Projekt 1

Sokobomb

Denis Simonet und Christoph Bruderer

Dozent: P. Schwab

Biel, 13.06.2013

1 Inhalt

2 Projektbeschrieb

2.1 Einleitung

Im Rahmen des Moduls Projekt 1 wird das Spiel Sokoban in der Programmiersprache Java umgesetzt. Statt der klassischen Kisten werden bei unserer Umsetzung Bomben verschoben - der Name ist passend dazu Sokobomb. Es gelten die üblichen Spielregeln: Die Bomben können von der Spielfigur nur geschoben und nicht gezogen werden, ein Verschieben mehrerer Bomben zugleich ist nicht möglich. Die möglichen Bewegungsrichtungen der Spielfigur sind nach oben, unten, rechts und links (diagonale Züge sind also nicht erlaubt). Das Ziel ist, jede Bombe in einer vorgegebenen Zeit an je eines der Zielfelder (Behälter zur sicheren Detonation) zu schieben.

2.2 Spielfeld parsen

Spielfelder werden mit der verbreiteten Sokoban-Ascii-Notation eingelesen. Dabei wird eine Wand durch das Symbol # dargestellt, die Spielfigur (Sokoban) am Startfeld durch @, ein Zielfeld als ., die Startpositionen der Kisten als $, eine auf einem Zielfeld stehende Kiste als * und die Spielfigur auf einem Zielfeld als +. Da eine zeitliche Komponente dazu kommt wird noch die Möglichkeit für Metainformationen geschaffen (es muss gespeichert werden, wie viel Zeit man insgesamt zur Verfügung hat und wie viel Zeit schon vergangen ist).

2.3 Spielfeld zeichnen

Um grösstmögliche Kompatibilität zu erreichen, wollen wir die grafische Oberfläche mit OpenGL umsetzen. Das eingelesene Spielfeld wird also über OpenGL dargestellt (die Bewegung funktioniert folglich auch über diese Library).

2.4 Interaktivität

Mit den Pfeiltasten soll die Spielfigur über das Feld bewegt werden können. Dabei müssen die Mauern unüberschreitbar sein und Bomben müssen sich mitbewegen, wenn man vor einer steht und dieser keine Mauer im Weg steht. Ausserdem ist eine Undo-Funktion vorgesehen, die beliebig viele Schritte rückgängig machen kann. Ein Button zum neu starten wird auch umgesetzt.

2.5 Menu

Es gibt ein Menu, die Inhalte müssen noch definiert werden.

2.6 Path finding

Es soll eine Möglichkeit geben, auf ein Feld zu klicken, so dass sich die Spielfigur selbstständig an diese Stelle bewegt (sofern ein Pfad dorthin existiert). Hintergrund dieser Funktion ist, dass es insbesondere bei grossen Feldern mühsam ist, sich von Feld zu Feld an eine entfernte Position zu bewegen. Für die Realisierung wird der Dijkstra-Algorithmus umgesetzt.

2.7 Android

Als letzten Schritt wollen wir versuchen, die Applikation auf Android zum Laufen zu bekommen und diese für Smartphones zu optimieren. Der Umfang dieses letzten Schritts hängt stark davon ab, wie rasch wir bei den vorhergehenden Teilproblemen vorankommen. Bei dieser Portierung muss berücksichtigt werden, dass sich die Benutzereingabe über einen Touch-Screen und wenige zusätzliche Buttons statt über die Tastatur gestaltet.

3 Level Files

Die Grundlage für das Spiel bilden die verschiedenen Levels, welche als .txt Dateien abgespeichert werden. Eine Level Datei besteht aus einer Zeile mit den Metadaten Titel des Levels und zur Verfügung stehende Zeit und aus der ASCII Darstellung des eigentlichen Levels. Diese beiden Teile der Datei werden separat geparst und danach das aufgrund der Datei erstellte Level geladen.

Aus der folgenden ASCII Datei:

```
title=Level 1;time=100
  #########
  #       .#
###  $$$$ #
#.         .#
#.     @####
#########  #
        ####
```

Wird dieses Level aufbereitet:

In der ersten Zeile wird der Titel „Level 1“ und die verbleibende Zeit angezeigt. Darunter Befindet sich das Spielfeld

4 Menus

Da das ganze Spiel in OpenGL angezeigt wird war es nicht möglich wie z.B. in java.awt schon bestehende Module und Methoden für ein Menu zu verwenden. Ein Teil der Arbeit bestand deshalb darin die ganze Menu Logik zu implementieren. Menus können an verschiedenen Orten (am Anfang oder in den Pausen) zum Einsatz kommen und immer andere Einträge und Funktionen haben. Deshalb haben wir eine Menu Klasse erstellt welche die ganze Logik der Menus verwaltet, diese Menus werden bei Bedarf von einem State erstellt. Für die einzelnen Menu Einträge gibt es daneben noch eine Klasse MenuItem.

Beispiel vom Start Menu:

5 Spiel Modus

Der Spiel Modus ist natürlich das wichtigste Stück der Anwendung, hier kann die Spielfigur mit den Pfeiltasten hin und her bewegt werden oder mittels Mausklick an einen Ort geschickt werden sofern dieser erreichbar ist. Daneben sin folgende Funktionstasten implementiert:

- Pausieren [ESC]
- Level zurücksetzen [R]
- Zug rückgängig machen [U]

6 Level Design Modus

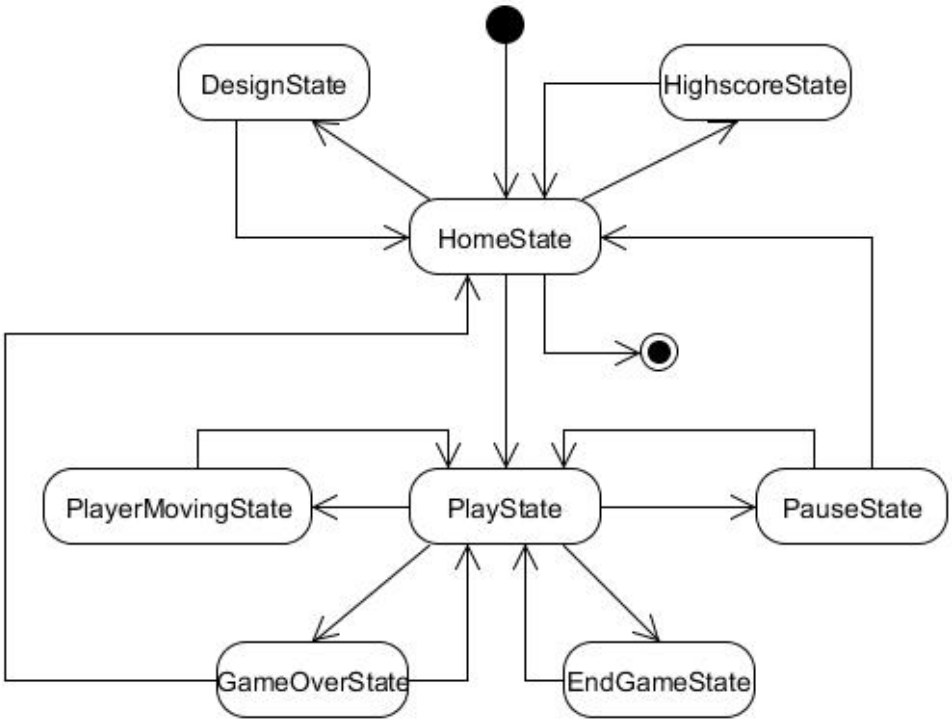
Vom Start Menu aus kommt man durch auswählen des entsprechenden Menu Punkt in den Design Modus. Das Ziel des Design Modus ist es neue Levels zu erstellen. Der Design Modus ist noch nicht vollumfänglich implementiert, im Moment sind folgende Funktionen verfügbar welche über die entsprechenden Tasten gesteuert werden:

- Mauer zeichnen [1]
- Boden zeichnen [2]
- Ein Feld entfernen [0]
- Undo Funktion [U]
- Reset Funktion [R]
- Verlassen des Design Modus [ESC]

Als Erweiterung könnten diese Funktionalitäten zusätzlich über ein Menu verfügbar gemacht werden, daneben brauchte es natürlich noch entsprechende Funktionen um ein erstelltes Level abzuspeichern.

7 State Diagramm

Um den ganzen Ablauf des Programms übersichtlich zu halten haben wir ein state pattern implementiert. Ein State Element speichert die notwendigen Informationen für den jeweiligen Status und verarbeitet auch die Benutzer Eingabe die während des momentanen Status erfolgen. Hier sind die verschiedenen States und Übergänge abgebildet.



8 Übersicht der Klassen

Es folgen Hintergrunderklärungen zu den einzelnen Klassen. Im Anhang befindet sich ein Javadoc-Export, der weitere Details verrät.

- **Application:** Ermöglicht den Zugriff auf die Controller sowie auf die Datenbank.

8.1 Command

Um es zu ermöglichen, beispielsweise einen Solver zu bauen, werden Bewegungen über Commands ausgelöst. So kann z.B. ein Solver gebaut werden, der eine Liste von Commands liefert, die Schritt für Schritt ausgeführt werden müssen, um das aktuelle Level zu lösen.

- **Command:** Abstrakte Klasse, definiert die Methoden execute und undo.
- **MoveCommand:** Abstrakte Klasse, erbt von Command. Grundlage für Bewegungs-Commands, in unserem Fall für Bomben und Spieler.
- **BombMoveCommand:** Erbt von MoveCommand, implementiert die Bewegung für eine Bombe.

- **PlayerMoveCommand:** Erbt von MoveCommand, implementiert die Bewegung für den Spieler.

8.2 Controller

Die Controller stellen Singleton-Instanzen zur Verfügung und ermöglichen es, von überall her Befehle auszuführen.

- **FieldController:** Initialisiert und verwaltet das aktuelle Field und die Fieldhistory
- **MainController:** Startet das Programm, definiert das Field und initialisiert das OpenGL.
- **OpenGLController:** Lädt die LWJGL Bibliothek, initialisiert und zeichnet das Spielfeld, erfasst die aktuelle Position der Maus auf dem Spielfeld
- **StateController:** Initialisiert und Verwaltet den aktuellen State, implementiert die Methoden zum Zeichnen der States und zur Erfassung und Bearbeitung der User Eingaben.

8.3 Datamapper

- **Datamapper:** Stellt die Verbindung mit der SQL-DB her und stellt die Methoden zur Verfügung um die Highscore-Verwaltung zu ermöglichen.

8.4 Exception

- **InvalidCoordinateException:** Signalisiert den Zugriff auf eine Koordinate, die ausserhalb des Spielfelds liegt.
- **LexerException:** Fehler im Lexer werden über diese Exception signalisiert.
- **NoNextLevelException:** Wenn es keine Levels mehr gibt.
- **NoSolutionFoundException:** Wird im Solver verwendet.
- **OutOfBoundException:** Zugriff auf ein nicht existierendes Element in einem Container.

8.5 Field

Ein Field beinhaltet die notwendigen Methoden und Attribute, um ein Spielfeld darzustellen. In unserem Fall gibt es das PlayField und das DesignField. Ersteres stellt ein spielbares Feld dar. Zweiteres ist der Level-Design-Modus.

- **Field:** Abstrakte Klasse, erstellt das Spielfeld entsprechend dem geparsten Level-File, verwaltet die aktuellen Positionen von Bomben und Spieler auf dem Spielfeld.
- **FieldCache:** Um einen Zugriff auf Objekte im Feld mit O(1) zu gewähren, gibt es diesen Cache. Er ist insbesondere für Dijkstra praktisch.
- **FieldHistoryItem:** Speichert die Positionen von Bomben und Spieler auf dem Spielfeld, damit diese wieder Rückgängig gemacht werden können.
- **DesignField:** Das Feld für den Designmodus.
- **PlayField:** Initialisiert ein neues Spielfeld, verwaltet die Levels.

8.6 Model

Klassen, die die notwendigen Informationen für einzelne Objekte enthalten.

- **Drawable:** Interface für alle Klassen welche zeichenbar sind, definiert die Methode draw.
- **HighscoreItem:** Verwaltet die Informationen für einen Highscore-Eintrag.
- **Menu:** Stellt die grundlegende Logik zum Erstellen von verschiedenen Menus zur Verfügung, die einzelnen Menueinträge werden dabei in einer LinkedList verwaltet.
- **MenuItem:** Definiert die Informationen, welche ein Menu Eintrag benötigt.
- **Header:** Verwaltet die Informationen welche während des Spiels in der Titelzeile angezeigt werden
- **Time:** Verwaltet den Timer welcher während des Spiels angezeigt wird.
- **Coordinate**
    - **Coordinate:** Koordinate in Pixeln.

        - **DeltaCoordinate:** Erbt von Coordinate, stellt ein Delta zu einer aktuellen Koordinate dar.
        - **TileCoordinate:** Erbt von Coordinate, repräsentiert die Koordinaten der Kacheln (Tile), aus welchen das Spielfeld aufgebaut ist (und nicht die Pixel).
    - Tiles
        - **Tile:** Verwaltet die Position und den Typ einer Kachel
        - **DijkstraNode:** Erweitert die Klasse Tile um die Informationen, welche der Dijkstra-Algorithmus benötigt
        - **Bomb, Floor, Player, Target, Wall:** Erweitert die Klasse DijkstraNode um die Informationen für die Grafik zu speichern.

8.7 Parser

Der Parser liest Spielfelder ein und wandelt sie in Field-Objekte um. Um die Dateien einlesen zu können, gibt es ausserdem Lexer-Klassen.

- **Lexer:** Interface für die Implementation von inhaltsspezifischen Lexern, definiert die Methoden initLexer und nextToken
- **FieldLexer:** Interpretiert die ASCII Sokoban Notation aus einem Level File und speichert die entsprechenden Tokens.
- **LevelInformationLexer:** Liest die Level Informationen aus der ersten Zeile des Level Files.
- **Parser:** Parst die Tokens die vom Lexer generiert werden.
- **Token:** Stellt die Informationen zur Verfügung welche benötigt werden um Tokens zu erstellen.

8.8 Solver

Dieses Package ist für das automatische Lösen eines Levels vorgesehen. Die Zeit für ein Solver war nicht vorhanden, es gibt aber eine Dijkstra-Klasse Zwecks Path-Finding. Dijkstra (statt z.B. eine Breitensuche) haben wir gewählt, weil der Dijkstra einfach zu implementieren ist und im Falle von Komplizierungen in den Feldern immer noch mächtig genug wäre. Uns ist bewusst, dass er für Sokoban eigentlich overkill ist.

- **Dijkstra:** Implementierung des Dijkstra-Algorithmus zum Finden des kürzesten Weges. Er wäre effizienter, wenn wir eine PriorityQueue gewählt hätten, um den nächsten billigsten und unbesuchten Node zu finden.
- **Path:** Berechnet mittels Dijkstra den kürzesten Pfad von Punkt A nach Punkt B.

8.9 State

Das State Pattern. Durch die vielen möglichen Status war es naheliegend, dieses Pattern zu implementieren. Nur so konnte eine gewisse Übersichtlichkeit gewahrt werden.

- **State:** Abstrakte Klasse, definiert die Variablen für alle States und stellt die Methoden zur Verfügung.

Alle weiteren Klassen in diesem Package erben von der Klasse State:

- **DesignState:** Verwaltet den Spielfeld-Editor-Modus.
- **EndGameState:** Wird am Ende des letzten Levels aufgerufen.
- **FieldSolvingState:** Verwaltet den Solver-Modus.
- **HighscoreState:** Stellt die Highscore dar.
- **HomeState:** Stellt das Home-Menu dar.
- **PauseState:** Stellt das Pausen-Menu dar.
- **PlayerMovingState:** Verschiebt den Spieler schrittweise auf ein anderes Feld (bei Mausklick, siehe Klasse Path).
- **PlayState:** Der Spielmodus erfasst und verarbeitet die Benutzereingaben während des Spiels.

- **WonState:** Wird nach abschliessen eines Levels angezeigt.
- **GameOverState:** Wird angezeigt wenn die Zeit in einem Level abgelaufen ist.

8.10 Util

Alles, was sonst nirgendwo gepasst hat.

- **Levels:** Hier werden die Levelinformationen verwaltet.
- **OpenGLLoader:** Hier werden die notwendigen LWJGL Bibliotheken geladen.
- **Tiles:** Enthält die Grösse der Kacheln sowie die Pfade der Grafiken.