächsten Seiten sind alle Klassendiagramme des Projekts ersichtlich, die Beschreibungen der Klassen dazu befinden sich ebenfalls in diesem Dokument.

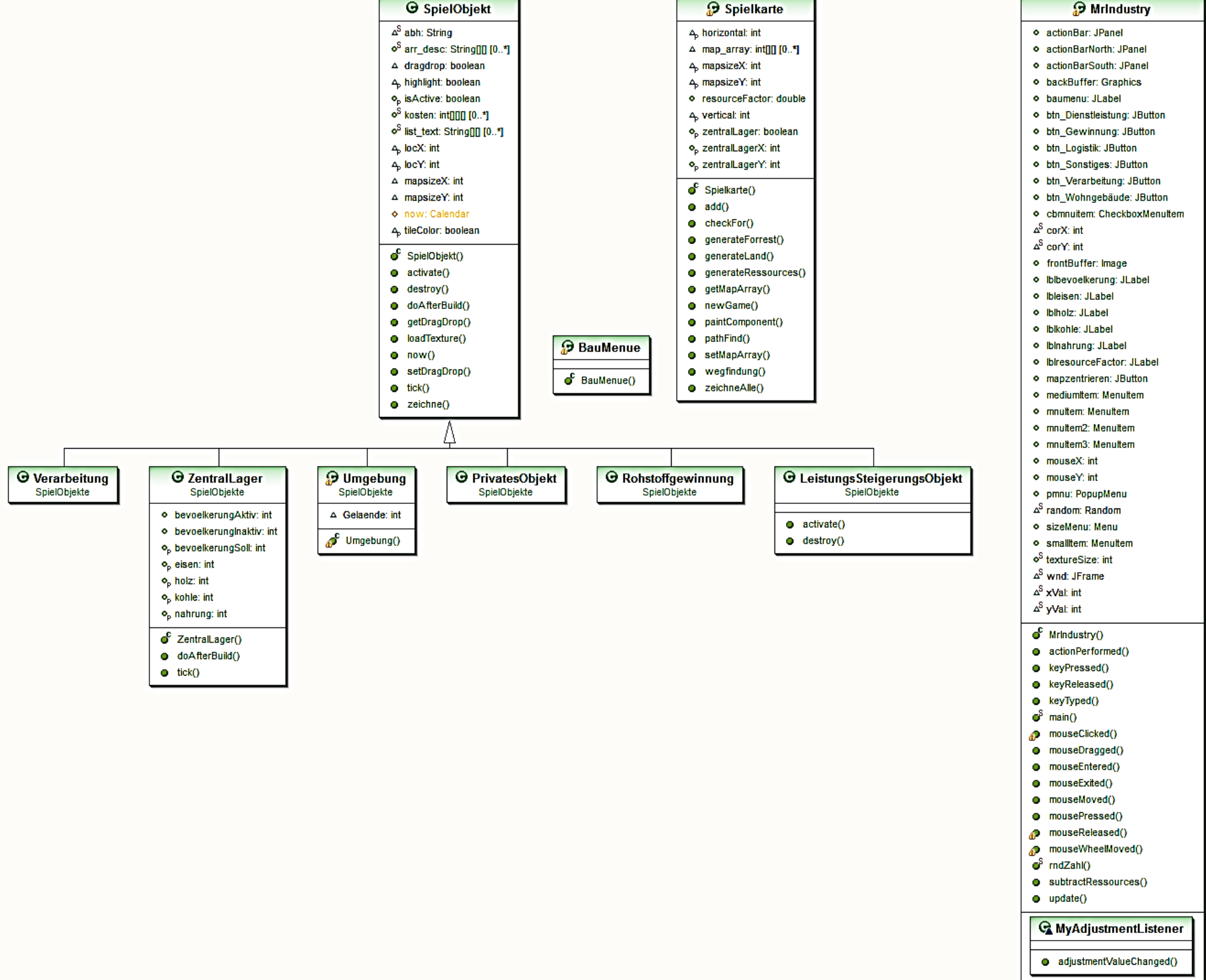


Abbildung 2 - Klassendiagramm Paket MrIndustry

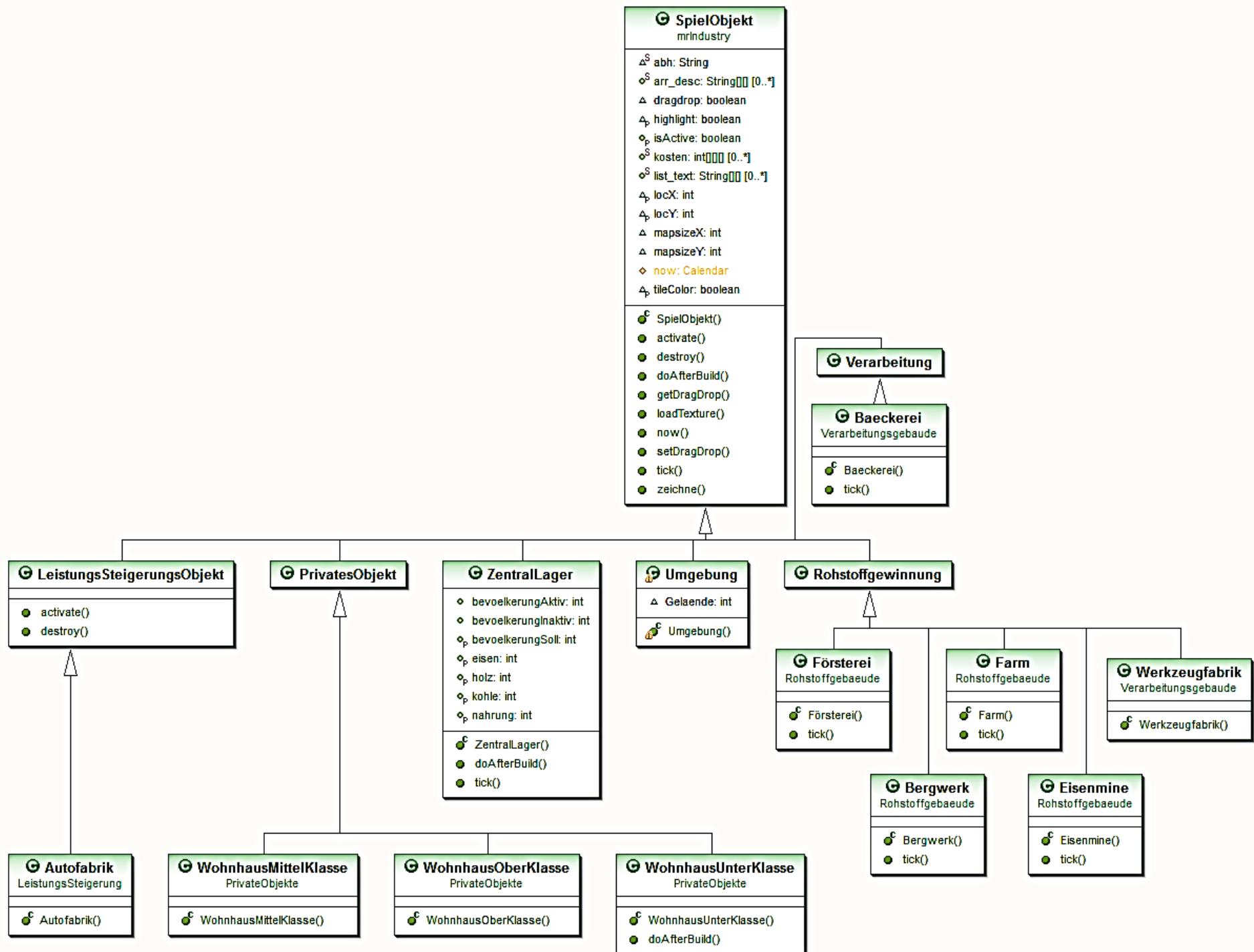


Abbildung 5 - Klassendiagramm des Pakets SpielObjekt

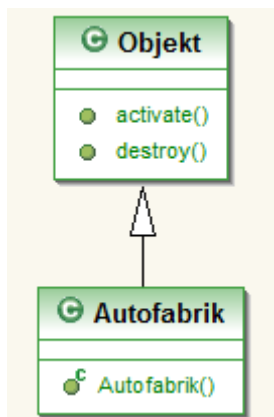


Abbildung 6 - Klassendiagramm des Pakets LeistungsSteigerungsObjekt

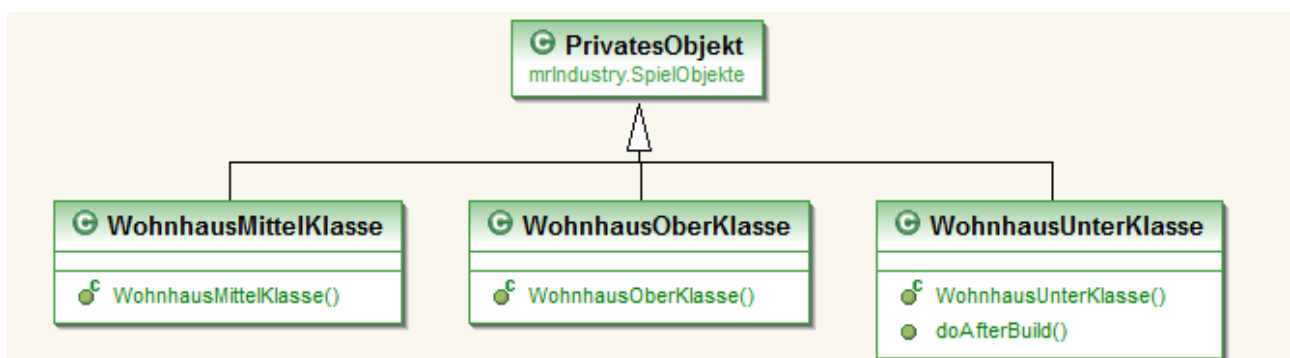


Abbildung 7 - Klassendiagramm des Pakets PrivatesObjekt

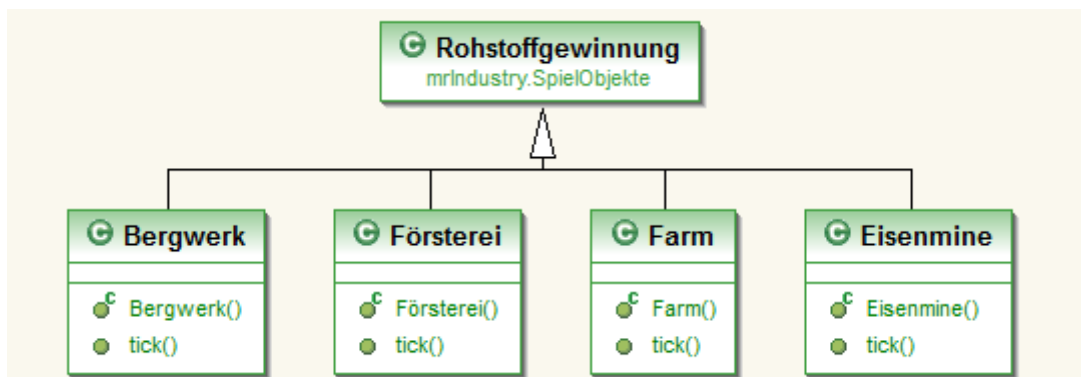


Abbildung 8 - Klassendiagramm des Pakets Rohstoffgewinnu

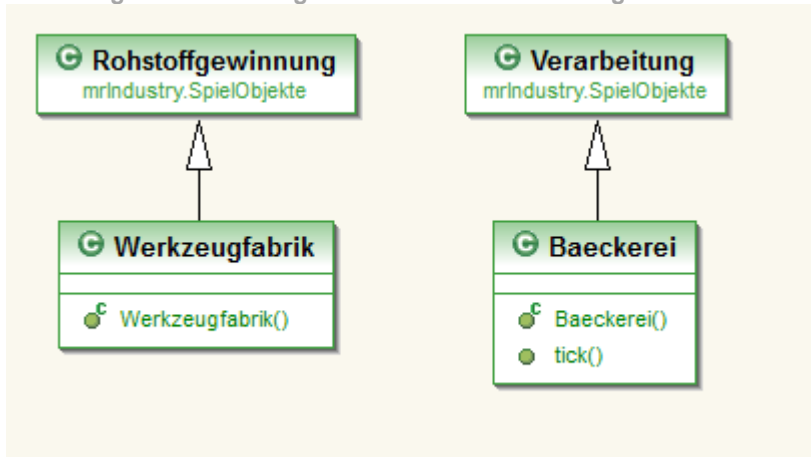


Abbildung 9 - Klassendiagramm des Pakets Verarbeitung

2.1.3 Beschreibung der Elemente

Im folgenden Teil sind die Klassen des Spiels beschrieben:

Werte welche vom Konzept abweichen sind hier vermerkt, ansonsten stehen sie im Konzept.

- mrIndustry
 - BauMenue.java
Ermöglicht die Darstellung der Baumenü-Form. Das Baumenü ändert seinen Look je nachdem welchen Gebäudetyp man im Spiel ausgewählt hat. Das Baumenü erstellt auch entsprechend neue Objekte auf der Spielkarte wenn der User etwas bauen will.
 - MrIndustry.java
Dies ist die Hauptklasse des Spiels, welche alle anderen Elemente aufruft und den Spielverlauf bestimmt. In dieser Klasse werden die „Ticks“ und die Darstellungsfunktionen aller Unterklassen ausgeführt. Zusätzlich verarbeitet diese Klasse die Benutzereingaben, verarbeitet die Kollisionserkennung und führt dementsprechend Code aus.
 - Spielkarte.java
Diese Klasse generiert, wie der Name schon besagt, die gesamte Spielkarte des Spiels, welche dynamisch erstellt werden kann. Den Funktionen der Klasse können Prozentwerte übergeben werden, wie viel Anteil ein Landschaftsobjekt auf der Spielkarte haben soll und wie gross die Karte sein soll.. Die Karte. Diese Klasse verwaltet auch alle Objekte welche sich auf der Karte befinden und führt deren Paint-Methoden aus. Die Karteneigenschaften sind während des ganzen Spielverlaufs verfügbar.
 - SpielObjekt.java
In dieser Klasse befinden sich hauptsächlich Getter und Setter und Informationen für die jeweiligen Objekte. Über diese Klasse wird die Position der Objekte, die Aktivität eines Objekts, ob ein Objekt am Mauszeiger haften bleiben muss, ob ein Kartenfeld umrandet werden muss und mit welcher Farbe es umrandet werden muss bestimmt und abgerufen. Zusätzlich befindet sich hier das „Texturen-Management“, welches das Laden von neuen Texturen oder das Abrufen schon bestehender Texturen ermöglicht. Auf einem Kartenfeld können mehrere Ebenen von Texturen angezeigt werden, dies erspart einem viel Aufwand beim Bearbeiten von Bildern bzw. Texturen. Über diese Klasse wird zusätzlich noch die Zeit abgerufen, welche für den „Tick“ benötigt wird.
- mrIndustry.SpielObjekte
Hier befinden sich die Oberklassen für die jeweiligen Unterklassen in den weiteren Paketen. Die Oberklassen beschreiben alles der Unterklassen was sie gemeinsam haben. In den Unterklassen dieser Klassen ist ausserdem bestimmt, welche Texturen für das jeweilige Gebäude geladen werden müssen.
 - LeistungsSteigerungsObjekt.java
Ein „Leistungssteigerungsobjekt“ beschreibt ein Objekt/Gebäude in unserem Spiel, welches die Arbeitsleistung der gesamten Siedlung des Spielers erhöht.
In der Klasse wird der Faktor bestimmt, um den ein Gebäude die Leistung erhöhen kann. Dieser Faktor wird aus den Unterklassen geholt. Der „Leistungsfaktor“ eines einzelnen Leistungssteigerungsobjekts zählt nicht nur für einzelne Gebäude, sondern steigert gleich die gesamte Leistung der Siedlung.
 - PrivatesObjekt.java
Die speziellen Fähigkeiten eines privaten Objekts im Spiel umfassen das Hinzufügen von Bewohnern beim Bau des Objekts. Wieviele Bewohner beim Bau hinzugefügt werden ist jeweils in den Unterklassen bestimmt.
 - Rohstoffgewinnung.java
Die Gebäude welche Rohstoffe gewinnen. Wieviele Rohstoffe das sind wird in den Unterklassen beschrieben.
 - Umgebung.java
Diese Klasse bestimmt welche Umgebungstexturen bei welchem Objekt geladen werden sollen. Zusätzlich werden Szenerie-Elemente auf der Karte geladen, wessen Positionen aber an keinem Ort gespeichert werden. Diese erscheinen zufällig auf der Karte.
 - Verarbeitung.java
Ein Verarbeitungsobjekt kann eine gewisse Menge an Ressourcen in eine andere Art Ressourcen umwandeln. Welche Ressourcen in welche umgewandelt werden bestimmen die Unterklassen.
 - Zentrallager.java
Das Zentrallager ist die Klasse, welche alle Ressourcen verwaltet. Über diese Klasse können bei einem Tick, Zerstörung oder Bau eines Objekts Ressourcen hinzugefügt oder abgezogen werden.

- mrIndustry.SpielObjekte.Leistungssteigerung
 - Autofabrik.java
Die Autofabrik steigert die Leistung um 0.1.
- mrIndustry.SpielObjekte.PrivateObjekte
 - WohnHausOberKlasse.java
Ein Wohnhaus der Oberklasse fügt zehn Bewohner hinzu.
 - WohnHausMittelKlasse.java
Fügt sieben Bewohner hinzu.
 - WohnHausUnterKlasse.java
Fügt fünf Bewohner hinzu.
- mrIndustry.SpielObjekte.Rohstoffgebaeude
In diesem Paket sind die Klassen welche Rohstoffe gewinnen.
 - Bergwerk.java
 - Eisenmine.java
 - Farm.java
 - Försterei.java
- mrIndustry.SpielObjekte.Verarbeitungsgebäude
 - Baeckerei.java
 - Werkzeugfabrik.java

2.2 Schnittstellendefinitionen

Zentrallager - HUD

Das HUD zeigt die Anzahl Rohstoffe mithilfe von Informationen aus dem Zentrallager an.

Spielkarte - Unterklassen

Liest die Positionen der Objekte aus den Unterklassen aus und stellt diese dar.

BauMenu – MrIndustry

Erstellt ein neues Gebäude im Spiel, je nachdem was der User auswählt.

2.3 Datenmodell

Siehe Klassendiagramme und deren Beschreibung.

2.4 Sicherheit

Unser Programm ist ein Spiel bei welchem niemals irgendwelche sensiblen Dateneingaben getätigt werden, darum bestehen auch keine Datenschutzanforderungen.

3 Anforderungszuordnung

Im Verlauf der Entwicklung wurden einige Klassen nicht umgesetzt, andere wurden neu hinzugefügt und die Hierarchie wurde überarbeitet.

Die Strassen wurden in der jetzigen Version noch nicht umgesetzt, werden später aber noch integriert.

Klassenhierarchie:

1. MrIndustry
2. Spielkarte
3. Baumenü
4. Hauptmenü
5. SpielObjekt
 - a. Zentrallager
 - b. Rohstoffgewinnung
 - i. Eisenmine
 - ii. Bergwerk
 - iii. Farm
 - iv. Försterei
 - c. LeistungssteigerungsObjekt
 - i. Autofabrik
 - ii. Statue
 - iii. Park
 - d. PrivatesObjekt
 - i. WohnhausUnterklasse
 - ii. WohnhausMittelklasse
 - iii. WohnhausOberklasse
 - e. Umgebung
 - f. Verarbeitung
 - i. Bäckerei
 - ii. Werkzeugfabrik

Klassennummerierung(für Anforderungstabelle):

- | | |
|--------------------------------|--------------------------|
| 1. MrIndustry | 13. Autofabrik |
| 2. Spielkarte | 14. Werkzeugfabrik |
| 3. Baumenü | 15. Statue |
| 4. Hauptmenü | 16. Park |
| 5. SpielObjekt | 17. PrivatesObjekt |
| 6. Zentrallager | 18. WohnhausUnterkasse |
| 7. Rohstoffgewinnung | 19. WohnhausMittelklasse |
| 8. Eisenmine | 20. WohnhausOberklasse |
| 9. Bergwerk | 21. Umgebung |
| 10. Farm | 22. Verarbeitung |
| 11. Försterei | 23. Bäckerei |
| 12. LeistungssteigerungsObjekt | |

Tabellarische Übersicht der Anforderungen:

3	Zeigt essentielle Informationen über das Spiel	X					X																			
4	Zeigt das Hauptmenü und das Pausenmenü des Spiels an				X																					
5	Oberklasse der Spielobjekte mit der Paint Methode					X																				
6	Ressourcenverwaltung im Zentrallager	X					X																			
7	Oberklasse Rohstoffgewinnung						X																			
8	Oberklasse Leistungssteigerung									X																
9	Oberklasse Privater Objekte														X											
10	Verbrauchen Rohstoffe und generieren andere							X	X		X		X	X	X	X		X	X	X						
11	Speichert und generiert Geländearten (Wasser, Wald, Eisenvorkommen, Kohlevorkommen, Wiese)																									
12	Routine welche die Paint-Methode der Klassen aufruft und Konflikte beim Bauen überprüft		X																							
13	Oberklasse Verarbeitung																									
14	Überprüft Konflikte beim Bauen	X	X																							
15	Generiert die Spielwelt		X																							
16	Zieht nach einer gewissen Zeit dem Zentrallager Rohstoffe ab oder fügt diese hinzu	X					X																			
17	Speichert die Kosten und Beschreibungen der Gebäude					X																				
18	Zeigt das Kontextmenü mit den richtigen Daten an.																									
19	Fügt neue Gebäude hinzu oder entfernt diese.	X		X																						
20	Berechnet die Karte neu beim Zoomen oder Bewegen	X		X																						

Seite 13 von 84

4 Systemdokumentation

4.1 Inline-Dokumentation

Die Erläuterung der Funktionalität im Quellcode befindet sich im Anhang.

4.2 Benutzerhandbuch

Zuerst vorab: Der Rohstoff „Geld“ existiert jetzt in unserem Spiel nicht mehr, jedoch war es so im Konzept vorgesehen. Die Verarbeitungsgebäude erhöhen jetzt die Arbeitsgeschwindigkeit der Siedlung, anstatt Geld zu produzieren.

4.2.1 Systemübersicht

Das Spiel ist im Grunde genommen einfach zu bedienen.

Spielstart:

Beim Spielstart wird dem User das Hauptmenü angezeigt, dort kann er die Kartengrösse wählen. Wir empfehlen die Karte nicht grösser als 100 (100x100 Felder) zu machen, ansonsten leidet die Performance des Spiels. Nach dem Klick auf Start wird das neue Spiel generiert. Das Pausenmenü kann während des Spiels jederzeit durch drücken der ESC-Taste aufgerufen werden und ein neues Spiel gestartet werden. Das Spiel benötigt eine Weile zum Laden, währenddessen wird hauptsächlich die Spielkarte generiert und die Texturen werden geladen.

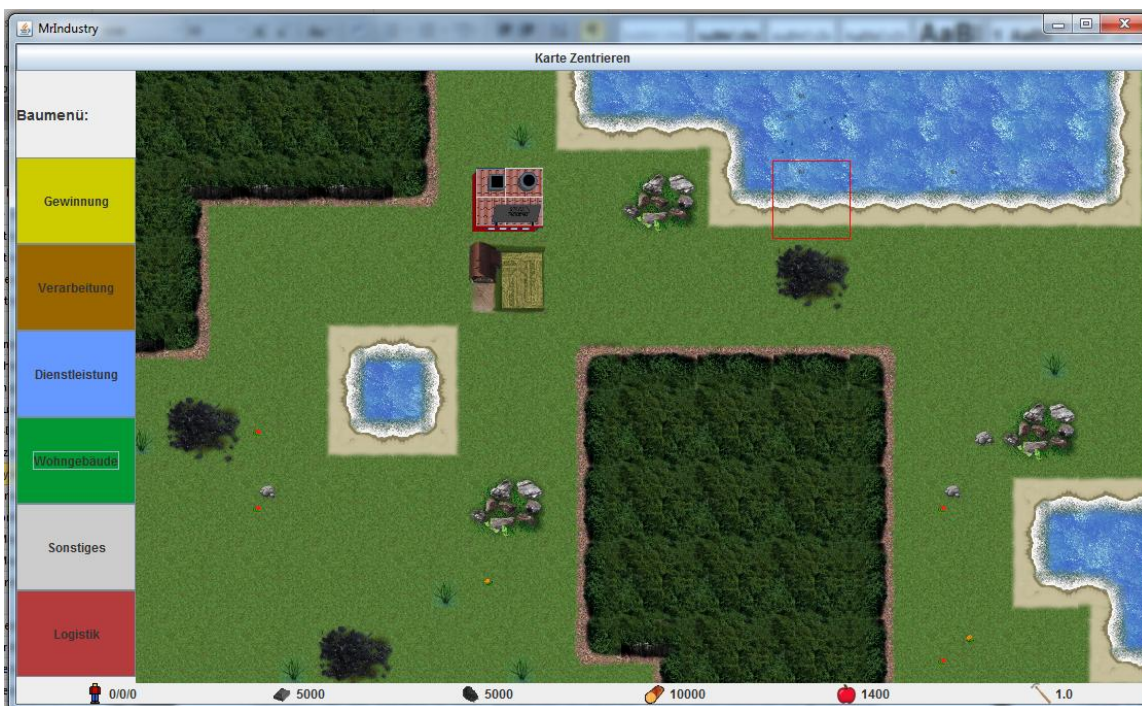


Abbildung 10 - Das Spiel zeigt die generierte Karte - Das Zentrallager und eine Farm wurden bereits gebaut

Am Anfang muss das Zentrallager gebaut werden, dort werden alle Rohstoffe des Spielers gespeichert. Im Baumodus sieht man am Viereck um das Gebäude herum ob es gebaut werden kann oder nicht. Rot bedeutet dass das Gebäude nicht errichtet werden kann, bei gelb kann man bauen. Wenn man einen passenden Platz für das Gebäude gefunden hat klickt man auf den Platz und das Gebäude ist errichtet.

Nun erscheinen in der unteren Leiste die Rohstoffe welche man zur Verfügung hat:

Bevölkerung	Eisen	Kohle	Holz	Nahrung	Arbeitsfaktor
Speicherdatum: ####.###					

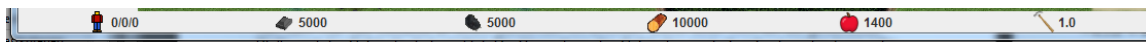


Abbildung 11 - Anzeige der Rohstoffe

Gebäudewahl:

Wenn man ein weiteres Gebäude bauen will, wählt man in der linken Leiste die Kategorie aus in der sich das Gebäude befindet. Dann wählt man im neuen Fenster aus der Liste das gewünschte Gebäude aus und klickt auf „Auswahl bauen“.



Abbildung 12 - Das Baumenü der Kategorie Gewinnung

Baumodus:

Nun kann man das Gebäude an einem passenden Ort errichten. Durch einen Klick auf die gewünschte Stelle wird das Gebäude dort errichtet. Die Umrandung des Gebäudes zeigt ob gebaut werden kann oder nicht:



Abbildung 13 - Das Gebäude kann hier nicht errichtet werden



Abbildung 14 - Das Gebäude kann errichtet werden

Die Abhängigkeit der Position der Gebäude steht in der Beschreibung im Baumenü.

Die Gebäude benötigen Ressourcen zum Betrieb, jedoch gewinnen sie auch wieder Rohstoffe.

Dekonstruktiver-Modus:

Wenn man ein Gebäude zerstören oder deaktivieren will, klickt man mit der rechten Maustaste auf das Gebäude bei welchem die Aktion ausgeführt werden soll und klickt auf die gewünschte Aktion.



Abbildung 15 - Dekonstruktiver Modus

Wenn ein Gebäude deaktiviert wird, benötigt es keine Ressourcen mehr, stellt aber auch keine mehr her. Bei der Zerstörung gibt es wieder Platz für neue Gebäude und die Hälfte des Kaufpreises wird rückerstattet.

Ein Gebäude kann auch automatisch vom Spiel deaktiviert werden, wenn für den Betrieb zu wenig Ressourcen vorhanden sind. Der User kann dies nicht verhindern. Wenn ein Gebäude inaktiv wird erscheint über dem Gebäude ein rotes Kreuz.

View-Modus:

Im View-Modus kann man die Spielkarte ansehen. Die Karte wird bewegt indem man die linke oder mittlere Maustaste auf der Karte gedrückt hält und diese nach Belieben umherzieht um die Karte zu verschieben. Durch Drehen des Mausekzes kann man die Karte vergrößern oder verkleinern.



Abbildung 16 - Eine Spielkarte (30x30 Felder) auf kleinster Zoom-Stufe

Strategien und Spielablauf:

Wenn das Spiel gestartet wurde muss wie gesagt zuerst das Zentrallager gebaut werden. Danach erhält der Spieler von allen Rohstoffen eine bestimmte Anzahl, ausser von den Bewohnern. Deshalb wird empfohlen zuerst Wohngebäude zu errichten damit die anderen Gebäude in Betrieb genommen werden können. Sobald ein Wohngebäude steht muss man aber warten bis die Anzahl an Soll Bevölkerung produziert wird. Die Bevölkerungsanzeige ist folgendermassen aufgebaut: 0/0/0. Die dritte Stelle bedeutet wieviel Bevölkerung in den Häusern Platz hat. Die zweite Stelle bedeutet wieviele davon Leute effektiv in der Siedlung des Spielers sind, und die erste Stelle besagt, wieviele Leute von der zweiten Stelle am arbeiten sind. Wenn dem Spieler die Nahrung ausgeht, sinkt die Anzahl an Bewohnern pro Sekunde um einen Bewohner.

Anmerkung:

Da das Spiel noch nicht zu 100 % ausbalanciert ist, kann es sein dass es gewisse Konflikte beim Bauen gibt. Zum Beispiel in ein paar Fällen meldet das Spiel dass man zu wenig Holz hat wenn man eine Eisenmine bauen will und wenn man eine Försterei bauen will meldet das Spiel dass man zu wenig Eisen hat.

4.2.2 Anwenderfunktionalität

Aufgabe:

Der Anwender errichtet sein eigenes kleines Imperium und versucht dieses erfolgreich zu verwalten.

Applikation starten:

Wir als Entwickler starten unsere Applikation via Eclipse. Der Benutzer startet seine Applikation jedoch durch Ausführen der kompilierten Java-Datei (.jar).

Applikation beenden:

Die Applikation kann im Kontextmenü unter „Beenden“, oder durch Schliessen des Fensters beendet werden.

Wiederanlauf (Reset):

Wenn das Spiel beendet wird geht jeglicher Spielfortschritt verloren, eine Speicherfunktion wurde nicht entwickelt.

Fehlermeldungen:

Im Verlauf des Spiels ist es möglich, dass das Spiel dem User meldet, dass er von einer gewissen Art von Ressourcen nicht genügend zur Verfügung hat. Diese Fehlermeldung kann umgehen werden indem man mehr Ressourcen der benötigten Art sammelt.

4.3 Supporthandbuch

4.3.1 Massnahmen bei Benutzerproblemen

Problem:

- Das Spiel liefert eine Exception und stürzt ab:
In diesem Fall kann nichts anderes gemacht werden als das Spiel neu zu starten. Der Spielfortschritt geht dabei verloren. Wenn möglich sollte der Text der Exception kopiert werden und auf unserer Google Code Projektseite als Issue gepostet werden.
- Das Spiel liefert eine Exception, läuft aber weiter:
In diesem Fall kann weitergespielt werden unter der Bedingung, dass dann eventuell etwas unerwartet nicht so funktioniert wie es sollte. Wenn möglich sollte der Text der Exception kopiert werden und auf unserer Google Code Projektseite als Issue gepostet werden.
- Das Spiel ruckelt stark:
In diesem Fall ist der Computer auf dem das Spiel ausgeführt wird zu schwach oder die Spielkarte zu gross. Wenn möglich sollte die Spielkarte verkleinert werden.

Wer Lust hat kann die Probleme selbst angehen, indem er unseren Projekt-Quellcode von der Google Code Projektseite herunterlädt [2].

4.3.2 Massnahmen bei technischen Problemen

Da unser Spiel nicht von einem externen System abhängig ist, sollte es eigentlich funktionieren solange die Umgebung auf der es ausgeführt wird eine Java-Umgebung installiert hat und funktionsfähig ist.

4.3.3 Anhang zum Supporthandbuch

Auf unserer Google-Code-Projektseite [2] können Probleme mit der Applikation gemeldet werden. Die User können dort ihre entdeckten Fehler oder ev. auch Anliegen melden und wir werden diese dann umsetzen/beheben.

5 Systemtest

5.1 Testspezifikation

5.1.1 Kritikalität der Funktionseinheit

Spielstart: Hoch
Gebäudewahl: Mittel
Baumodus: Hoch
Dekonstruktiver Modus: Hoch
View-Modus: Hoch
Ressourcenmanagement: Hoch

Alle Elemente ausser die Gebäudewahl der Applikation haben die Kritikalität hoch, da wenn eine Funktion fehlerhaft ist, der weitere korrekte Spielverlauf nicht garantiert ist. Bei der Gebäudewahl ist es entbehrlich wenn man zum Beispiel eine Statue nicht bauen kann.

5.1.2 Testanforderungen

Um den Test durchzuführen wird eine funktionierende

5.1.3 Testverfahren

Die von uns durchgeführten Tests bedürfen keiner Generierung von Testdaten, jedoch muss man wissen welche Bedingungen man schaffen sollte. Dies wird in den Testfällen genauer erläutert.

5.1.4 Testkriterien

Abdeckungsgrad:

Die verschiedenen Modi unseres Spiels sollen in 9 von 10 Fällen erfolgreich funktionieren.

Checklisten:

- ✓ Erscheint der Spielablauf nach dem Test logisch?
- ✓ Liefert das Programm das gewünschte Resultat zurück?
- ✓ Entspricht der Ablauf des Programms dem beschriebenen Soll-Zustand?
- ✓ Funktionieren alle Elemente des Spiels auch noch im Nachhinein?

Endkriterien:

Der Test ist dann erfolgreich abgeschlossen, wenn der Tester vom Programm angezeigt bekommt was er auch erwartet hat.

5.1.5 Testfälle

Die Prüfergebnisse der folgenden Testfälle entsprechen den aktuellsten Resultaten. Die Resultate im Testprotokoll können jedoch auch ältere sein.

Nr.	AFo-Nr.	Anwendungsfall (ggf. orientiert an Use Cases)	Ausgangs- situation	Eingabe- daten	erwartetes Ergebnis	Bemerkungen, Prüfergebnis
01	01	Spiel eröffnen	Anzeigen der generierten Karte mit dem GUI	Der gewählte Schwierigkeit sgrad im Hauptmenü.	Spiel wird erstellt mit der entsprechenden Anzahl Ressourcen	Erfolgreich
02	02	Baumenü durchforsten	Anzeigen der verschiedenen Gebäudetypern im Menü	Die Auswahl in der Listbox des Baumenüs.	Die richtige Beschreibung und die richtigen Kosten werden geladen.	Erfolgreich
03	03	Ressourcen anschauen	Anzeigen der verfügbaren Ressourcen	Das Zentrallager muss zuerst gebaut werden.	Anzeige der Ressourcen welche sich im Zentrallager befinden.	Erfolgreich
04	04	Hauptmenü öffnen				
05	19	Gebäude bauen (mit falscher Position)05	Errichten eines Gebäudes auf einer Position wo es nicht erlaubt ist.	Testweise wählen wir eine Position wo das Gebäude nicht gebaut werden darf.	Das Spiel verhindert den Bau.	Erfolgreich
06	19	Gebäude bauen	Errichten eines Gebäudes der Kategorie Gewinnung.	Das Gebäude so platzieren dass es gebaut werden darf.	Das Spiel baut das Gebäude, zieht die Kosten ab und es beginnt Rohstoffe zu gewinnen.	Erfolgreich
07	19	Gebäude bauen	Errichten des Zentrallagers	Das Gebäude so platzieren dass es gebaut werden darf.	Das Spiel errichtet das Zentrallager und zeigt ab dann die verfügbaren Ressourcen an.	Erfolgreich

08	19	Gebäude zerstören	Versuch das Zentrallager zu zerstören (nicht erlaubt)	Rechtsklick auf das Zentrallager und dann auf Gebäude zerstören	Das Gebäude dürfte nicht zerstört werden können (Menüpunkt deaktiviert)	Erfolgreich
09	19	Gebäude zerstören	Ein Wohnhaus wird zerstört	Rechtsklick auf das Wohnhaus und dann auf zerstören	Das Gebäude wird zerstört und verschwindet von der Spielkarte	Erfolgreich
10	01,19	Gebäude deaktivieren	Eine Farm wird deaktiviert	Rechtsklick auf die Farm und dann auf deaktivieren	Das Gebäude hört auf zu produzieren, der Menüpunkt im Kontextmenü wechselt auf „Gebäude aktivieren“	Erfolgreich
11	11	Riesige Karte generieren	Versuch eine Karte mit 100x100 Feldern zu generieren.	Im Programmcode den Mapsize wert von X & Y auf 100 stellen.	Die Karte wird generiert und angezeigt.	Erfolgreich, jedoch ruckelt das Spiel so stark.
12	20	Karte bewegen	Die Karte anschauen	Mit dem Cursor die Karte packen und zur gewünschten Stelle fahren.	Die Karte wird verschoben und der gewünschte Punkt angezeigt.	Erfolgreich
13	20	Karte Zoomen	Die Karte von Weitem und von Nahem anschauen.	Mit dem Mausrad die Kartengrösse verändern.	Die Kartengrösse wird entsprechend verändert.	Erfolgreich, jedoch wird die Karte etwas verschoben beim Zoomen.
14	11	Winzige Karte generieren	Eine Karte der grösse 4x4 wird generiert.	Im Programmcode den Mapsize wert von X & Y auf 4 stellen.	Eine Karte mit sehr wenig Land wird generiert	Fehlgeschlagen , die Karte enthält nur Wasser.

5.2 Testprozedur

Was genau bei welchem Testfall vorgenommen werden muss steht in den Testfällen. Der Benutzer kann die Testverläufe mithilfe des Benutzerhandbuchs leicht nachvollziehen.

5.2.1 Vorbereitung

Damit man weiss wie man die Tests durchführen kann, sollte man sich zur korrekten Bedienung die Benutzeranleitung zu Gemüte führen.

5.2.2 Durchführung

Die Tests werden manuell ausgeführt. Der Tester befolgt die Anweisungen in den Testfällen und wertet die Tests nachher anhand der Checkliste aus.

5.2.3 Nachbearbeitung

Misslingt ein Test, wird dieser von uns vermerkt und wir versuchen den Fehler dann schnellstmöglich zu beheben.

5.3 Testprotokoll

Datum	Testfall	Name	Erfolgreich+Bemerkungen (Y/N)
18.4.2011	12	Oliver Schmidhauser	Y
19.4.2011	14	Luca Herzog	Y
19.4.2011	02	Luca Herzog	N, das Baumenü zeigt die Kosten nicht korrekt an.
20.4.2011	13	Oliver Schmidhauser	Y
20.4.2011	06	Oliver Schmidhauser	Teilweise erfolgreich, das Gebäude kann noch keine Rohstoffe produzieren
23.4.2011	06	Oliver Schmidhauser	Y
26.4.2011	08	Luca Herzog	Y
28.4.2011	03	Oliver Schmidhauser	Y
28.4.2011	06	Oliver Schmidhauser	Y
29.4.2011	05	Luca Herzog	Teilweise erfolgreich, die Försterei kann auf einer Wiese platziert werden, dies darf nicht geschehen.
29.4.2011	05	Luca Herzog	Y
30.4.2011	11	Oliver Schmidhauser	Y
30.4.2011	14	Oliver Schmidhauser	Y
1.5.2011	10	Luca Herzog	N, das Gebäude wird nicht deaktiviert obwohl man im Kontextmenü den Menüpunkt deaktivieren wählt
1.5.2011	10	Luca Herzog	Y
3.5.2011	09	Oliver Schmidhauser	N, das Gebäude verschwindet mitsamt dem Untergrund von der Karte
3.5.2011	09	Oliver Schmidhauser	Y
4.5.2011	07	Luca Herzog	Y
5.5.2011	04	Oliver Schmidhauser	Y

5.3.1 Testauswertung

Die Tests wurden alle spezifisch ausgewertet und die erforderlichen Massnahmen damit der Test das nächste Mal korrekt durchgeführt wird ergriffen.

6 Mittelbedarf

Schätzung über den Bedarf an:

- Sachmittel
 - 2 Computer
 - Java
- Personal
 - 1 Entwickler
 - 1 Projektmanager
- Ausbildung
 - Lehrlinge
- Dienstleistungen
 - Das Internet



7 Planung und Organisation

- Projektorganisation
 - Das Projekt wird laut der Hermes-Methodik hauptsächlich aufgrund eines Projektplans geplant und durchgeführt.
- Anwenderorganisation
 - Die Organisation erfolgt durch den Projektleiter, welcher möglichst den Projektplan einhält. (wer in welcher Phase als Leiter aktiv ist, ist auf dem Projektplan ersichtlich)
- Termine:

Datum	Woche	Projektphase
01.02.2011	5	
08.02.2011	6	Initialisierung abgeschlossen
15.02.2011	7	
22.02.2011	8	
01.03.2011	9	Voranalyse abgeschlossen
08.03.2011	10	
15.03.2011	11	
22.03.2011	12	Konzept abgeschlossen
29.03.2011	13	
05.04.2011	14	Frühlingsferien
12.04.2011	15	
19.04.2011	16	
26.04.2011	17	
03.05.2011	18	
10.05.2011	19	
17.05.2011	20	Realisierung abgeschlossen
24.05.2011	21	
31.05.2011	22	Einführung abgeschlossen
07.06.2011	23	Abschlussphase
14.06.2011	24	
21.06.2011	25	Reserve / Repetition
28.06.2011	26	MAP

- Prioritäten
 - Der erfolgreiche Abschluss des Projektes mit gutem Management hat oberste Priorität.
 - Zahlreiche Funktionen die noch eingebunden werden könnten, und schön zu haben wären, haben niedrigere Priorität.

8 Wirtschaftlichkeit

Das Spiel sollte eventuell kommerziellen Nutzen davontragen, Kosten wird es uns nichts.

9 Konsequenzen

- Auswirkungen (organisatorisch, personell, baulich, Vorschriften/Weisungen):
 - Bei Nichtrealisierung des Projekts wird das Projekt sehr wahrscheinlich so schlecht wie möglich bewertet
 - Verspätete Realisierung bewirkt schlechte Noten
- Auf Schnittstellen zu anderen Systemen:
 - Keine
- Qualitätsverbesserungen:
 - Es wird sicher ein grosser Lerneffekt vorhanden sein
- Risikobeurteilung:
 - Es wird bestimmt ein Risiko vorhanden sein, dass das Projekt scheitert, doch meistens geschieht dies gleich ziemlich am Anfang.
- Ausweichmöglichkeiten:
 - Wenn das Projekt offensichtlich am scheitern ist, kann man eventuell ein „Alternativprojekt“ starten

10 Antrag

Antrag auf Freigabe der nächsten Projektphase

11 Abbildungsverzeichnis

Abbildung 1 - Ordnerhierarchie unserer Applikation	5
Abbildung 2 - Klassendiagramm des Pakets MrIndustry.....	6
Abbildung 3 - Klassendiagramm des Pakets SpielObjekte	7
Abbildung 4 - Klassendiagramm des Pakets LeistungsSteigerungsObjekt	8
Abbildung 5 - Klassendiagramm des Pakets PrivatesObjekt	8
Abbildung 6 - Klassendiagramm des Pakets Rohstoffgewinnu	8
Abbildung 7 - Klassendiagramm des Pakets Verarbeitung	8
Abbildung 8 - Das Spiel zeigt die generierte Karte - Das Zentrallager wurde bereits gebaut.....	14
Abbildung 9 - Anzeige der Rohstoffe.....	15
Abbildung 10 - Das Baumenü der Kategorie Gewinnung.....	15
Abbildung 11 - Das Gebäude kann hier nicht errichtet werden	15
Abbildung 12 - Das Gebäude kann errichtet werden	15
Abbildung 13 - Dekonstruktiver Modus.....	16
Abbildung 14 - Die Spielkarte auf kleinster Zoom-Stufe.....	16

12 Anhang

Der Quellcode für die Inline-Dokumentation befindet sich hier:

MrIndustry.java

```
package mrIndustry;

import java.awt.BorderLayout;
import java.awt.CheckboxMenuItem;
import java.awt.Color;
import java.awt.Component;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.Menu;
import java.awt.MenuItem;
import java.awt.PopupMenu;
import java.awt.ScrollPane;
import java.awt.ScrollPaneAdjustable;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.AdjustmentEvent;
import java.awt.event.AdjustmentListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseWheelEvent;
import java.awt.event.MouseWheelListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.BufferedImage;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import javax.swing.AbstractButton;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.border.Border;

import mrIndustry.SpielObjekte.Umgebung;
import mrIndustry.SpielObjekte.ZentralLager;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Bergwerk;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Eisenmine;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Farm;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Försterei;
```

```
import mrIndustry.SpielObjekte.Verarbeitungsgebaude.Baeckerei;
import mrIndustry.SpielObjekte.Verarbeitungsgebaude.Werkzeugfabrik;
import mrIndustry.SpielObjekte.LeistungsSteigerung.Autofabrik;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausMittelKlasse;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausOberKlasse;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausUnterKlasse;

public class MrIndustry extends JPanel implements KeyListener, ActionListener,
    MouseMotionListener, MouseWheelListener, MouseListener, Runnable {
    //Wird benötigt um die Grösse eines einzelnen Kartenfelds zu bestimmen
    (hauptsächlich
    //nötig für Zoom und Scrolling):
    public static int textureSize = 50;

    //Hauptfenster:
    static JFrame wnd;

    //Beinhalten die aktuellen Koordinaten des Cursors im Kartenraster, also
    nicht in absoluten Angaben:
    static int xVal = 0;
    static int yVal = 0;

    //Beinhalten die absoluten Mauskoordinaten:
    //Mit Abzug der Abweichungen durch Formelemente:
    static int corX = 0;
    static int corY = 0;
    //Ohne:
    public int mouseX;
    public int mouseY;
    //Spielkarte:
    public Spielkarte map;

    public Graphics backBuffer;
    public Image frontBuffer;

    //Gebäudekategoriewahl:
    public JPanel actionBar;
    public JLabel baumenu;
    public JButton btn_Gewinnung;
    public JButton btn_Verarbeitung;
    public JButton btn_Dienstleistung;
    public JButton btn_Wohngebäude;
    public JButton btn_Logistik;
    public JButton btn_Sonstiges;

    //Kontextmenü und seine Elemente:
    public PopupMenu pmnu;
    public MenuItem mnuItem3;
    public MenuItem mnuItem2;
    public MenuItem mnuItem;
    public MenuItem smallItem;
    public MenuItem mediumItem;
    //Leiste mit dem Karte zentrieren Button & der Konsole:
    public JPanel actionBarNorth;
    protected JTextField consoleField = new JTextField();
    public JButton mapzentrieren;
    //Ressourcenleiste:
    public JPanel actionBarSouth;
    public JLabel lblbevoelkerung;
    public JLabel lbleisen;
```

```

public JLabel lblresourceFactor;
public JLabel lblkohle;
public JLabel lblholz;
public JLabel lblnahrung;

//Bestimmt ob die Karte gerade bewegt wird oder nicht:
protected boolean moveWithMouse;

public BauMenue bauMenue;

//Hauptmenüelemente:
private JPanel menufeld;
private JLabel title;
private JButton start;
private JLabel space;
private JButton reset;
private JLabel lblMapSize;
private JTextField MapSize;
<<<<<<< .mine

//Wird benötigt für rndZahl:
=====
Thread t;
private boolean run;

private JButton close;

private ImageIcon holzicon;

private ImageIcon foodicon;

private ImageIcon ironicon;

private ImageIcon kohleicon;

private ImageIcon resourceFactorIcon;

private ImageIcon humanicon;
>>>>>>> .r77
static java.util.Random random = new java.util.Random();

public static void main(String[] args) {
    //Startet das Spiel:
    wnd = new JFrame("MrIndustry");

    wnd.setContentPane(new MrIndustry());
}

public MrIndustry() {
    this.mouseX = 0;
    this.mouseY = 0;
    //Hinzufügen der Listeners:
    wnd.addMouseMotionListener(this);
    wnd.addMouseWheelListener(this);
    consoleField.addKeyListener(this);
    wnd.addMouseListener(this);
}

```

```

        wnd.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                System.out.println(event);
                System.exit(0);
            }
        });
    };
<<<<<<< .mine
    //Generierung einer Spielkarte mit 30x30 Feldern:
    map = new Spielkarte(30, 30);

    setLayout(new BorderLayout());
    map.newGame();

    wnd.setSize(500, 500);
=====
>>>>>>> .r77

    actionBar = new JPanel(new GridLayout(7, 1));

    actionBarSouth = new JPanel(new GridLayout(1, 5));
    actionBarNorth = new JPanel(new GridLayout(1, 2));
    baumenu = new JLabel("<HTML><BODY><H3>Baumenü:");
    mapzentrieren = new JButton("Karte Zentrieren");

    holzicon = new ImageIcon("holzicon.png", "Holz");
    humanicon = new ImageIcon("humanicon.png", "Bevoelkerung");
    foodicon = new ImageIcon("foodicon.png", "Nahrung");
    ironicon = new ImageIcon("ironicon.png", "Eisen");
    kohleicon = new ImageIcon("kohleicon.png", "Kohle");
    resourceFactorIcon = new ImageIcon("resourceFactorIcon.png",
        "ResoureFactor");
    lblholz = new JLabel(holzicon);
    lblbevoelkerung = new JLabel(humanicon);
    lblleisen = new JLabel(ironicon);
    lblkohle = new JLabel(kohleicon);
    lblnahrung = new JLabel(foodicon);
    lblresourceFactor = new JLabel(resourceFactorIcon);

    btn_Gewinnung = new JButton("Gewinnung");
    btn_Verarbeitung = new JButton("Verarbeitung");
    btn_Dienstleistung = new JButton("Dienstleistung");
    btn_Wohngebäude = new JButton("Wohngebäude");
    btn_Logistik = new JButton("Logistik");
    btn_Sonstiges = new JButton("Sonstiges");
    btn_Gewinnung.addMouseListener(this);
    btn_Verarbeitung.addMouseListener(this);
    btn_Dienstleistung.addMouseListener(this);
    btn_Wohngebäude.addMouseListener(this);
    btn_Logistik.addMouseListener(this);
    btn_Sonstiges.addMouseListener(this);
    mapzentrieren.addMouseListener(this);

    //Hinzufügen des KeyListeners zum Fenster wnd, speziell: der Fokus
    muss noch auf das Fenster
    //gesetzt werden:
    wnd.addKeyListener(this);
    wnd.setFocusable(true);
    wnd.requestFocus();

    //Kontextmenü:
    pmnu = new PopupMenu();

```



```

mnuItem = new JMenuItem("Gebäude Zerstören");
mnuItem.addActionListener(this);
pmnu.add(mnuItem);

mnuItem3 = new JMenuItem("Gebäude Deaktivieren");
mnuItem3.addActionListener(this);
pmnu.add(mnuItem3);

pmnu.addSeparator();

mnuItem2 = new JMenuItem("Beenden");
mnuItem2.addActionListener(this);

pmnu.add(mnuItem2);

actionBarSouth.add(lblbevoelkerung);
actionBarSouth.add(lbleisen);
actionBarSouth.add(lblkohle);
actionBarSouth.add(lblholz);
actionBarSouth.add(lblnahrung);
actionBarSouth.add(lblresourceFactor);
actionBarNorth.add(mapzentrieren);
actionBar.add(baumenu);
actionBar.add(btn_Gewinnung);
actionBar.add(btn_Verarbeitung);
actionBar.add(btn_Dienstleistung);
actionBar.add(btn_Wohngebäude);
actionBar.add(btn_Sonstiges);
actionBar.add(btn_Logistik);

btn_Gewinnung.setBackground(new Color(204, 204, 0));
btn_Verarbeitung.setBackground(new Color(153, 102, 0));
btn_Dienstleistung.setBackground(new Color(102, 153, 255));
btn_Wohngebäude.setBackground(new Color(0, 153, 51));
btn_Sonstiges.setBackground(new Color(204, 204, 204));
btn_Logistik.setBackground(new Color(180, 60, 60));

wnd.setSize(800, 800);
wnd.setExtendedState(Frame.MAXIMIZED_BOTH);
wnd.setVisible(true);
<<<<<<< .mine

    this.run();

}

public void menu() {
    //Spielmenü:
    menufeld = new JPanel(new GridLayout(6, 6));
=====
    menufeld = new JPanel(new GridLayout(7, 7));
>>>>>>> .r77
    title = new JLabel("MrIndustry");
    start = new JButton("Start!");
    reset = new JButton("Reset!");
    close = new JButton("Beenden!");
    space = new JLabel();

```

```
        reset.setEnabled(false);
        start.addActionListener(this);
        reset.addActionListener(this);
        close.addActionListener(this);
        lblMapSize = new JLabel("Map Grösse");
        MapSize = new JTextField();
        MapSize.setText("30");

        title.setFont(new Font("Impact", 20, 20));
        menufeld.add(title);
        menufeld.add(space);
        menufeld.add(start);
        menufeld.add(reset);

        menufeld.add(lblMapSize);
        menufeld.add(MapSize);
        menufeld.add(close);

        this.menu();
        t = new Thread(this);
        wnd.setVisible(true);
    }

    public void startGame(int mapsizeX, int mapsizeY) {

        map = new Spielkarte(mapsizeX, mapsizeY);
        setLayout(new BorderLayout());
        map.newGame();
        map.add(pmnu);

        wnd.add(actionBar, BorderLayout.WEST);
        wnd.add(actionBarSouth, BorderLayout.SOUTH);
        wnd.add(actionBarNorth, BorderLayout.NORTH);
        wnd.add(map, BorderLayout.CENTER);
        wnd.setVisible(true);
    }

    public void menu() {

        this.add(menufeld);
        wnd.setVisible(true);
    }
    //Random-Zahl-Funktion:
    public static int rndZahl(int h) {
        return random.nextInt(h);
    }

    //Scrollbar-Listener, wird eigentlich nicht mehr benötigt, jedoch belassen
    wir diesen, eventuell
    //braucht das Spiel später Scrollbars:
    class MyAdjustmentListener implements AdjustmentListener {

        @Override
        public void adjustmentValueChanged(AdjustmentEvent e) {
            if (e.getAdjustable().getOrientation() == 0) {
                map.setHorizontal(e.getValue());
            } else {
                map.setVertical(e.getValue());
            }
        }
    }
```

```

    }
}

public void run() {
    //Endlosschleife des Spiels:
    while (true) {
<<<<<<<< .mine
        //Wenn das Zentrallager existiert werden die Ressourcen
angezeigt:
        if (map.zentrallager) {
            this.lblresourceFactor.setText("" + map.resourceFactor);
            this.lblholz.setText(""
                + ((Zentrallager) map.spielObjekte[map
                    .getZentrallagerX()][map.getZentrallagerY()])
                    .getHolz());
            this.lblleisen.setText(""
                + ((Zentrallager) map.spielObjekte[map
                    .getZentrallagerX()][map.getZentrallagerY()])
                    .getEisen());
            this.lblbevoelkerung
                .setText(""
=====
>>>>>>> .r77
<<<<<<<< .mine
                + ((Zentrallager)
map.spielObjekte[map
    .getZentrallagerX()][map
    .getZentrallagerY()]).bevoelkerungAktiv
                + "/"
                + ((Zentrallager)
map.spielObjekte[map
    .getZentrallagerX()][map
    .getZentrallagerY()]).bevoelkerungInaktiv
                + "/"
                + ((Zentrallager)
map.spielObjekte[map
    .getZentrallagerX()][map
    .getZentrallagerY()]).bevoelkerungSoll);
            this.lblkohle.setText(""
                + ((Zentrallager) map.spielObjekte[map
                    .getZentrallagerX()][map.getZentrallagerY()])
                    .getKohle());
            this.lblnahrung.setText(""
                + ((Zentrallager) map.spielObjekte[map
                    .getZentrallagerX()][map.getZentrallagerY()])
                    .getNahrung());
        }
        //Wenn ein Gebäude aktiv ist, wird ein Tick (Siehe
Realisierungsbericht für Definition) ausgeführt:

```

```

        for (int x = 0; x < map.spielObjekte.length; x++) {
            for (int y = 0; y < map.spielObjekte[x].length; y++) {
                if (map.spielObjekte[x][y].isActive == true) {
                    map.spielObjekte[x][y].tick();
                }
            }
        }

=====

        if (this.run) {
            if (map.zentralLager) {
                this.lblresourceFactor.setText(" " +
                    map.resourceFactor);
                this.lblholz
                    .setText(" "
                        + ((ZentralLager)
                            map.spielObjekte[map
                                .getZentralLagerX()][map
                                    .getZentralLagerY()]).getHolz());
                this.lblbleisen
                    .setText(" "
                        + ((ZentralLager)
                            map.spielObjekte[map
                                .getZentralLagerX()][map
                                    .getZentralLagerY()]).getEisen());
                this.lblbevoelkerung
                    .setText(" "
                        + ((ZentralLager)
                            map.spielObjekte[map
                                .getZentralLagerX()][map
                                    .getZentralLagerY()]).bevoelkerungAktiv
                                + "/"
                                + ((ZentralLager)
                                    map.spielObjekte[map
                                        .getZentralLagerX()][map
                                            .getZentralLagerY()]).bevoelkerungInaktiv
                                        + "/"
                                        + ((ZentralLager)
                                            map.spielObjekte[map
                                                .getZentralLagerX()][map
                                                    .getZentralLagerY()]).bevoelkerungSoll);
                this.lblkohle
                    .setText(" "
                        + ((ZentralLager)
                            map.spielObjekte[map
                                .getZentralLagerX()][map
                                    .getZentralLagerY()]).getKohle());
                this.lblnahrung
                    .setText(" "
                        + ((ZentralLager)
                            map.spielObjekte[map
                                .getZentralLagerX()][map
                                    .getZentralLagerY()]).getNahrung());
            }
        }
    }
}

```

```

        .getZentralLagerX()) [map
        .getZentralLagerY()) .getNahrung());
        }
        for (int x = 0; x < map.spielObjekte.length; x++) {
            for (int y = 0; y < map.spielObjekte[x].length;
y++) {
                if (map.spielObjekte[x][y].isActive == true)
                {
                    map.spielObjekte[x][y].tick();
                }
            }
        }
        wnd.repaint();
        try {
            Thread.sleep(150);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //ganzes Fenster wird gerendert:
<<<<<<< .mine
        wnd.repaint();

        try {
            Thread.sleep(150);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        =====
>>>>>>> .r77
    }

}

@Override
public void update(Graphics g) {
    super.update(g);
    // map.zeichneAlle(g);
}
//Zoom-Funktion des Spiels:
@Override
public void mouseWheelMoved(MouseWheelEvent e) {
    //Zoom-Limit für die Nähe:
    if ((this.textureSize - (e.getScrollAmount() *
e.getWheelRotation())) >= 300) {
        this.textureSize = 299;
    //Zoom-Limit für die Weite
    } else if ((this.textureSize - (e.getScrollAmount() * e
        .getWheelRotation())) <= 10) {
        textureSize = 11;
    } else {
        //ansonsten normale Größenänderung:
        this.textureSize = this.textureSize
            - (e.getScrollAmount() * e.getWheelRotation());
    }
}

```

```
switch (e.getWheelRotation()) {
case 0:
    map.setHorizontal(map.getHorizontal());
    map.setVertical(map.getVertical());

    break;

case 1:
    map.setHorizontal(map.getHorizontal());
    map.setVertical(map.getVertical());
    break;
}

}
//MouseClicked handler:
@Override
public void mouseClicked(MouseEvent e) {
    // TODO Auto-generated method stub
    String title = null;
    //Karte zentrieren nach klick auf Button::
    if (e.getSource() == mapzentrieren) {
        map.setHorizontal(((map.getMapsizeX() / 2) * textureSize)
            - wnd.getWidth() / 4);
        map.setVertical(((map.getMapsizeY() / 2) * textureSize)
            - wnd.getHeight() / 4);
        //Entsprechendes Baumenü öffnen nach Klick auf Button:
        //(Titel des Baumenüs wird gesetzt und der Gebäudekategorietyp
und die Karte werden mitgegeben:
    } else if (e.getSource() == btn_Gewinnung) {
        bauMenue = new BauMenue(map, 0);
        title = "Gewinnung";
        bauMenue.setTitle("Baumenü Kategorie " + title
            + " - Wähle dein Gebäude:");
        bauMenue.setVisible(true);

    } else if (e.getSource() == btn_Verarbeitung) {
        bauMenue = new BauMenue(map, 1);
        title = "Verarbeitung";
        bauMenue.setTitle("Baumenü Kategorie " + title
            + " - Wähle dein Gebäude:");
        bauMenue.setVisible(true);
    } else if (e.getSource() == btn_Dienstleistung) {
        bauMenue = new BauMenue(map, 2);
        title = "Dienstleistung";
        bauMenue.setTitle("Baumenü Kategorie " + title
            + " - Wähle dein Gebäude:");
        bauMenue.setVisible(true);
    } else if (e.getSource() == btn_Wohngebäude) {
        bauMenue = new BauMenue(map, 3);
        title = "Wohngebäude";
        bauMenue.setTitle("Baumenü Kategorie " + title
            + " - Wähle dein Gebäude:");
        bauMenue.setVisible(true);
    } else if (e.getSource() == btn_Sonstiges) {
        bauMenue = new BauMenue(map, 4);
        title = "Sonstiges";
        bauMenue.setTitle("Baumenü Kategorie " + title
            + " - Wähle dein Gebäude:");
        bauMenue.setVisible(true);
    } else if (e.getSource() == btn_Logistik) {
        bauMenue = new BauMenue(map, 5);
```

```

        title = "Logistik";
        bauMenue.setTitle("Baumenü Kategorie " + title
            + " - Wähle dein Gebäude:");
        bauMenue.setVisible(true);
    } else {
        //Baumodus:
        //Wenn das Gebäude nicht mit einem Hindernis konfrontiert wird
        if (!(xVal <= 0 && yVal <= 0)) {
            if (map.spielObjekte[xVal][yVal].getTileColor() == true)
{
                //Überprüfungsstring:
                if (map.newObject != null) {
                    String substractResult = "ok";
                    //Ressourcen werden abgezogen:
                    if (map.newObject instanceof Bergwerk) {
                        substractResult = subtractRessources(0,
0);
                    }
                    if (map.newObject instanceof Eisenmine) {
                        substractResult = subtractRessources(0,
1);
                    }
                    if (map.newObject instanceof Försterei) {
                        substractResult = subtractRessources(0,
2);
                    }
                    if (map.newObject instanceof Farm) {
                        substractResult = subtractRessources(0,
3);
                    }
                    if (map.newObject instanceof Autofabrik) {
                        substractResult = subtractRessources(1,
0);
                    }
                    if (map.newObject instanceof Werkzeugfabrik)
{
                        substractResult = subtractRessources(1,
1);
                    }
                    if (map.newObject instanceof
WohnhausUnterKlasse) {
                        substractResult = subtractRessources(3,
0);
                    }
                    if (map.newObject instanceof
WohnhausMittelKlasse) {
                        substractResult = subtractRessources(3,
1);
                    }
                    if (map.newObject instanceof
WohnhausOberKlasse) {
                        substractResult = subtractRessources(3,
2);
                    }
                    if (map.newObject instanceof Baeckerei) {
                        substractResult = subtractRessources(1,
2);
                    }
                }
                /*
                * if (map.newObject instanceof Eisenmine) {
                *
                */

```



```

        * }
        */

<<<<<<< .mine
gebaut werden,
Rohstoff es mangelt:
=====
>>>>>>> .r77

        if (substractResult == "ok") {
            map.newObject.setDragDrop(false);
            map.newObject.setLocX(xVal);
            map.newObject.setLocY(yVal);
            map.newObject.activate(true);
            map.spielObjekte[xVal][yVal] =

map.newObject;

            map.spielObjekte[xVal][yVal].doAfterBuild();

        } else {
            JOptionPane.showMessageDialog(wnd,
                "Nicht genügend " +
substractResult);
        }
        //Beim Bau des Zentrallagers wird zusätzlich
noch die Position des Zentrallagers
        //an die Karte weitergeleitet (damit das
Spiel weiss an welcher Position das Zentrallager
        //im Vektor-Array ist):
        if (map.newObject instanceof Zentrallager) {
            map.setZentrallagerX(xVal);
            map.setZentrallagerY(yVal);
            map.zentrallager = true;
        }
        map.newObject = null;
        System.out.println("possible");
    }

    } else {
        //Wenn der User das Gebäude auf ein Hindernis
bauen will, wird dies verhindert:
        if (!(map.newObject instanceof Zentrallager)) {
            map.newObject = null;
        }
        System.out.println("not possible");
    }

    }

    }

}

//Rohstoffe abziehen vom Zentrallager:
public String subtractRessources(int ktype, int btype) {
    //immer noch ok:
    String error = "ok";
    System.out
        .println(((Zentrallager) map.spielObjekte[map
        .getZentrallagerX()][map.getZentrallagerY()]).bevoelkerungInaktiv
        - SpielObjekt.kosten[ktype][btype][3]);

```

```

        System.out.println(SpielObjekt.kosten[ktype][btype][3]);
        //Vorherige Überprüfung:
        //Wenn von einem Rohstoff nach dem Bau weniger als 0 übrig wäre,
wird der Bau verhindert,
        //indem der "error"-String von "ok" auf die Art der Ressource
gewechselt wird von dem der Rohstoff mangelt
        //somit wird gleich noch zurückgegeben an welchem Rohstoff es
mangelt:
        if (((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
            .getZentralLagerY()]).nahrung
            - SpielObjekt.kosten[ktype][btype][0] < 0) {
            error = "Nahrung";
        }

        if (((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
            .getZentralLagerY()]).kohle
            - SpielObjekt.kosten[ktype][btype][1] < 0) {
            error = "Kohle";
        }

        if (((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
            .getZentralLagerY()]).eisen
            - SpielObjekt.kosten[ktype][btype][2] < 0) {
            error = "Eisen";
        }

        if (((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
            .getZentralLagerY()]).bevoelkerungInaktiv
            - SpielObjekt.kosten[ktype][btype][3] < 0

        ) {
            error = "Bevölkerung";
        }

        if (((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
            .getZentralLagerY()]).holz
            - SpielObjekt.kosten[ktype][btype][4] < 0) {

            error = "Holz";
        }
        //Effektive Subtraktion der Rohstoffe:
        if (error == "ok") {
            ((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
                .getZentralLagerY()]).nahrung -=
SpielObjekt.kosten[ktype][btype][0];
            ((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
                .getZentralLagerY()]).kohle -=
SpielObjekt.kosten[ktype][btype][1];
            ((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
                .getZentralLagerY()]).eisen -=
SpielObjekt.kosten[ktype][btype][2];
            ((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
                .getZentralLagerY()]).bevoelkerungInaktiv -=
SpielObjekt.kosten[ktype][btype][3];
            ((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
                .getZentralLagerY()]).bevoelkerungAktiv +=
SpielObjekt.kosten[ktype][btype][3];
            ((ZentralLager) map.spielObjekte[map.getZentralLagerX()][map
                .getZentralLagerY()]).holz -=
SpielObjekt.kosten[ktype][btype][4];

        }
        System.out.println(error);
        return error;

```

```
}

@Override
public void mouseEntered(MouseEvent e) {
    // TODO Auto-generated method stub
}

@Override
public void mouseExited(MouseEvent e) {
    // TODO Auto-generated method stub
}

//Wenn die Maus gedrückt gehalten wird, kann die Karte verschoben werden:
@Override
public void mousePressed(MouseEvent e) {
    if (e.getButton() == 2 || e.getButton() == 1) {
        this.moveWithMouse = true;
    }
}

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == close) {
        int n = JOptionPane.showConfirmDialog(wnd, "Sind Sie sicher?",
            "Beenden", JOptionPane.YES_NO_OPTION);
        if (n == 0) {
            System.exit(0);
        }
    }
    if (e.getSource() == reset) {
        this.MapSize.setBackground(Color.WHITE);
        wnd.getContentPane().removeAll();
        wnd.validate();
        wnd.repaint();
        this.startGame(Integer.parseInt(this.MapSize.getText()),
            Integer.parseInt(this.MapSize.getText()));
        this.run = true;
        wnd.setVisible(true);
    }
    if (e.getSource() == start) {
        //Bearbeitung der Eingaben im Kontextmenü:
        boolean err = false;
        try {
            Integer.parseInt(this.MapSize.getText());
        } catch (NumberFormatException n) {
            err = true;
        }
        if (!err) {

            this.MapSize.setBackground(Color.WHITE);
            wnd.getContentPane().removeAll();
            wnd.validate();
            wnd.repaint();
            this.startGame(Integer.parseInt(this.MapSize.getText()),
                Integer.parseInt(this.MapSize.getText()));
            this.run = true;
            wnd.setVisible(true);
        }
    }
}
```

```

        this.t.start();
    } else {
        this.MapSize.setBackground(Color.RED);
    }
} else if (map != null) {
    Object o = e.getSource();
    if (((MenuItem) o).getLabel() == "Gebäude Zerstören") {

        map.spielObjekte[corX][corY].destroy();
        map.spielObjekte[corX][corY] = new Umgebung(
            map.map_array[corX][corY], corX, corY);
        // map.spielObjekte[corX][corY].activate(false);
        //
        map.spielObjekte[corX][corY].activate(!map.spielObjekte[corX][corY]
            // .isActive());

    }

    if (((MenuItem) o).getLabel() == "Gebäude Aktivieren") {
        map.spielObjekte[corX][corY].activate(true);
    }

    if (((MenuItem) o).getLabel() == "Gebäude Deaktivieren") {
        map.spielObjekte[corX][corY].activate(false);
    }
    if (((MenuItem) o).getLabel() == "Beenden") {
        System.exit(0);
    }
}

@Override
public void mouseReleased(MouseEvent e) {
    //Öffnen des Kontextmenüs, mit den Korrekten Angaben zum Gebäude:
    if (e.isPopupTrigger()) {

        if (e.getButton() == 3) {
            corX = xVal;
            corY = yVal;
            if (map.spielObjekte[corX][corY] instanceof Umgebung
                || map.spielObjekte[corX][corY] instanceof
ZentralLager) {

                mnuItem3.enable(false);
                mnuItem.enable(false);
            } else {

                System.out.println(map.spielObjekte[corX][corY].isActive());
                mnuItem3.enable(true);
                mnuItem.enable(true);
                if (map.spielObjekte[corX][corY].isActive() ==
true) {

                    mnuItem3.setLabel("Gebäude Deaktivieren");
                } else {
                    mnuItem3.setLabel("Gebäude Aktivieren");
                }
            }
            pmnu.show(map, e.getX() - actionBar.getWidth() - 8,
e.getY()
                - actionBarNorth.getHeight() - 30);
        }
    }
}

```

```

    }

    if (e.getButton() == 2 || e.getButton() == 1) {
        this.moveWithMouse = false;
    }
}
//Wenn die Maus gedrückt gehalten wird, kann die Karte bewegt werden,
ansonsten nicht..
@Override
public void mouseDragged(MouseEvent e) {

    if (this.moveWithMouse) {
        map.setHorizontal(map.getHorizontal() + (1 * (mouseX -
e.getX())));

        map.setVertical(map.getVertical() + (1 * (mouseY -
e.getY())));

    }
    mouseX = e.getX();
    mouseY = e.getY();
    wnd.setVisible(true);
}

@Override
public void mouseMoved(MouseEvent e) {
    //Absolute Mausposition:
    mouseX = e.getX();
    mouseY = e.getY();
<<<<<<< .mine

    if (map.newObject != null) {
=====
        if (map != null) {
            if (map.newObject != null) {
>>>>>>> .r77
                //Objekt im Baumodus folgt so dem Mauszeiger:
<<<<<<< .mine
                map.newObject.setLocX(xVal * textureSize -
map.getHorizontal());
                map.newObject.setLocY(yVal * textureSize - map.getVertical());

            }

            //Berechnung der Position im Raster (für Zugriff auf Vektor-Array):
=====
                map.newObject.setLocX(xVal * textureSize -
map.getHorizontal());
                map.newObject.setLocY(yVal * textureSize -
map.getVertical());
                // map.newObject.setLocX(mouseX-actionBar.getWidth()-8);
                // map.newObject.setLocY(mouseY-
actionBarNorth.getHeight()-textureSize);
            }
            // for (int a = 0; a < map.getSpielObjekte().length; a++) {
            // for (int b = 0; b < map.getSpielObjekte().length; b++) {
            // if (map.SpielObjekte[a][b].getDragDrop() == true) {
            // map.SpielObjekte[a][b].setLocX(mouseX);
            // map.SpielObjekte[a][b].setLocY(mouseY);
            // }

```

```

// }
// }

>>>>>>> .r77
<<<<<<< .mine
    xVal = ((map.getHorizontal() + e.getX() - actionBar.getWidth()) - 8)
            / textureSize;
    yVal = ((map.getVertical() + e.getY()) -
(actionBarNorth.getHeight()) - 30)
            / textureSize;
    //Effektive Kartengrösse:
    int sizeX = (map.getMapsizeX()) * textureSize;
    int sizeY = (map.getMapsizeY()) * textureSize;
=====
    xVal = ((map.getHorizontal() + e.getX() -
actionBar.getWidth()) - 8)
            / textureSize;
    yVal = ((map.getVertical() + e.getY())
            - (actionBarNorth.getHeight()) - 30)
            / textureSize;
    int sizeX = (map.getMapsizeX()) * textureSize;
    int sizeY = (map.getMapsizeY()) * textureSize;
>>>>>>> .r77
    //Wenn der Cursor nicht ausserhalb der Karte ist
<<<<<<< .mine
    if ((e.getX()) < sizeX && (e.getY()) < sizeY) {
        //Wird die Markierung aller Kartenfelder zuerst deaktiviert,
ansonsten wäre das Spiel
        //noch ein Zeichenprogramm...:
        for (int x = 0; x < map.getSpielObjekte().length; x++) {
            for (int y = 0; y < map.getSpielObjekte().length; y++) {
=====
                if ((e.getX()) < sizeX && (e.getY()) < sizeY) {
                    for (int x = 0; x < map.getSpielObjekte().length; x++) {
                        for (int y = 0; y < map.getSpielObjekte().length;
y++) {
>>>>>>> .r77

                            map.spielObjekte[x][y].setHighlight(false);
                            map.spielObjekte[x][y].setTileColor(false);
                        }
                    }
                }
                //Das Kartenfeld wo sich der Cursor befindet wird
herausgehoben mit einem farbigen Viereck:
                if (xVal < map.getMapsizeY() && xVal >= 0
                    && yVal < map.getMapsizeX() && yVal >= 0) {

                    map.spielObjekte[xVal][yVal].setHighlight(true);
                    //Die Farbe des Viereck (für die Kollisionserkennung
beim Bau)

                    //wird bestimmen:
                    if (map.newObject instanceof Eisenmine
                        || map.newObject instanceof Bergwerk
                        || map.newObject instanceof Försterei)
{
                        if (map.newObject instanceof Eisenmine
                            && map.getMapArray()[xVal][yVal]
== 27) {

                            map.spielObjekte[xVal][yVal].setTileColor(true);
                        }
                    }
                }
            }
        }
    }
}

```



```

        if (map.getMapArray()[xVal][yVal] == 28
            && map.newObject instanceof
Bergwerk) {

    map.spielObjekte[xVal][yVal].setTileColor(true);
    }
    if (map.getMapArray()[xVal][yVal] == 2
        && map.newObject instanceof
Försterei) {

    map.spielObjekte[xVal][yVal].setTileColor(true);
    }
    } else if (map.getMapArray()[xVal][yVal] == 1) {

    map.spielObjekte[xVal][yVal].setTileColor(true);
    }

    }
    }

}
//Zeigt die Konsole an oder nicht:
public boolean console = false;

//Öffnen der Entwickler-Konsole (dort kann gecheatet werden):
public void keyPressed(KeyEvent e) {
    System.out.println(e.getKeyCode());
    if (this.map != null) {
        if (e.getKeyCode() == 27) {
            if (this.run) {
                this.run = false;
                setLayout(new FlowLayout());
                wnd.getContentPane().removeAll();
                wnd.validate();
                wnd.repaint();
                this.menu();
                reset.setEnabled(true);
                start.setEnabled(false);
            } else {
                this.run = true;
                wnd.getContentPane().removeAll();
                wnd.validate();

                setLayout(new BorderLayout());
                map.add(pmnu);

                wnd.add(actionBar, BorderLayout.WEST);
                wnd.add(actionBarSouth, BorderLayout.SOUTH);
                wnd.add(actionBarNorth, BorderLayout.NORTH);
                wnd.add(map, BorderLayout.CENTER);
                wnd.setVisible(true);
            }
        }
    }
    if (e.getKeyChar() == '$') {
        consoleField.setBackground(Color.GRAY);
        if (this.console) {
            this.console = false;

```

```

        actionBarNorth.removeAll();
        actionBarNorth.add(mapzentrieren);
        wnd.requestFocusInWindow();
    } else {
        this.console = true;
        actionBarNorth.removeAll();
        actionBarNorth.add(consoleField);
        consoleField.requestFocusInWindow();
    }
    wnd.setVisible(true);
}
//Beliebige Anzahl an Ressourcen kann eingegeben werden:
if (e.getSource() == consoleField) {
    if (e.getKeyCode() == 10) {
        if (consoleField.getText() != "") {
            String text = consoleField.getText();
            String var = text.substring(0, text.indexOf('='));
            String value = text.substring(text.indexOf('=') +
1,
                                text.length());
            this.console = false;
            actionBarNorth.removeAll();
            actionBarNorth.add(mapzentrieren);
            wnd.requestFocusInWindow();
            consoleField.setText("");
            if (var.equals("holz")) {
                map.getZentralLager().holz =
Integer.parseInt(value);
            }
            if (var.equals("eisen")) {
                map.getZentralLager().eisen =
Integer.parseInt(value);
            }
            if (var.equals("kohle")) {
                map.getZentralLager().kohle =
Integer.parseInt(value);
            }
            if (var.equals("bevoelkerung")) {
                map.getZentralLager().bevoelkerungSoll =
Integer
                                .parseInt(value);
                map.getZentralLager().bevoelkerungInaktiv =
Integer
                                .parseInt(value);
                map.getZentralLager().bevoelkerungAktiv;
            }
            if (var.equals("nahrung")) {
                map.getZentralLager().nahrung =
Integer.parseInt(value);
            }
        }
    }
}

@Override

```

```
public void keyReleased(KeyEvent e) {  
    // TODO Auto-generated method stub  
  
}  
  
@Override  
public void keyTyped(KeyEvent e) {  
    // TODO Auto-generated method stub  
  
}  
  
}
```

Baumenü.java

```
package mrIndustry;  
  
import java.awt.BorderLayout;  
import java.awt.Color;  
import java.awt.Cursor;  
import java.awt.FlowLayout;  
import java.awt.Graphics;  
import java.awt.GridLayout;  
import java.awt.Image;  
import java.awt.Point;  
import java.awt.Toolkit;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import java.io.IOException;  
  
import javax.imageio.ImageIO;  
import javax.swing.BoxLayout;  
  
import javax.swing.DefaultComboBoxModel;  
import javax.swing.Icon;  
import javax.swing.ImageIcon;  
import javax.swing.JButton;  
import javax.swing.JLabel;  
import javax.swing.JList;  
import javax.swing.JPanel;  
import javax.swing.JSplitPane;  
import javax.swing.JTabbedPane;  
import javax.swing.JTextPane;  
import javax.swing.ListModel;  
import javax.swing.SwingConstants;  
  
import javax.swing.WindowConstants;  
import javax.swing.SwingUtilities;  
import javax.swing.event.ChangeEvent;  
import javax.swing.event.ChangeListener;  
import javax.swing.event.ListSelectionEvent;  
import javax.swing.event.ListSelectionListener;  
  
import mrIndustry.MrIndustry;  
import mrIndustry.SpielObjekte.LeistungsSteigerung.Autofabrik;  
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausMittelKlasse;  
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausOberKlasse;  
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausUnterKlasse;  
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Bergwerk;  
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Eisenmine;  
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Farm;
```

```
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Foersterei;
import mrIndustry.SpielObjekte.Verarbeitungsgebaude.Baeckerei;
import mrIndustry.SpielObjekte.Verarbeitungsgebaude.Werkzeugfabrik;

public class BauMenue extends javax.swing.JFrame {

    private static final long serialVersionUID = 1L;

    //Form-Elemente des Baumenüs:

    //Trennt die Liste von der Beschreibung (Trennbalken):
    private JSplitPane jSplitPanel;
    //Zeigt das Bild des Gebäudes an:
    private JLabel lbl_img;
    private JPanel panel_image;
    //Baut das Gebäude:
    private JButton btn_bauen;
    //Layout Elemente:
    private JPanel jPanel3;
    private JPanel jPanel1;
    private JPanel jPanel2;
    //Zeigt die Kosten eines Gebäudes an:
    private JLabel lbl_kosten;
    //Zeigt die Beschreibung eines Gebäudes an:
    private JLabel text_desc;
    //Panel für die Beschreibung:
    private JPanel panel_desc;
    //Zeigt die Liste der verfügbaren Gebäude der Kategorie:
    private JList List_Gew;

    public BauMenue(final Spielkarte map, final int ktype) {
        super();
        try {
            setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
            getContentPane().setLayout(null);
            {
                jPanel3 = new JPanel();
                getContentPane().add(jPanel3, "Center");
                jPanel3.setBounds(0, 0, 492, 363);
                {
                    //Je nachdem welche Kategorie gewählt wurde, wird
das Baumenü
//
in anderer Farbe dargestellt:
switch (ktype) {
case 0:
    jPanel3.setBackground(new Color(204, 204,
0));
break;
case 1:
    jPanel3.setBackground(new Color(153, 102,
0));
break;
case 2:
    jPanel3.setBackground(new Color(102, 153,
255));
break;
case 3:
    jPanel3.setBackground(new Color(0, 153,
51));

```

```

        break;
    case 4:
        jPanel3.setBackground(new Color(204, 204,
204));

        break;
    case 5:
        jPanel3.setBackground(new Color(180, 60,
60));

        break;

    }
    jSplitPanel = new JSplitPane();
    jPanel3.add(jSplitPanel);
    jSplitPanel.setBounds(12, 12, 468, 339);
    jSplitPanel.setPreferredSize(new
java.awt.Dimension(468,
                    352));
    {
        jPanel2 = new JPanel();
        GridLayout jPanel2Layout = new GridLayout(1,
1);

        jPanel2Layout.setHgap(5);
        jPanel2Layout.setVgap(5);
        jPanel2Layout.setColumns(1);

        jPanel2.setLayout(jPanel2Layout);
        jSplitPanel.add(jPanel2, JSplitPane.LEFT);

        // jPanel2.setLayout(jPanel2Layout);
        jPanel2.setPreferredSize(new
java.awt.Dimension(163,
                    298));
        {
            panel_image = new JPanel();
            BorderLayout panel_imageLayout = new
BorderLayout();

            panel_image.setLayout(panel_imageLayout);
            jPanel2.add(panel_image);
            panel_image.setBounds(0, 128, 144,
132);

            {
                String list_text[][] =

                ListModel List_GewModel = new

                    list_text[ktype]);
                List_Gew = new JList();
                panel_image.add(List_Gew,

                List_Gew.setModel(List_GewModel);
                List_Gew.setBounds(0, 0, 105,

                List_Gew.setPreferredSize(new

                    163, 82));

                List_Gew.addListSelectionListener(new ListSelectionListener() {
                    public void valueChanged(

                    ListSelectionEvent arg0) {

```

```

gesetzt, je nachdem welche Gebäudetyp
//
Texte sind im Array arr_desc in der Klasse
//
    SpielObjekte gespeichert)

kosten[][][x]: Geld:0
Eisen:3 Arbeiter:4
kosten[][x][]:

,Försterei,Farm,Autofabrik,Werkzeugfabrik

Wohngeb.unter.,Wohngeb

.ober.,Statue,Park,Strasse Index von

Gewinnung,Verarbeitung,
Dienstleistung,private
Gebäude,Sonstiges,Logistik

(List_Gew.getSelectedIndex()) {

    SpielObjekt.arr_desc[ktype][0],

ktype);

    SpielObjekt.arr_desc[ktype][1],

ktype);

    SpielObjekt.arr_desc[ktype][2],

ktype);

    SpielObjekt.arr_desc[ktype][3],

ktype);

```

```

// Die Texte werden
ausgewählt wurde (Die

/*
* Index von
* Nahrung:1 Kohle:2
* Holz:5 Index von
* Eisenmine
*
* ,Markt,Bar,Theater
*
* .mittel.,Wohngeb
*
* kosten[x][[]]:
*
*
*
*/
switch
case 0:
    setTheText(

                                0,

        break;
case 1:
    setTheText(

                                1,

        break;
case 2:
    setTheText(

                                2,

        break;
case 3:
    setTheText(

                                3,

```

```
(SpielObjekt.kosten[ktype][List_Gew
    .getSelectedIndex()][5]) {
    lbl_img wird gesetzt, mithilfe der Funktion setImg:

    setImg("bergwerk.png");

    setImg("eisenmine.png");

    setImg("föresterei.png");

    setImg("farm.png");

    setImg("autofabrik.png");

    setImg("werkzeugfabrik.png");

    setImg("baeckerei.png");

    setImg("houseUnterKlasse.png");

    setImg("houseMittelKlasse.png");

    setImg("house.png");

    setImg("house.png");

    btn_bauen.setEnabled(true);

    break;
}
switch
//Das Bild für das
case 0:

    break;
case 1:

    break;
case 2:

    break;
case 3:

    break;
case 4:

    break;
case 5:

    break;
case 6:

    break;
case 9:

    break;
case 10:

    break;
case 11:

    break;
default:

    break;
}

repaint();
```



```

generated method stub

filename) {

    ImageIcon(filename);
    icon.getImage();
    new BufferedImage(

        BufferedImage.TYPE_INT_ARGB);
    bi.createGraphics();
    0, 50, 50, null);

    lbl_img.setIcon(icon);

    setTheText(String desc,
    ktype) {

        SpielObjekt.kosten;

        .setText("<HTML><BODY><DIV>"
        + desc
        + "</DIV></BODY></HTML>");

        lbl_kosten.setText("<HTML><BODY><DIV>"

        "<br><p>Kosten:</p>"
        "<br>Nahrung:"
        kosten[ktype][btype][0]
        "<br>Kohle:"
        kosten[ktype][btype][1]
        "<br>Eisen:"
        kosten[ktype][btype][2]
        "<br>Arbeiter/Bevölkerung:"
    }

    // TODO Auto-
}

public void setImg(String

    ImageIcon icon = new
    Image img =
    BufferedImage bi =
        100, 100,

    Graphics g =
    g.drawImage(img, 0,

}

public void

    int btype, int

    int kosten[][][] =

    text_desc

+
+
+
+
+
+
+
+
+
+

```

```

kosten[ktype][btype][3]
"<br/>Holz:"
kosten[ktype][btype][4]
"</DIV></BODY></HTML>");
    }
    });
}
{
    JPanel1 = new JPanel();
    BorderLayout jPanel1Layout = new
        BorderLayout();
    jPanel1.setLayout(jPanel1Layout);
    panel_image.add(jPanel1,
        jPanel1.setPreferredSize(new
            Dimension(
                163, 260));
        {
            lbl_img = new JLabel();
            jPanel1.add(lbl_img,
                BorderLayout.SOUTH);
        }
    }
}
{
    panel_desc = new JPanel();
    BorderLayout panel_descLayout = new
        BorderLayout(
            javax.swing.BoxLayout.Y_AXIS);
    JSplitPane.add(panel_desc,
        panel_desc.setPreferredSize(new
            Dimension(296,
                303));
    panel_desc.setLayout(panel_descLayout);
    {
        text_desc = new JLabel();
        panel_desc.add(text_desc);
        text_desc
            .setText("<HTML><BODY><DIV>Beschreibung</DIV></BODY></HTML>");
        text_desc.setVerticalAlignment(JLabel.TOP);
    }
    {
        lbl_kosten = new JLabel();
        panel_desc.add(lbl_kosten);
        lbl_kosten.setText("Kosten");
    }
    {
        btn_bauen = new JButton();
        btn_bauen.setEnabled(false);
        panel_desc.add(btn_bauen);
    }
}

```

```

        btn_bauen.setText("<HTML><BODY><H3>Auswahl Bauen");
        btn_bauen.addActionListener(new
ActionListener() {
                                public void
actionPerformed(ActionEvent arg0) {

                                // Nach dem Klick auf den
                                btn_bauen wird das entsprechende in dem Menü gewählte Objekt auf der
                                //Karte hinzugefügt:
                                switch
(SpielObjekt.kosten[ktype][List_Gew
                                .getSelectedIndex()][5]) {

                                case 0:
                                    map.setNewObject(new
Bergwerk(
                                MrIndustry.xVal,
                                MrIndustry.yVal, true, map));

                                break;
                                case 1:
                                    map.setNewObject(new
Eisenmine(
                                MrIndustry.xVal,
                                MrIndustry.yVal, true, map));

                                break;
                                case 2:
                                    map.setNewObject(new
Försterei(
                                MrIndustry.xVal,
                                MrIndustry.yVal, true, map));

                                break;
                                case 3:
                                    map.setNewObject(new
Farm(
                                MrIndustry.xVal,
                                MrIndustry.yVal, true, map));

                                break;
                                case 4:
                                    map.setNewObject(new
Autofabrik(
                                MrIndustry.xVal,
                                MrIndustry.yVal, true, map));

                                break;
                                case 5:
                                    map.setNewObject(new
Werkzeugfabrik(
                                MrIndustry.xVal,

```

Speicherdatum: ##### Seite 53 von 84

Spielkarte.java

```
package mrIndustry;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Component;
import java.awt.GridLayout;
import java.awt.Image;
import java.util.Arrays;

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import mrIndustry.SpielObjekte.Umgebung;
import mrIndustry.SpielObjekte.ZentralLager;
import mrIndustry.SpielObjekte.LeistungsSteigerung.Autofabrik;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausMittelKlasse;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausOberKlasse;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausUnterKlasse;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Bergwerk;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Eisenmine;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Farm;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Försterei;
import mrIndustry.SpielObjekte.Verarbeitungsgebaude.Baeckerei;
import mrIndustry.SpielObjekte.Verarbeitungsgebaude.Werkzeugfabrik;

public class Spielkarte extends JPanel {
    public SpielObjekt newObject;
    //Arbeitsgeschwindigkeit:
    public double resourceFactor = 1;

    public SpielObjekt getNewObject() {
        return newObject;
    }

    public void setNewObject(SpielObjekt newObject) {
        this.newObject = newObject;
    }
    //Array für die Position der SpielObjekte auf der Karte:
    public SpielObjekt spielObjekte[][];

    public SpielObjekt[][] getSpielObjekte() {
        return spielObjekte;
    }

    public ZentralLager getZentralLager() {
        return ((ZentralLager)
this.spielObjekte[this.zentralLagerX][this.zentralLagerY]);
    }
    //Kartengrösse:
    int mapsizeX;
    int mapsizeY;

    //Scrollwerte:
    int horizontal;
    int vertical;

    //Karten-Array zum speichern des Untergrunds der Karte (Wald, Wasser usw.):
```

```
int map_array[][];

public int getMapsizeX() {
    return mapsizeX;
}

public void setMapsizeX(int mapsizeX) {
    this.mapsizeX = mapsizeX;
}

public int getMapsizeY() {
    return mapsizeY;
}

public void setMapsizeY(int mapsizeY) {
    this.mapsizeY = mapsizeY;
}

public int getHorizontal() {
    return horizontal;
}

public void setHorizontal(int horizontal) {
    this.horizontal = horizontal;
}

public int getVertical() {
    return vertical;
}

public void setVertical(int vertical) {
    this.vertical = vertical;
}

public boolean zentralLager;
public int zentralLagerX;
public int zentralLagerY;

public int getZentralLagerX() {
    return zentralLagerX;
}

public void setZentralLagerX(int zentralLagerX) {
    this.zentralLagerX = zentralLagerX;
}

public int getZentralLagerY() {
    return zentralLagerY;
}

public void setZentralLagerY(int zentralLagerY) {
    this.zentralLagerY = zentralLagerY;
}

public Spielkarte(int sizeX, int sizeY) {
//Setzen der Karteneigenschaften:
    this.mapsizeX = sizeX;
    this.mapsizeY = sizeY;
    this.horizontal = 0;
    this.vertical = 0;
}
```

```

        setLayout(new GridLayout(sizeX, sizeY));
        this.spielObjekte = new SpielObjekt[this.mapsizeX][this.mapsizeX];
        //Generiert Anzahl der besagten Elemente in Prozent auf der Karte:
        //Land:
        this.map_array = generateLand(50);
        //Wald:
        this.map_array = generateForrest(50);
        //Ressourcen (Eisen,Kohle):
        this.map_array = generateRessources(5);
        //Zentrallager existiert noch nicht, muss gesetzt werden vom
Spieler:
        zentrallager = false;
    }

    public int[][] getMapArray() {
        return map_array;
    }

    public void setMapArray(int map_array[][]) {
        this.map_array = map_array;
    }
    //Fehlerhafte Wegfindung, wird nicht verwendet im Spiel:
    public int[][] pathFind(int currX, int tarX, int currY, int tarY) {

        double[] path_arr = new double[4];
        int[][] weg_array = new int[getMapsizedX()][getMapsizedY()];
        boolean zielerreicht = false;
        while (zielerreicht == false) {

            path_arr[0] = Math.abs(currX - tarX) + Math.abs((currY - 1) -
tarY);
            path_arr[1] = Math.abs((currX + 1) - tarX) + Math.abs(currY -
tarY);
            path_arr[2] = Math.abs(currX - tarX) + Math.abs((currY + 1) -
tarY);
            path_arr[3] = Math.abs((currX - 1) - tarX) + Math.abs(currY -
tarY);

            double erg = path_arr[0];
            int a = 0;
            for (int i = 0; i < path_arr.length; i++) {
                if (path_arr[i] < erg) {
                    erg = path_arr[i];
                    a = i;
                }
            }
            switch (a) {
                case 0:
                    currY -= 1;
                    break;
                case 1:
                    currX += 1;
                    break;
                case 2:
                    currY += 1;
                    break;
                case 3:
                    currX -= 1;
                    break;
            }
            weg_array[currX][currY] = 1;
        }
    }

```



```
        if (currX == tarX && currY == tarY) {
            zielerreicht = true;
        }
    }

    for (int z = 0; z < getMapsizeX(); z++) {
        for (int g = 0; g < getMapsizeY(); g++) {
            if (weg_array[z][g] == 1) {
                map_array[z][g] = 2;
            }
        }
    }

    setMapArray(map_array);
    return map_array;
}

//Land-Generator:
public int[][] generateLand(int freq) {
    int w = getMapsizeX();
    int h = getMapsizeY();
    int map_array[][] = new int[w][h];
    //Über die größe des ganzen Karten-Arrays:
    for (int x = 1; x < w - 3; x++) {
        for (int y = 2; y < h - 3; y++) {
            //Für die Umsetzung des Prozentsatzes:
            int rndMr = MrIndustry.rndZahl(100);
            //Wenn ein Feld von 2x2 Feldern auf die Position des
            Zeigers auf der Karte passt,
            //wird es dort eingefügt (Die Position der Felder ist
            zufällig):

            //Die Nummern im Kartenarray sagen dem Spiel welche
            Textur geladen werden soll.

            if (rndMr <= freq && x != 1 && y != 1 && x % 4 == 0
                && y % 4 == 0) {

                map_array[x][y] = 1;
                map_array[x + 1][y] = 1;
                map_array[x][y + 1] = 1;
                map_array[x + 1][y + 1] = 1;
            }
        }
    }

    //Zur Generierung von Landgruppen, damit die Karte nicht zu
    "löchrig" ist:
    // (Es wird nur noch Land hinzugefügt wenn auf dem Zeiger auch Land
    ist):
    for (int x = 4; x < w - 4; x++) {
        for (int y = 4; y < h - 4; y++) {
            if (x > 1 && y > 1 && x < w - 1 && y < h - 1) {
                if (map_array[x][y] == 1) {
                    if (MrIndustry.rndZahl(100) <= freq) {
                        map_array[x][y] = 1;
                    }
                }
            }
        }
    }
}
```

```

        map_array[x + 1][y] = 1;
        map_array[x][y + 1] = 1;
        map_array[x + 1][y + 1] = 1;

    }

}

}

}

//Füllen von kleinen Landlücken, ansonsten kann der Strand nicht
richtig generiert werden:
    for (int x = 1; x < w; x++) {
        for (int y = 2; y < h; y++) {
            // Beispiel:
            // Wenn [1][0][1] dann [1][1][1]
            // Somit kann eine Horizontale Lücke geschlossen werden
            if (map_array[x][y] >= 1 && map_array[x + 1][y] < 1
                && map_array[x + 2][y] >= 1) {
                map_array[x + 1][y] = 1;

            }

            if (map_array[x][y] >= 1 && map_array[x + 1][y] < 1
                && map_array[x + 2][y + 2] >= 1) {
                map_array[x][y + 1] = 1;

            }

            if (map_array[x][y] >= 1 && map_array[x][y + 1] < 1
                && map_array[x][y + 2] >= 1) {
                map_array[x][y + 1] = 1;

            }

        }

    }

//Generierung vom Strand:
    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            if (x > 0 && y > 0 && x < w - 1 && y < h - 1) {
                //Der Strand wird nur im Wasser gemacht:
                if (map_array[x][y] == 0) {
                    //Strand am Wasserrand links:
                    //[0][1] wird zu [11][0]:
                    if (map_array[x + 1][y] == 1) {
                        map_array[x][y] = 11;
                    } else if (map_array[x][y + 1] == 1) {
                        map_array[x][y] = 12;
                    } else if (map_array[x - 1][y] == 1) {
                        map_array[x][y] = 13;
                    } else if (map_array[x][y - 1] == 1) {
                        map_array[x][y] = 14;
                    }

                    } else if (map_array[x + 1][y] == 0
                        && map_array[x + 1][y + 1] == 1
                        && map_array[x][y + 1] == 0) {
                        map_array[x][y] = 15;
                    }

                    //Strand am Wasserrand oben rechts:
                    //[0][0]
                    [0][16]

```

```

//[[1][0] wird zu [[1][0]
else if (map_array[x - 1][y] != 1
        && map_array[x][y + 1] != 1
        && map_array[x - 1][y + 1] == 1)
{
    map_array[x][y] = 16;
}

else if (map_array[x - 1][y] != 1
        && map_array[x][y + 1] != 1
        && map_array[x - 1][y - 1] == 1)
{
    map_array[x][y] = 17;
}

else if (map_array[x + 1][y] != 1
        && map_array[x + 1][y - 1] >= 1
        && map_array[x][y - 1] != 1) {
    map_array[x][y] = 18;
}

if (map_array[x - 1][y] == 1
    && map_array[x][y - 1] == 1) {
    map_array[x][y] = 19;
}

if (map_array[x + 1][y] == 1
    && map_array[x][y - 1] == 1
    && map_array[x + 1][y - 1] == 1)
{
    map_array[x][y] = 20;
}

if (map_array[x - 1][y] == 1
    && map_array[x - 1][y + 1] == 1
    && map_array[x][y + 1] == 1) {
    map_array[x][y] = 21;
}

if (map_array[x + 1][y] == 1
    && map_array[x][y + 1] == 1) {
    map_array[x][y] = 22;
}
}
}
}
return map_array;
}

//Überprüfung ob der Wald einen gewissen Abstand vom Wasser hat(für den
Waldgenerator):
public boolean checkFor(int x, int y) {
    boolean verification = false;
    int map_array[][] = getMapArray();
    if (map_array[x][y] >= 1 && map_array[x - 1][y] >= 1
        && map_array[x - 2][y] >= 1 && map_array[x + 1][y] >= 1
        && map_array[x + 2][y] >= 1 && map_array[x][y - 1] >= 1
        && map_array[x][y - 2] >= 1 && map_array[x][y + 1] >= 1

```

```

1] >= 1
1] >= 1

2] >= 1
2] >= 1

>= 1
1] >= 1
1] >= 1
1] >= 1

1] <= 2
1] <= 2

2] <= 2
2] <= 2

<= 2
1] <= 2
1] <= 2
1] <= 2

    && map_array[x - 2][y + 2] <= 2) {
        verification = true;
    } else {
        verification = false;
    }
    return verification;
}

//Waldgenerator, funktioniert fast identisch wie der Landgenerator:
public int[][] generateForrest(int freq) {
    int w = getMapsizeX();
    int h = getMapsizeY();
    int map_array[][] = getMapArray();
    for (int x = 1; x < w; x++) {
        for (int y = 2; y < h; y++) {
            int rndMr = MrIndustry.rndZahl(100);
            if (rndMr <= freq) {
                if (checkFor(x, y) == true) {

```

```

//Es werden einfach nur einzelne Felder
hinzugefügt:
    map_array[x][y] = 2;
}
}
}
//Erzeugung von "Waldgruppen", damit nicht einzelne Felder Wald haben sondern
mehrere nebeneinander:
    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            if (x > 1 && y > 1 && x < w - 1 && y < h - 1) {
                if (map_array[x][y] == 2) {
                    if (MrIndustry.rndZahl(100) <= freq) {
                        if (checkFor(x, y) == true) {
                            map_array[x][y] = 2;
                        }
                        if (checkFor(x + 1, y) == true) {
                            map_array[x + 1][y] = 2;
                        }
                        if (checkFor(x + 1, y + 1) == true) {
                            map_array[x + 1][y + 1] = 2;
                        }
                        if (checkFor(x, y + 1) == true) {
                            map_array[x][y + 1] = 2;
                        }
                    }
                }
            }
        }
    }
//Schliessung von zu kleinen Waldlichtungen (Wie beim
Landgenerator):
    for (int x = 1; x < w; x++) {
        for (int y = 2; y < h; y++) {
            if (map_array[x][y] == 2 && map_array[x + 1][y] == 1
                && map_array[x + 2][y] == 2) {
                map_array[x + 1][y] = 2;
            }
            if (map_array[x][y] == 2 && map_array[x + 1][y] == 1
                && map_array[x + 2][y + 2] == 2) {
                map_array[x][y + 1] = 2;
            }
            if (map_array[x][y] == 2 && map_array[x][y + 1] == 1
                && map_array[x][y + 2] == 2) {
                map_array[x][y + 1] = 2;
            }
        }
    }
//Erzeugung des Waldrands, natürlich mit anderen Texturnummern:
    for (int x = 0; x < w; x++) {

```

```

for (int y = 0; y < h; y++) {
    if (x > 0 && y > 0 && x < w - 1 && y < h - 1) {
        if (map_array[x][y] == 1) {

            if (map_array[x + 1][y] == 2) {
                map_array[x][y] = 3;
            } else if (map_array[x][y + 1] == 2) {
                map_array[x][y] = 4;
            } else if (map_array[x - 1][y] == 2) {
                map_array[x][y] = 5;
            } else if (map_array[x][y - 1] == 2) {
                map_array[x][y] = 6;

            } else if (map_array[x + 1][y] == 1
                && map_array[x + 1][y + 1] == 2
                && map_array[x][y + 1] == 1) {
                map_array[x][y] = 9;
            } else if (map_array[x - 1][y - 1] == 2) {
                map_array[x][y] = 7;
            }

            else if (map_array[x - 1][y + 1] == 2) {
                map_array[x][y] = 10;
            }

            else if (map_array[x + 1][y - 1] == 2) {
                map_array[x][y] = 8;
            }

            if (map_array[x - 1][y] == 2
                && map_array[x][y - 1] == 2) {
                map_array[x][y] = 23;
            }

            if (map_array[x + 1][y] == 2
                && map_array[x][y - 1] == 2
                && map_array[x + 1][y - 1] == 2)

                map_array[x][y] = 24;
            }

            if (map_array[x - 1][y] == 2
                && map_array[x][y + 1] == 2) {
                map_array[x][y] = 25;
            }

            if (map_array[x + 1][y] == 2
                && map_array[x][y + 1] == 2) {
                map_array[x][y] = 26;
            }
        }
    }
}

return map_array;
}

```

//Generierung von Ressourcen, ähnlich wie beim Wald, jedoch ohne Gruppierung:

```

public int[][] generateRessources(int freq) {
    int[][] map_array = getMapArray();
    int w = getMapsizeX();
    int h = getMapsizeY();
    for (int x = 4; x < w; x++) {
        for (int y = 4; y < h; y++) {
            if (x > 1 && y > 1 && x < w - 1 && y < h - 1) {
                if (map_array[x][y] == 1) {
                    if (MrIndustry.rndZahl(100) <= freq) {

                        switch (MrIndustry.rndZahl(2)) {
                            case 1:
                                map_array[x][y] = 28;
                                break;

                            case 2:
                                map_array[x][y] = 27;
                                break;
                            default:
                                map_array[x][y] = 27;
                                break;
                        }
                    }
                }
            }
        }
    }

    return map_array;
}

public void add(SpielObjekt Obj) {
    this.spielObjekte[Obj.locX][Obj.locY] = Obj;
}
//Nachdem das map_array gefüllt wurde, werden die Entsprechenden
Objekte/Texturen der Karte hinzugefügt
public void newGame() {
    int[][] map_array = getMapArray();

    for (int x = 0; x < this.spielObjekte.length; x++) {
        for (int y = 0; y < this.spielObjekte[x].length; y++) {

            this.spielObjekte[x][y] = new Umgebung(map_array[x][y],
x, y);

        }
    }
    //Das Zentrallager wird beim Spielstart/nach der Generierung der
Karte hinzugefügt:
    this.newObject = new ZentralLager(0, 0, true, this);

}
//Alle Objekte darstellen:
public void zeichneAlle(Graphics g) {
    for (int x = 0; x < this.spielObjekte.length; x++) {

```



```

        for (int y = 0; y < this.spielObjekte[x].length; y++) {

            if (this.spielObjekte[x][y].getDragDrop() == false)
                this.spielObjekte[x][y].zeichne(g,

this.horizontal,

                                this.vertical);

        }

        for (int x = 0; x < this.spielObjekte.length; x++) {
            for (int y = 0; y < this.spielObjekte[x].length; y++) {

                //Darstellung der Markierung der Elemente wo der
Mauszeiger drüber ist:
                if (this.spielObjekte[x][y].getHighlight() == true) {
                    //Abrufen der Farbe der Markierung (Konflikte im
Baumodus):

                    if (this.spielObjekte[x][y].getTileColor() ==

true) {

                        g.setColor(Color.YELLOW);
                    } else {
                        g.setColor(Color.RED);
                    }

                    g.drawRect(

                                this.spielObjekte[x][y].locX
                                    * MrIndustry.textureSize
                                    - this.getHorizontal() - 1,
                                this.spielObjekte[x][y].locY
                                    * MrIndustry.textureSize
                                    - this.getVertical() - 1,
                                (MrIndustry.textureSize + 1),
                                (MrIndustry.textureSize + 1));
                    g.setColor(Color.BLACK);
                }

                //Objekte welche dem Mauszeiger folgen müssen am Schluss
gerendert werden,
                //ansonsten würden sie von anderen Objekten auf der
Karte überdeckt werden:
                if (this.spielObjekte[x][y].getDragDrop() == true)
                    this.spielObjekte[x][y].zeichne(g,

this.horizontal,

                                this.vertical);

            }

            //Das gleiche gilt hier:
            if (this.newObject != null) {
                this.newObject.zeichne(g, this.horizontal, this.vertical);
            }

        }

        public void paintComponent(Graphics g) {
            super.paintComponent(g);
            this.zeichneAlle(g);
        }

        //Deaktivieren der Gebäude wenn zu wenig Rohstoffe vorhanden sind:
        public void shutDownOne() {
            int x;
            int y;
            boolean successful = false;

```

```

        x = 0;
        y = 0;
        while (!successful) {
            if (this.spielObjekte[x][y].bevoelkerung > 0
                && this.spielObjekte[x][y].isActive()) {
                successful = true;
                this.spielObjekte[x][y].activate(false);
            } else {
                x++;

                if (x > this.spielObjekte.length - 1) {
                    x = 0;
                    y++;
                }
                if (y > this.spielObjekte[x].length - 1) {
                    y = 0;
                }
            }
        }
    }
}

SpielObjekt.java
package mrIndustry;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.GraphicsDevice;
import java.awt.GraphicsEnvironment;
import java.awt.Transparency;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.Vector;

import javax.imageio.ImageIO;

import mrIndustry.SpielObjekte.LeistungsSteigerung.Autofabrik;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausMittelKlasse;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausOberKlasse;
import mrIndustry.SpielObjekte.PrivateObjekte.WohnhausUnterKlasse;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Bergwerk;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Eisenmine;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Farm;
import mrIndustry.SpielObjekte.Rohstoffgebaeude.Försterei;
import mrIndustry.SpielObjekte.Verarbeitungsgebaude.Baeckerei;
import mrIndustry.SpielObjekte.Verarbeitungsgebaude.Werkzeugfabrik;

public class SpielObjekt {
    // Array speichert die Namen der Objekte, die im Baumenü angezeigt werden.
    public static String[][] list_text = {
        { "Bergwerk", "Eisenmine", "Försterei", "Farm" },
        { "Autofabrik", "Werkzeugfabrik", "Bäckerei" },
        { "Markt", "Bar", "Theater" },
        { "Unterschicht", "Mittelschicht", "Oberschicht" },
    }

```

```

        { "Statue", "Park" }, { "Strassenstück" } };

/*
ID:6 * Index von kosten[][][x]: Nahrung:0 Kohle:1 Eisen:2 Arbeiter:3 Holz:
* Index von kosten[][x][]: Bergwerk Eisenmine,Försterei,Farm
* *,Autofabrik,Werkzeugfabrik *,Markt,Bar,Theater *
* Wohngeb.unter.,Wohngeb.mittel.,Wohngeb.ober.,Statue,Park,Strasse Index
* von kosten[x][[]]: Gewinnung,Verarbeitung,Dienstleistung,private
* Gebäude,Sonstiges,Logistik
*/
// Bergwerk[0]:
// Speichert die Kosten der jeweiligen Gebäude
public static int[][][] kosten = {
    { { 0, 0, 3000, 15, 4500, 0 }, { 0, 3000, 2500, 20, 5000, 1 },
      { 0, 2000, 2500, 25, 2000, 2 },
      { 0, 2500, 3000, 10, 3000, 3 } },
    { { 0, 2500, 3500, 15, 3000, 4 },
      { 10000, 0, 3000, 2500, 10, 3500, 5 },
      { 0, 3000, 2500, 10, 3500, 6 } },
    { { 0, 3000, 3000, 10, 3000, 15 },
      { 2500, 0, 500, 800, 5, 1500, 7 },
      { 4000, 0, 1000, 1000, 20, 1500, 8 } },
    { { 0, 100, 500, 0, 1000, 9 }, { 0, 500, 1200, 0, 2000, 10 },
      { 0, 500, 2200, 0, 2500, 11 } },
    { { 0, 0, 3000, 0, 500, 12 }, { 0, 500, 1500, 0, 1000, 13 } },
    { { 0, 200, 100, 0, 100, 14 } } };
// Speichert das aktuelle Frame der Animation der Texturen
protected int paintlevel;
static String abh = "<p>Abhängigkeit der Position:</p>";
// Speichert alle Beschreibungen der Objekte.
public static String[][] arr_desc = {
    {
        "<h3>Bergwerk</h3>Das Bergwerk dient zur Gewinnung
des Rohstoffs Kohle."
        + abh
        + "<br/>Das Gebäude muss auf einem
Kohlevorkommen erbaut werden.",
        "<h3>Eisenmine</h3>Die Eisenmine dient zur
Gewinnung des Rohstoffs Eisen."
        + abh
        + "<br/>Das Gebäude muss auf einem
Eisenvorkommen erbaut werden.",
        "<h3>Försterei</h3>Die Försterei dient zur
Gewinnung des Rohstoffs Holz."
        + abh
        + "<br/>Das Gebäude muss an einem Wald
erbaut werden.",
        "<h3>Farm</h3>Die Farm dient zur Gewinnung des
Rohstoffs Nahrung."
        + abh
        + "<br/>Das Gebäude muss auf einer
Wiese gebaut werden." },
    {
        "<h3>Autofabrik</h3>Die Autofabrik baut und
verkauft Autos."
        + abh
        + "<br/>Das Gebäude darf nicht auf
Hindernissen stehen.",
        "<h3>Werkzeugfabrik</h3>Die Werkzeugfabrik stellt
Werkzeuge her und verkauft diese."
    }
}

```

```

+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen.",
    "<h3>Bäckerei</h3>Die Bäckerei macht aus Nahrung
und Holz zwei Nahrung"
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen." },
    {
        "<h3>Markt</h3>Der Markt ermöglicht dir Rohstoffe
einzukaufen oder diese in andere umzuwandeln."
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen.",
        "<h3>Bar</h3>Erhöht den Gesamtertrag"
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen.",
        "<h3>Theater</h3>Erhöht den Gesamtertrag"
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen." },
        {
            "<h3>Wohngebäude Unterschicht</h3>Das Gebäude
welches Bewohner hat welche am wenigsten Steuern zahlen können."
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen.",
            "<h3>Wohngebäude Mittelschicht</h3>Das Gebäude
welches Bewohner hat welche am mittelmässig Steuern zahlen können."
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen.",
            "<h3>Wohngebäude Oberschicht</h3>Das Gebäude
welches Bewohner hat welche am meisten Steuern zahlen können."
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen." },
        {
            "<h3>Statue von Mr. Industry</h3>Eine grosse
Statue, welche das Aussehen der Siedlung aufwertet."
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen.",
            "<h3>Park</h3>Ein Park welcher die Bewohner der
Siedlung zum Entspannen einlädt."
+ abh
+ "<br/>Das Gebäude darf nicht auf
Hindernissen stehen." },
        { "<h3>Strassenstück</h3>Ein Strassenstück welches die
Abwicklung der Logistik in der Siedlung ermöglicht. Die Kosten werden pro
Strassenstück (oder auch Strassenparzelle) abgerechnet."
+ abh
+ "<br/>Das Bauwerk darf nicht auf Hindernissen
stehen. Die Strasse muss zwingend mit mindestens einem Gebäude und dem
Strassennetz welches mit dem Hauptlager verbunden ist verbunden sein. Die
Strasse ist also immer in einem Stück." } };
// Statische HashMap für die Texturen. Damit müssen die Texturen nur
einmal
// ausgelesen werden.

```

```

    protected static Map<String, BufferedImage> textureImages = new
HashMap<String, BufferedImage>();
    // Multidimensionaler Vector. Speichert das alle Texturen die übereinander
    // geladen werden für jedes Frame in der Animation.
    protected Vector<Vector<BufferedImage>> textures;
    int locX;
    int locY;
    int mapsizeX;
    int mapsizeY;
    boolean dragdrop;
    boolean highlight;
    boolean tileColor;
    boolean activatedOnce;
    protected long lastTime;
    protected Calendar now;
    protected Spielkarte map;
    // Speichert ob das Gebäude aktiv ist oder nicht
    public boolean isActive;
    // Bevoelkerungs Kosten. Wird verwendet damit Bevölkerung sterben kann.
    int bevoelkerung;

    // Aktiviert bzw. deaktiviert ein Gebäude
    public void activate(boolean isActive) {
        this.isActive = isActive;
        if (!(this.isActive()) || !(this.activatedOnce)) {
            this.activatedOnce = true;
            this.loadTexture("inactive.png", 0);
            if (map.zentrallager) {
                // Löscht die Bevölkerung des Gebäudes
                map.getZentrallager().bevoelkerungAktiv -=
this.bevoelkerung;
                map.getZentrallager().bevoelkerungInaktiv +=
this.bevoelkerung;
            }
        }
        if (this.isActive()) {
            this.textures.lastElement().removeElementAt(
                this.textures.lastElement().size() - 1);
            if (map.zentrallager) {
                // Fügt die Bevölkerung des Gebäudes hinzu
                map.getZentrallager().bevoelkerungAktiv +=
this.bevoelkerung;
                map.getZentrallager().bevoelkerungInaktiv -=
this.bevoelkerung;
            }
        }
    }

    public boolean isActive() {
        return isActive;
    }

    public void doAfterBuild() {
    }

    public Spielkarte getMap() {
        return map;
    }

```

```
public void setMap(Spielkarte map) {
    this.map = map;
}
//Wird bei jedem Intervall ausgeführt.
public void tick() {

}

public boolean getDragDrop() {
    return dragdrop;
}

public void setDragDrop(boolean dragdrop) {
    this.dragdrop = dragdrop;
}

public boolean getHighlight() {
    return highlight;
}

public void setHighlight(boolean highlight) {
    this.highlight = highlight;
}

public boolean getTileColor() {
    return tileColor;
}

public void setTileColor(boolean color) {
    this.tileColor = color;
}

public int getLocX() {
    return locX;
}

public void setLocX(int locX) {
    this.locX = locX;
}

public int getLocY() {
    return locY;
}

public void setLocY(int locY) {
    this.locY = locY;
}

public void zeichne(Graphics g, int x, int y) {

    for (int i = 0; i < this.textures.elementAt(this.paintlevel).size();
i++) {
        // Zeichnet das Objekt wenn es platziert wird
        if (getDragDrop() == true) {
            g.drawImage(
                (BufferedImage) this.textures
                    .elementAt(this.paintlevel).elementAt(i),
                    this.locX, this.locY,
MrIndustry.textureSize,
                    MrIndustry.textureSize, null);
        }
    }
}
```

```

        } else {
            // Zeichnet das aktuelle Frame der Animation
            g.drawImage(

                this.textures.elementAt(this.paintlevel).elementAt(i),
                this.locX * MrIndustry.textureSize - x,
                this.locY
                                * MrIndustry.textureSize - y,
                                MrIndustry.textureSize,
                                MrIndustry.textureSize, null);
        }

    }
    // Nächste Position in der Animation
    if (this.paintlevel < this.textures.size() - 1) {
        this.paintlevel++;
    } else {
        this.paintlevel = 0;
    }
    /*
    * if (getHighlight() == true) { if (getTileColor()==true) {
    * g.setColor(Color.YELLOW); } else { g.setColor(Color.RED); }
    *
    * g.drawRect(this.locX * MrIndustry.textureSize - x-1, this.locY
    * MrIndustry.textureSize - y, MrIndustry.textureSize,
    * MrIndustry.textureSize); g.setColor(Color.BLACK); }
    */
}

public SpielObjekt() {

    this.textures = new Vector<Vector<BufferedImage>>();
    this.paintlevel = 0;
    // Liest die Bevölkerung und bei Bedarf auch die Restlichen Kosten
aus
    // dem eigenen Objekt.
    if (this instanceof Bergwerk) {
        this.bevoelkerung = SpielObjekt.kosten[0][0][3];
    }

    if (this instanceof Eisenmine) {
        this.bevoelkerung = SpielObjekt.kosten[0][1][3];
    }

    if (this instanceof Försterei) {
        this.bevoelkerung = SpielObjekt.kosten[0][2][3];
    }

    if (this instanceof Farm) {
        this.bevoelkerung = SpielObjekt.kosten[0][3][3];
    }

    if (this instanceof Autofabrik) {
        this.bevoelkerung = SpielObjekt.kosten[1][0][3];
    }

    if (this instanceof Werkzeugfabrik) {
        this.bevoelkerung = SpielObjekt.kosten[1][1][3];
    }
}

```



```
        if (this instanceof Baeckerei) {
            this.bevoelkerung = SpielObjekt.kosten[1][2][3];
        }

        if (this instanceof WohnhausUnterKlasse) {
            this.bevoelkerung = SpielObjekt.kosten[3][0][3];
        }

        if (this instanceof WohnhausMittelKlasse) {
            this.bevoelkerung = SpielObjekt.kosten[3][1][3];
        }

        if (this instanceof WohnhausOberKlasse) {
            this.bevoelkerung = SpielObjekt.kosten[3][2][3];
        }

        if (this instanceof Baeckerei) {
            this.bevoelkerung = SpielObjekt.kosten[0][1][3];
        }
    }

    // Lädt eine neue Textur für das Objekt. level gibt die Position in der
    // Animation an. Wird die Methode mehrmals mit dem selben level aufgerufen
    // wird die neue Texture die alte überlagern.
    public void loadTexture(String filename, int level) {
        if (textureImages.get(filename) == null) {
            try {
                BufferedImage image = ImageIO.read(new File(filename));
                textureImages.put(filename, image);
            } catch (IOException e) {
                // TODO Auto-generated catch block
                System.out.println(filename + " not found!");
            }
        }
        try {
            this.textures.elementAt(level);
        } catch (ArrayIndexOutOfBoundsException e) {
            this.textures.add(new Vector<BufferedImage>());
        }
        this.textures.elementAt(level).add(textureImages.get(filename));
    }

    public Calendar now() {
        Calendar cal = Calendar.getInstance();
        return cal;
    }

    public void destroy() {
        // TODO Auto-generated method stub
    }
}

LeistungSteigerungsObjekt.java
package mrIndustry.SpielObjekte;

import mrIndustry.SpielObjekt;
```

```
public class LeistungsSteigerungsObjekt extends SpielObjekt {
    //Arbeitsgeschwindigkeit:
    protected double resoureFactorIncrement = 0;
    //Übernahme der Eigenschaften von Oberklassen:
    public LeistungsSteigerungsObjekt() {
        super();
    }
    //Veränderung der Arbeitsgeschwindigkeit der Siedlung, abhängig von der
    Aktivität:
    public void activate(boolean isActive) {
        super.activate(isActive);
        if (isActive) {
            map.resourceFactor += this.resoureFactorIncrement;
        } else {
            map.resourceFactor -= this.resoureFactorIncrement;
        }
    }

    public void destroy() {
        this.activate(false);
        super.destroy();
    }
}
```

PrivatesObjekt.java

```
package mrIndustry.SpielObjekte;

import mrIndustry.SpielObjekt;

public class PrivatesObjekt extends SpielObjekt {
    //Bevölkerung der Siedlung:
    protected int bevoelkerung;
    //Übernahme der Eigenschaften von Oberklassen:
    public PrivatesObjekt() {
        super();
    }
    //Veränderung der Anzahl an Bevölkerung der Siedlung, abhängig von der
    Aktivität:
    public void activate(boolean active) {
        if (active) {
            this.map.getZentralLager().bevoelkerungSoll += bevoelkerung;
        } else {
            this.map.getZentralLager().bevoelkerungSoll -= bevoelkerung;
        }
        super.activate(active);
    }
}
```

Rohstoffgewinnung.java

```
package mrIndustry.SpielObjekte;

import mrIndustry.SpielObjekt;

public class Rohstoffgewinnung extends SpielObjekt {
    public Rohstoffgewinnung() {
        super();
    }
}
```

Umgebung.java

```
package mrIndustry.SpielObjekte;

//import java.awt.Graphics;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;

import mrIndustry.MrIndustry;
import mrIndustry.SpielObjekt;

public class Umgebung extends SpielObjekt {
    int Gelaende;
    //Umgebung (Wald,Wasser,Land usw..) im Grunde genommen ein Kartenfeld:
    public Umgebung(int Gelaende, int locX, int locY) {
        super();
        this.setLocX(locX);
        this.setLocY(locY);
        this.Gelaende = Gelaende;
        this.setHighlight(false);
        //Hinzufügen der jeweiligen Texturen:
        switch (this.Gelaende) {
            case 0:
                this.addWater();
                break;
            //Hinzufügen von zufälliger Szenerie (Steine,Blumen,Pflanzen):
            case 1:
                double rnd = Math.random();
                double rnd1 = Math.random();
                double rnd2 = Math.random();
                double rnd3 = Math.random();
                double rnd4 = Math.random();
                double rnd5 = Math.random();
                double rnd6 = Math.random();

                int count = (int) (Math.random() * 100);
                for (int i = 0; i <= 1; i++) {

                    this.loadTexture("grass.png", i);
                    if (rnd1 > 0.9) {
                        this.loadTexture("bush0.png", i);
                    }
                    if (rnd2 > 0.95) {
                        this.loadTexture("flowerYellow0.png", i);
                    }
                    if (rnd3 > 0.95) {
                        this.loadTexture("flowerYellow1.png", i);
                    }
                    if (rnd4 > 0.95) {
                        this.loadTexture("flowerYellow2.png", i);
                    }
                    if (rnd5 > 0.95) {
                        this.loadTexture("stone0.png", i);
                    }
                    if (rnd6 > 0.95) {
                        this.loadTexture("stone1.png", i);
                    }
                }
            }
        }
    }
}
```

```
    }  
    break;  
  
case 2:  
    this.loadTexture("forrest.png", 0);  
    break;  
case 3:  
    this.loadTexture("grass.png", 0);  
    this.loadTexture("forrest_left.png", 0);  
    break;  
case 4:  
    this.loadTexture("grass.png", 0);  
    this.loadTexture("forrest_top.png", 0);  
    break;  
case 5:  
    this.loadTexture("grass.png", 0);  
    this.loadTexture("forrest_right.png", 0);  
    break;  
case 6:  
    this.loadTexture("grass.png", 0);  
    this.loadTexture("forrest_bot.png", 0);  
    break;  
case 7:  
    this.loadTexture("grass.png", 0);  
    this.loadTexture("forrest_bot_right.png", 0);  
    break;  
case 8:  
    this.loadTexture("grass.png", 0);  
    this.loadTexture("forrest_bot_left.png", 0);  
    break;  
case 9:  
    this.loadTexture("grass.png", 0);  
    this.loadTexture("forrest_top_left.png", 0);  
    break;  
case 10:  
    this.loadTexture("grass.png", 0);  
    this.loadTexture("forrest_top_right.png", 0);  
    break;  
case 11:  
    this.addWater();  
    this.addToWater("water_left.png");  
  
    break;  
case 12:  
    this.addWater();  
  
    this.addToWater("water_top.png");  
  
    break;  
case 13:  
    this.addWater();  
    this.addToWater("water_right.png");  
  
    break;  
case 14:  
    this.addWater();  
    this.addToWater("water_bot.png");  
  
    break;  
case 15:  
    this.addWater();
```

```
        this.addToWater("water_bot_right.png");

        break;
    case 16:
        this.addWater();
        this.addToWater("water_bot_left.png");
        break;
    case 17:
        this.addWater();
        this.addToWater("water_top_left.png");
        break;
    case 18:
        this.addWater();
        this.addToWater("water_top_right.png");
        break;
    case 19:
        this.addWater();
        this.addToWater("water_ecke_top_left.png");
        break;
    case 20:
        this.addWater();
        this.addToWater("water_ecke_top_right.png");
        break;
    case 21:
        this.addWater();
        this.addToWater("water_ecke_bot_left.png");
        break;
    case 22:
        this.addWater();
        this.addToWater("water_ecke_bot_right.png");
        break;
    case 23:
        this.loadTexture("grass.png", 0);
        this.loadTexture("forrest_ecke_top_left.png", 0);
        break;
    case 24:
        this.loadTexture("grass.png", 0);
        this.loadTexture("forrest_ecke_top_right.png", 0);
        break;
    case 25:
        this.loadTexture("grass.png", 0);
        this.loadTexture("forrest_ecke_bot_left.png", 0);
        break;
    case 26:
        this.loadTexture("grass.png", 0);
        this.loadTexture("forrest_ecke_bot_right.png", 0);
        break;
    case 27:
        this.loadTexture("grass.png", 0);
        this.loadTexture("eisen.png", 0);
        break;
    case 28:
        this.loadTexture("grass.png", 0);
        this.loadTexture("kohle.png", 0);
        break;
    }

    }

    private void addWater() {
        //Animiertes Wasser
```

//Damit nicht alle Wasserfelder das gleiche Bild aufzeigen, wird dies hier abgewechselt:

```
double rnd = Math.random();
double rnd1 = Math.random();
int waterRnd = (int) (Math.random() * 6);
int sharkRnd = 0;
int incrementWater = 1;

for (int i = 0; i <= 33; i++) {

    this.loadTexture("waterGroundColor.png", i);
    if (rnd1 > 0.8) {

        this.loadTexture("fish/fish" + (i + 1) + ".png", i);
    }
    if (rnd > 0.98) {
        this.loadTexture("fish/hai" + sharkRnd + ".png", i);

        if (Math.random() > 0.5) {
            sharkRnd += 1;
        } else {
            sharkRnd -= 1;
        }
        if (sharkRnd > 9) {
            sharkRnd -= 2;
        }
        if (sharkRnd < 0) {
            sharkRnd += 2;
        }
    }
    if (rnd > 0.99) {
        this.loadTexture("boatwreck.png", i);
    }

    this.loadTexture("water" + waterRnd + ".png", i);
    waterRnd = waterRnd + incrementWater;
    if (Math.random() > 0.7) {

        incrementWater = incrementWater / (-1);
    }
    if (waterRnd == 7) {
        waterRnd = 5;
        incrementWater = -1;
    }
    if (waterRnd == -1) {
        waterRnd = 1;
        incrementWater = 1;
    }
}
```

```
private void addToWater(String filename) {
    for (int i = 0; i <= 33; i++) {
        this.loadTexture(filename, i);
    }
}
```

```
}
```

Verarbeitung.java

```
package mrIndustry.SpielObjekte;

import mrIndustry.SpielObjekt;

public class Verarbeitung extends SpielObjekt {
    public Verarbeitung() {
        super();
    }
}
```

Zentrallager.java

```
package mrIndustry.SpielObjekte;

import java.util.Calendar;
import java.util.Date;

import javax.swing.JLabel;

import mrIndustry.MrIndustry;
import mrIndustry.SpielObjekt;
import mrIndustry.Spielkarte;

//Das Zentrallager speichert die Ressourcen des aktuellen spiels.
public class Zentrallager extends SpielObjekt {
    public int getEisen() {
        return eisen;
    }

    public void setEisen(int eisen) {
        this.eisen = eisen;
    }

    public int getKohle() {
        return kohle;
    }

    public void setKohle(int kohle) {
        this.kohle = kohle;
    }

    public int getNahrung() {
        return nahrung;
    }

    public void setNahrung(int nahrung) {
        this.nahrung = nahrung;
    }

    public int bevoelkerungSoll;

    public int getBevoelkerungSoll() {
        return bevoelkerungSoll;
    }

    public void setBevoelkerungSoll(int bevoelkerungIst) {
        this.bevoelkerungSoll = bevoelkerungIst;
    }

    public int bevoelkerungAktiv;
    public int bevoelkerungInaktiv;
    public int eisen;
```



```
public int kohle;
public int holz;
public int nahrung;

public ZentralLager(int locX, int locY, boolean drag, Spielkarte map) {
    super();
    this.setMap(map);
    this.setLocX(locX);
    this.setLocY(locY);
    this.setDragDrop(drag);
    this.loadTexture("grass.png", 0);
    this.loadTexture("zentrallager.png", 0);
    lastTime = this.now().getTimeInMillis();
}

// Wird nach dem Bau ausgeführt. Die Standard Ressourcen müssen hier
// eingestellt werden.
public void doAfterBuild() {
    this.holz = 10000;
    this.kohle = 10000;
    this.nahrung = 200;
    this.eisen = 10000;
}

public int getHolz() {
    return holz;
}

public void setHolz(int holz) {
    this.holz = holz;
}

public void tick() {
    now = this.now();
    //Wird alle 5 Sekunden ausgeführt
    if (now.getTimeInMillis() - lastTime >= 5000) {
        //Bevölkerung wird ausgerechnet

        //Bevölkerung zieht hinzuh
        if (this.bevoelkerungSoll > this.bevoelkerungInaktiv
            + this.bevoelkerungAktiv) {
            if (this.nahrung > 0) {
                this.bevoelkerungInaktiv++;
            }
            //Bevölkerung zieht weg.
        } else if (this.bevoelkerungSoll < this.bevoelkerungInaktiv
            + this.bevoelkerungAktiv) {
            if (this.bevoelkerungInaktiv > 0) {
                this.bevoelkerungInaktiv--;
            }
        } else {
            this.bevoelkerungAktiv--;
        }
    }

    //Bevölkerung konsummiert Nahrung
    if (this.nahrung
        - (this.bevoelkerungInaktiv +
this.bevoelkerungAktiv) < 0) {
        //Nicht genügend Nahrung. Bevölkerung stirbt
    }
}
```

```

        if (this.bevoelkerungInaktiv > 0) {
            this.bevoelkerungInaktiv--;
            this.nahrung = 0;
        } else {
            //Nicht genügend Bevölkerung. Gebäude werden
            stillgelegt.

            this.nahrung = 0;
            map.shutdownOne();
        }

    } else {
        this.nahrung -= this.bevoelkerungInaktiv
            + this.bevoelkerungAktiv;
    }
    lastTime = this.now().getTimeInMillis();
}

}

```

Autofabrik.java

```

package mrIndustry.SpielObjekte.LeistungsSteigerung;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.LeistungsSteigerungsObjekt;
import mrIndustry.SpielObjekte.Rohstoffgewinnung;

public class Autofabrik extends LeistungsSteigerungsObjekt {

    public Autofabrik(int locX, int locY, boolean drag, final Spielkarte map)
    {
        //Bestimmung der Eigenschaften des Objekts Autofabrik:
        super();
        //Macht die Variable map zur Map:
        this.setMap(map);
        //Bestimmung der Position:
        this.setLocX(locX);
        this.setLocY(locY);
        //Bestimmung ob das Objekt dem Mauszeiger folgen soll oder nicht:
        this.setDragDrop(drag);
        //Bestimmung welche Texturen auf den verschiedenen Ebenen geladen
        werden soll:
        this.loadTexture("grass.png", 0);
        this.loadTexture("autofabrik.png", 0);
        //Angabe der Steigerung der Arbeitsgeschwindigkeit welche die
        Autofabrik bewirkt:
        this.resoureFactorIncrement=0.1;
    }

}

```

WohnHausMittelKlasse.java

```

package mrIndustry.SpielObjekte.PrivateObjekte;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.PrivatesObjekt;

public class WohnhausMittelKlasse extends PrivatesObjekt {
    public WohnhausMittelKlasse(int locX, int locY, boolean drag,
        //Bestimmung der Eigenschaften des Objekts Wohnhaus Mittelkl.:
        final Spielkarte map) {

```

```

        super();
        //Macht die Variable map zur Map:
        this.setMap(map);
        //Bestimmung der Position:
        this.setLocX(locX);
        this.setLocY(locY);
        //Bestimmung ob das Objekt dem Mauszeiger folgen soll oder nicht:
        this.setDragDrop(drag);
        //Bestimmung welche Texturen auf den verschiedenen Ebenen geladen
        werden soll:
        this.loadTexture("grass.png", 0);
        this.loadTexture("house.png", 0);
        //Angabe der Steigerung der Bevölkerung:
        this.bevoelkerung = 15;
    }
}

```

WohnHausOberKlasse.java

```

package mrIndustry.SpielObjekte.PrivateObjekte;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.PrivatesObjekt;

public class WohnhausOberKlasse extends PrivatesObjekt {
    public WohnhausOberKlasse(int locX, int locY, boolean drag,
                               final Spielkarte map) {
        //Hier gilt das gleiche wie beim WohnhausMittelKlasse, einfach
        andere Eigenschaften:
        super();
        this.setMap(map);
        this.setLocX(locX);
        this.setLocY(locY);
        this.setDragDrop(drag);
        this.loadTexture("grass.png", 0);
        this.loadTexture("house.png", 0);
        this.bevoelkerung=25;
    }
}

```

WohnHausUnterKlasse.java

```

package mrIndustry.SpielObjekte.PrivateObjekte;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.PrivatesObjekt;
import mrIndustry.SpielObjekte.ZentralLager;

public class WohnhausUnterKlasse extends PrivatesObjekt {
    public WohnhausUnterKlasse(int locX, int locY, boolean drag,
                               final Spielkarte map) {
        //Hier gilt das gleiche wie beim WohnhausMittelKlasse, einfach
        andere Eigenschaften:
        super();
        this.setMap(map);
        this.setLocX(locX);
        this.setLocY(locY);
        this.setDragDrop(drag);
        this.loadTexture("grass.png", 0);
        this.loadTexture("houseUnterKlasse.png", 0);
        this.bevoelkerung=5;
    }
}

```

```

}
Bergwerk.java
package mrIndustry.SpielObjekte.Rohstoffgebaeude;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.Rohstoffgewinnung;

public class Bergwerk extends Rohstoffgewinnung {
    public Bergwerk(int locX, int locY, boolean drag, final Spielkarte map) {
        super();
        this.setMap(map);
        //Bestimmung der Position:
        this.setLocX(locX);
        this.setLocY(locY);
        //Bestimmung ob das Objekt dem Mauszeiger folgen soll oder nicht:
        this.setDragDrop(drag);
        //Bestimmung welche Texturen auf den verschiedenen Ebenen geladen
        werden soll:
        this.loadTexture("grass.png", 0);
        this.loadTexture("bergwerk.png", 0);

    }
    //Bestimmung wieviel das Gebäude pro Tick an Ressourcen hinzufügt:
    public void tick() {
        now = this.now();
        if (now.getTimeInMillis() - lastTime >= 3000) {

            map.getZentralLager().kohle += 10 * map.resourceFactor;
            lastTime = this.now().getTimeInMillis();

        }
    }
}

```

```

Eisenmine.java
package mrIndustry.SpielObjekte.Rohstoffgebaeude;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.Rohstoffgewinnung;

public class Eisenmine extends Rohstoffgewinnung {
    public Eisenmine(int locX, int locY, boolean drag, final Spielkarte map) {
        super();
        this.setMap(map);
        //Bestimmung der Position:
        this.setLocX(locX);
        this.setLocY(locY);
        //Bestimmung ob das Objekt dem Mauszeiger folgen soll oder nicht:
        this.setDragDrop(drag);
        //Bestimmung welche Texturen auf den verschiedenen Ebenen geladen
        werden soll:
        this.loadTexture("grass.png", 0);
        this.loadTexture("eisenmine.png", 0);

    }
    //Bestimmung wieviel das Gebäude pro Tick an Ressourcen hinzufügt:
    public void tick() {
        now = this.now();
        if (now.getTimeInMillis() - lastTime >= 3000) {

            map.getZentralLager().eisen+=10*map.resourceFactor;
            lastTime = this.now().getTimeInMillis();

        }
    }
}

```

```

    }
}

```

Farm.java

```

package mrIndustry.SpielObjekte.Rohstoffgebaeude;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.Rohstoffgewinnung;

public class Farm extends Rohstoffgewinnung {
    public Farm(int locX, int locY, boolean drag, final Spielkarte map) {
        super();
        this.setMap(map);
        //Bestimmung der Position:
        this.setLocX(locX);
        this.setLocY(locY);
        //Bestimmung ob das Objekt dem Mauszeiger folgen soll oder nicht:
        this.setDragDrop(drag);
        //Bestimmung welche Texturen auf den verschiedenen Ebenen geladen
        werden soll:
        this.loadTexture("grass.png", 0);
        this.loadTexture("farm.png", 0);
        //Bestimmt die Geschwindigkeit des Ticks für das Gebäude:
        lastTime = this.now().getTimeInMillis();
    }
    //Bestimmung wieviel das Gebäude pro Tick an Ressourcen hinzufügt:
    public void tick() {
        now = this.now();
        if (now.getTimeInMillis() - lastTime >= 3000) {
            System.out.println(now.getTimeInMillis() - lastTime);
            map.getZentralLager().nahrung += 10 * map.resourceFactor;
            lastTime = this.now().getTimeInMillis();
        }
    }
}

```

Försterei.java

```

package mrIndustry.SpielObjekte.Rohstoffgebaeude;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.Rohstoffgewinnung;

public class Försterei extends Rohstoffgewinnung {
    public Försterei(int locX, int locY, boolean drag, final Spielkarte map) {
        super();
        this.setMap(map);
        //Bestimmung der Position:
        this.setLocX(locX);
        this.setLocY(locY);
        //Bestimmung ob das Objekt dem Mauszeiger folgen soll oder nicht:
        this.setDragDrop(drag);
        //Bestimmung welche Texturen auf den verschiedenen Ebenen geladen
        werden soll:
        this.loadTexture("grass.png", 0);
        this.loadTexture("försterei.png", 0);
    }
    //Bestimmung wieviel das Gebäude pro Tick an Ressourcen hinzufügt:
    public void tick() {
        now = this.now();
        if (now.getTimeInMillis() - lastTime >= 3000) {

```

```

        map.getZentralLager().holz+=10*map.resourceFactor;
        lastTime = this.now().getTimeInMillis();
    }
}

Baeckerei.java
package mrIndustry.SpielObjekte.Verarbeitungsgebäude;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.Verarbeitung;

public class Baeckerei extends Verarbeitung {
    public Baeckerei(int locX, int locY, boolean drag, final Spielkarte map) {
        super();
        this.setMap(map);
        //Bestimmung der Position:
        this.setLocX(locX);
        this.setLocY(locY);
        //Bestimmung ob das Objekt dem Mauszeiger folgen soll oder nicht:
        this.setDragDrop(drag);
        //Bestimmung welche Texturen auf den verschiedenen Ebenen geladen
        werden soll:
        this.loadTexture("grass.png", 0);
        this.loadTexture("Baeckerei.png", 0);
        //Bestimmt die Geschwindigkeit des Ticks für das Gebäude:
        lastTime = this.now().getTimeInMillis();
    }

    public void tick() {
        now = this.now();
        if (now.getTimeInMillis() - lastTime >= 3000) {
            if (map.getZentralLager().holz >= 1) {
                map.getZentralLager().holz -= 1;
                map.getZentralLager().nahrung += 1 * map.resourceFactor;
            }
            lastTime = this.now().getTimeInMillis();
        }
    }
}

```

Werkzeugfabrik.java

```
package mrIndustry.SpielObjekte.Verarbeitungsgebaude;

import mrIndustry.Spielkarte;
import mrIndustry.SpielObjekte.Rohstoffgewinnung;

public class Werkzeugfabrik extends Rohstoffgewinnung {
    public Werkzeugfabrik(int locX, int locY, boolean drag, final Spielkarte
map) {
        super();
        this.setMap(map);
        //Bestimmung der Position:
        this.setLocX(locX);
        this.setLocY(locY);
        //Bestimmung ob das Objekt dem Mauszeiger folgen soll oder nicht:
        this.setDragDrop(drag);
        //Bestimmung welche Texturen auf den verschiedenen Ebenen geladen
werden soll:
        this.loadTexture("grass.png", 0);
        this.loadTexture("werkzeugfabrik.png", 0);
        this.setMap(map);
    }
}
```