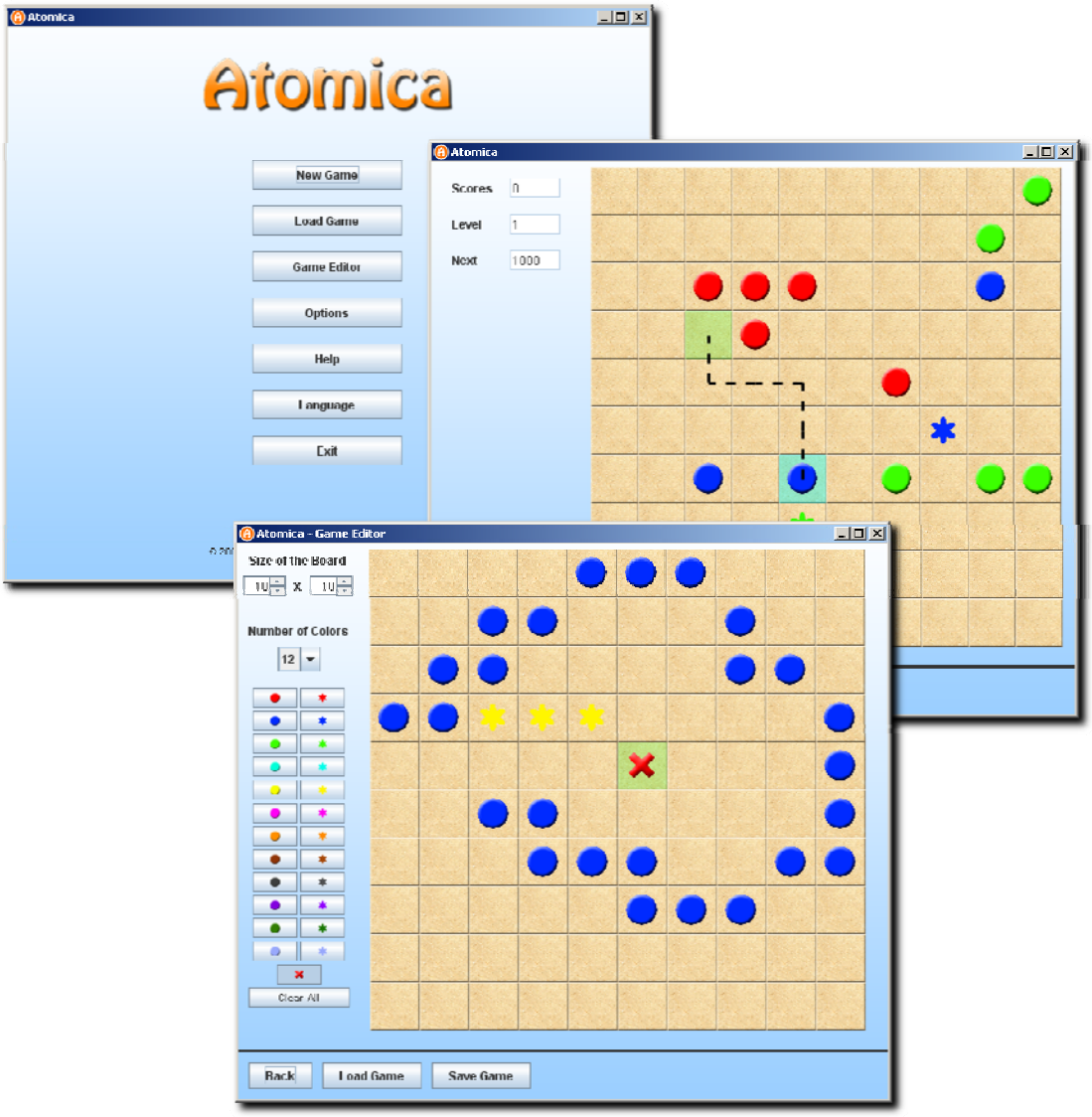
Atomica

Programmüberblick



Beschreibung des im Rahmen des Kurses 1580 Programmierpraktikum an der FernUniversität in Hagen erstellten Programms Atomica.

Georg Ludewig, Matrikelnummer 7321511

Inhaltsverzeichnis

1. Einleitung

Das Spiel Atomica startet mit einem Frame, welcher alle zentralen Navigationsmöglichkeiten des Spiels beinhaltet. Neben den in der Aufgabenstellung geforderten Spiel selbst, dem Konfigurationsdialog und dem Atomica Editor, finden sich außerdem die Option zum Einstellen der Sprache sowie das Öffnen der Hilfe.

Die Architektur wurde entsprechend der Aufgabenstellung aufgebaut. Es findet sich die drei Pakete für Benutzerschnittstelle, Anwendungslogik und Anwendungsobjekte. Zusätzlich wurde ein viertes Hilfspaket eingeführt, welches ein projektübergreifendes Logging ermöglicht. Da der komplette Quellcode inkl. Kommentare in Englisch gehalten wurde, sind nicht die vorgeschlagenen deutschen Namen verwendet worden sondern die englische Übersetzung:

Architekturkomponente	*Java Paket in Atomica*
Benutzerschnittstelle	`atomica.ui`
Anwendungslogik	`atomica.logic`
Anwendungsobjekte	`atomica.objects`
Logging	`atomica.util.logging`

2. Das Datemodell (atomica.objects)

Zentrale Komponente der Anwendungsobjekte ist die Klasse `GameSituation`. Sie wird sowohl vom Spiel als auch vom Atomica Editor verwendet und hält den Aufbau und die Belegung des Spielbrettes.

Jedes Feld wird durch ein Objekt der Klasse `Field` repräsentiert. Atome werden durch die Klasse `AtomToken` und Indikatoren durch die Klasse `IndicatorToken` repräsentiert, die gemeinsame Basisklasse `Token` ermöglicht dabei die gemeinsame Behandlung als feldbelegendes Element. Die Klasse `Field` hält dafür eine Instanzvariable dieser Klasse. Diese Assoziation ist rekursiv, das heißt jedes `Token` hält auch eine Referenz des Feldes auf dessen es sich befindet.

Die Klasse `Board` definiert die Größe des Spielbrettes. Sie hält die Anzahl der Spalten und Zeilen eines Spielbrettes, sowie dessen Default, Maximum und Minimum Werte.

Die Spielkonfigurationen (Größe des Spielbrettes, Basispunktezahl, Levels) werden durch die abstrakte Klasse `GameSettings` repräsentiert, von welcher es jeweils eine Spezialisierung für die Defaulteinstellungen eines Spiels (`DefaultGameSettings`) und eine Spezialisierung für den Editor gibt (`EditorGameSettings`). Der Grund für die Unterscheidung liegt zum einen in der unterschiedlichen Initialisierung der Einstellungen (im Editor Modus gibt es nur ein Level), zum anderen in der Möglichkeit beide Einstellungen über `java.util.prefs.Preferences` einfach zu speichern, so dass die Einstellungen bei einem Neustart des Programmes wieder vorhanden sind.

Die Klasse `Application` hält lediglich die Information, welche Sprache momentan verwendet wird sowie eine Referenz auf ein Objekt der Klasse `EditorGameSettings` und der `DefaultGameSettings`. Beim Starten des Programmes werden diese aus den Preferences geladen bzw. beim Beenden in die Preferences gespeichert. Änderungen an den Einstellungen durch den Benutzer werden auf diesen Instanzen durchgeführt.

3. Die Spiellogik (atomica.logic)

Atomica Spiel

Das Spiel selbst wird in der Anwendungslogik repräsentiert durch die Klasse `Game`. Ein Spiel kann entweder neu gestartet werden mit den Default Einstellungen (`DefaultGameSettings`) oder durch Laden einer initialen Spielsituation (`GameSituation`). Dementsprechend gibt es zwei Konstruktoren.

Die Klasse `Game` hält als Instanzvariable die Spielsituation (`GameSituation`), jede Änderung der Feldbelegung wird in ihr gespeichert. Mit dieser Information ist `Game` in der Lage, Operationen auf dem Spielbrett auszuführen. So enthält die Klasse sämtliche Methoden welche relevant sind für den Spielablauf. Diese umfassen insbesondere das Verschieben von Atomen, das Berechnen der Punkte, das Überprüfen der Leveländerung, das Platzieren neuer Indikatoren in jeder Runde sowie das Transformieren der bestehenden.

Die Klasse `Game` hält außerdem eine Instanzvariable der Klasse `PathFinder`, welche in der Lage ist durch Angabe einer Spielsituation (`GameSituation`) und eines Start- und Endfeldes den kürzesten Pfad auf Basis des A*Star Algorithmus zu finden. Das Ergebnis wird in Form eines `Path` Objektes zurückgegeben, welches lediglich eine Liste der zu überquerenden Felder beinhaltet.

Das Auffinden der Moleküle wird von der Klasse `Game` aus Gründen der Wiederverwendung im Editor an die Klasse `MoleculeDetector` delegiert. Ähnlich wie bei der Klasse `PathFinder`, ist hier die Übergabe der Spielsituation notwendig (`GameSituation`). Ein Objekt der Klasse `MoleculeDetector` durchsucht dann die gesamte Spielsituation nach größtmöglichen Molekülen und gibt diese zurück. Ein Molekül wird dabei durch die Klasse `Molecule` repräsentiert.

Atomica Editor

Der Atomica Editor wird in der Anwendungslogik repräsentiert durch die Klasse `Editor`. Die Klasse erlaubt das Anordnen von Atomen und Indikatoren innerhalb einer Spielsituation. Zum Beispiel validiert sie bereits belegte Felder, überprüft ob durch die gewünschte Belegung des Benutzers ggf. ein Molekül entsteht (via `MoleculeDetector`) oder erlaubt bspw. das Löschen aller `Token` einer Spielsituation.

Ein Objekt der Klasse Editor kann sich entweder im `ADD`, `MOVE` oder `DELETE` Modus befinden, je nachdem ob der Benutzer Token hinzufügen, verschieben oder Löschen möchte.

De-/Serialisierung und Persistenz

Spielsituation können sowohl aus dem Editor, als auch aus dem Spiel heraus persistent als Binärdatei gespeichert werden (Endung „.atomica“). Diese Spielsituation können entweder als zu spielendes Spiel oder als zu bearbeitendes Spiel im Editor geladen werden. Diese Aufgabe übernimmt die Klasse `GameSituationSerializer`. Sie wird sowohl von `Editor` als auch von `Game` verwendet.

Folgende Klassen wurden über das Marker Interface Serializable gekennzeichnet und werden zur Speicherung der Spielsituation verwendet: `GameSituation`, `GameSettings`, `Field`, `Board`, `AtomToken` und `IndicatorToken`.

4. Die Benutzerschnittstelle (atomica.ui)

Übersicht

Aus der Aufgabenstellung wurden 4 wesentliche Szenarien für Benutzerinteraktion extrahiert – das Hauptmenu, das Spiel selbst, der Editor und der Konfigurationsdialog. Zusätzlich wurde eine Hilfe und die Auswahl einer Sprache integriert. Um eine gute Trennung von Verantwortlichkeiten zu erreichen wurde für jedes Szenario eine Panel- bzw. Dialog-Klasse angelegt.

Szenario	*Java Klasse in atomica.ui*
Hauptmenu (Navigationsmöglichkeiten)	`ApplicationFrame`
Atomica Spiel	`GamePanel`
Atomica Editor	`EditorPanel`
Konfigurationsdialog	`GameSettingsPanel`
Hilfe	`HelpPanel`
Sprache auswählen	`LanguageDialog`

Je nach gewählter Option wird das gewünschte Panel im `ApplicationFrame` aktiv gesetzt. Jedes der einzelnen Panels enthält in einer Navigationsleiste einen Button um zum Hauptmenu zurückzukehren.

Das Spielbrett

Das Spielbrett wird repräsentiert durch die abstrakte Klasse `BoardPanel`. Sie enthält die wesentliche Logik zur Darstellung einer Spielsituation. Die beiden Spezialisierungen `GameBoardPanel` und `EditorBoardPanel` enthalten jeweils die Implementierung zur Verwendung im Spiel selbst (`GamePanel`) und im Editor (`EditorPanel`). Als Model dient die Klasse `Game` bzw. die Klasse `Editor` aus der Anwendungslogik.

5. UML Klassendiagramm