TECHNISCHE UNIVERSITÄT
CHEMNITZ

Bomberman

Analyse und Design

Erstellt von:

DONGER SoftwareEngineering

(Gruppe 7)

Inhalt

1. Vorwort

Im Rahmen des Softwarepraktikums an der Technischen Universität Chemnitz an der Professur Softwaretechnik sollte eine kleine Software von jeder Praktikumsgruppe entwickelt werden. Vom Pflichtenheft der zu erstellenden Software bis zum fertigen Produkt wurde alles von den Studenten des Praktikums erarbeitet. Ziel des Praktikums war es am gesamten Entwicklungsprozess einer Software mitzuarbeiten und diesen von Anfang an erlebt zu haben. Alle Details der zu erstellenden Software wurden uns in Form eines Lastenheftes erklärt. Es sollte ein kleines Spiel nach dem Klassiker „Bomberman“ werden.
Im Nachfolgendem wird gezeigt wie unsere Gruppe von der Planung und Analyse über das Design bis hin zur Implementierung die Software „Bomberman“ realisiert hat.

2. Planung und Analyse

2.1. Anforderungen

Zunächst hat unsere Gruppe das Lastenheft gründlich analysiert und alle funktionalen und nicht-funktionalen Anforderungen der zu entwickelnden Software herauskristallisiert. Im Folgendem finden Sie alle Anforderungen aufgelistet. Sofern kein Vermerk an der Anforderung steht wurde diese erfüllt.

Funktionale Anforderungen:

1. Der Verwaltungsserver kann die Spielserver anzeigen.(muss)
2. Der Client kann Spielserver auswählen und sich mit ihnen verbinden.(muss)
3. Der Client kann einen Namen auswählen.(muss, impliziert 2.)
4. Der Spielserver beugt doppelten Namen vor.(muss, impliziert 3.)
5. Unser Client funktioniert mit unserem Spielserver.(muss)
6. Unser Client ist mit anderen Spielservern kompatibel.(soll)
7. Unser Server ist mit anderen Clienten kompatibel.(soll)
8. Das Spiel kann gestartet werden.(muss)
9. Die Spieldaten werden korrekt zum Clienten übertragen.(muss)
10. Ein Bomberman kann laufen.(muss)
11. Ein Bomberman kann nicht über Bomben laufen.(muss)
12. Ein Bomberman kann nicht über Mauern laufen.(muss)
13. Ein Bomberman kann Bomben legen.(muss)
14. Ein Bomberman darf nicht zu viel Bomben legen.(muss)
15. Ein Bomberman kann Boni sammeln.(muss)
16. Ein Bomberman kann sterben.(muss)
17. Ein Bomberman mit Rüstung wird unverwundbar. (muss, impliziert 16.)

18. Ein Bomberman wird nach Ablauf der Rüstung wieder verwundbar. (muss, impliziert 17.)
19. Die Bomben haben den korrekten Explosionsradius des Spielers der sie platziert hat.(muss)
20. Bomben können brüchige Wände zerstören.(muss)
21. Bomben können feste Wände nicht zerstören.(muss)
22. Explodierende Bomben lösen Kettenreaktionen von Explosionen aus.(muss)
23. Ein Spieler mit Beschleunigungs-Powerup läuft schneller. (soll, Modus B)
24. Ein Spieler mit Tritt-Powerup kann Bomben wegstoßen. (soll, Modus B)
25. Getretene Bomben stoppen bei Kontakt mit Objekten. (soll, Modus B)
26. Die Explosionen der Superbombe wird nicht durch Mauern gestoppt. (soll, Modus B)
27. Das Maximum-Radius-Powerup setzt den Explosionsradius auf das Maximum.(soll, Modus B)
28. Ein Spieler mit Bombenläufer-Powerup kann Bomben überqueren. (soll, Modus B)
29. Inaktive Spieler explodieren nach 30s.(muss)
30. Die KI kann auf Wunsch eingeschaltet werden.(soll)
31. Die KI versucht Powerups zu sammeln.(soll, impliziert 30.) (nicht erfüllt)
32. Die KI schließt sich nicht selbst ein.(soll, impliziert 30.) (nicht erfüllt)
33. Die KI tötet sich nicht selbst.(soll, impliziert 30.) (nicht erfüllt)
34. Die KI läuft aus Bombenradien heraus.(soll, impliziert 30.) (nicht erfüllt)
35. Die KI erkennt Kettenreaktionen von Bomben.(soll, impliziert 30.) (nicht erfüllt)
36. Die KI versucht Gegner zu töten.(soll, impliziert 30.) (nicht erfüllt)
37.

38. Die KI versucht Kettenreaktionen von Explosionen zu nutzen.(soll, impliziert 30.) (nicht erfüllt)
39. Das Ranking wird korrekt berechnet.(soll)
40. Das Ranking wird korrekt angezeigt.(soll)
41. Unser Client ist mit den Servern anderer kompatibel.(soll)
42. Unser Server ist mit den Client anderer kompatibel.(soll)
43. Besiegte Spieler können dem Spiel zuschauen.(muss)
44. Brüchige Mauern werden bei Spielstart generiert.(muss)
45. Powerups werden unter brüchigen Mauern generiert.(muss)
46. Powerups sind nicht sichtbar bevor die Mauer zerstört ist.(muss)
47. Spieler der alten Partie werden in die Lobby des nächsten Spieles eingefügt.(soll)
48. Am Server können Spieleinstellungen vorgenommen werden.(soll)

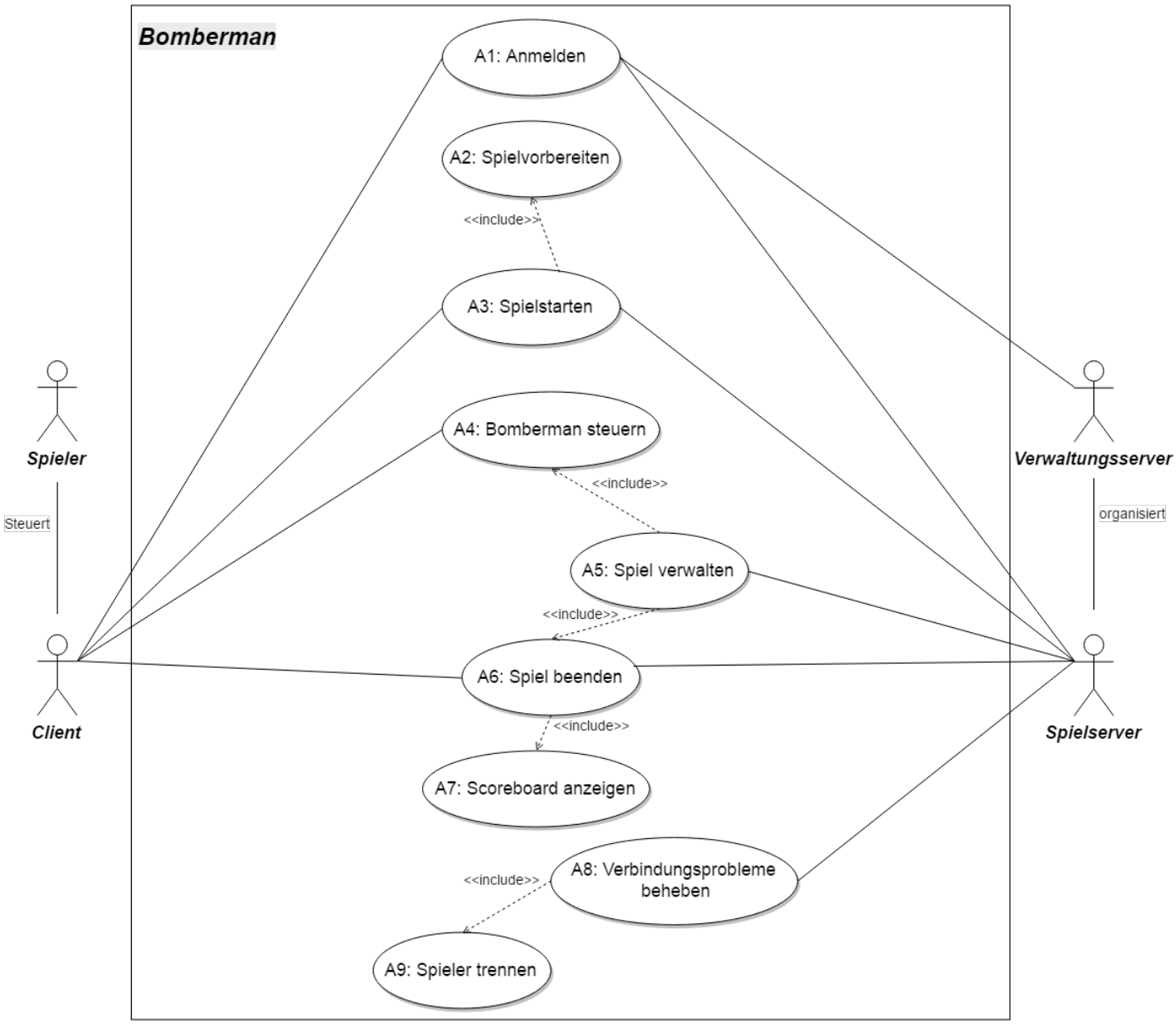
1.2. Nicht-funktionale Anforderungen

49. Das Handbuch steht als PDF-Datei zur Verfügung.(muss)
50. Das Handbuch ist grafisch visualisiert.(kann)
51. Die Fehlermeldungen klären aussagekräftig und verständlich über Probleme auf(soll)
52. Der Client erreicht mindestens 15 fps.(muss)
53. Der Client erreicht mindestens 30 fps.(soll, impliziert 4.) (nicht erfüllt)
54. Der Client erreicht mindestens 60 fps.(kann, impliziert 5.) (nicht erfüllt)
55. Die Verbindung wird mindestens 15 mal pro Sekunde aktualisiert. (muss)
56. Die Verbindung wird mindestens 30 mal pro Sekunde aktualisiert. (soll impliziert 7) (nicht erfüllt)
57. Die Verbindung wird mindestens 60 mal pro Sekunde aktualisiert. (kann impliziert 8) (nicht erfüllt)
58. Das Programm Bomberman wird mit der Programmiersprache Java programmiert.(muss)
59. Der Programmcode wird auf englisch verfasst.(muss)

60. Die Codebegleitenden Kommentare werden auf deutsch verfasst. (soll)
61. Kommentare werden mittels Javadoc Format erstellt.(soll)
62. Programmtests werden mittel Junit durchgeführt.(soll)

2.2. Use-Case-Szenarien

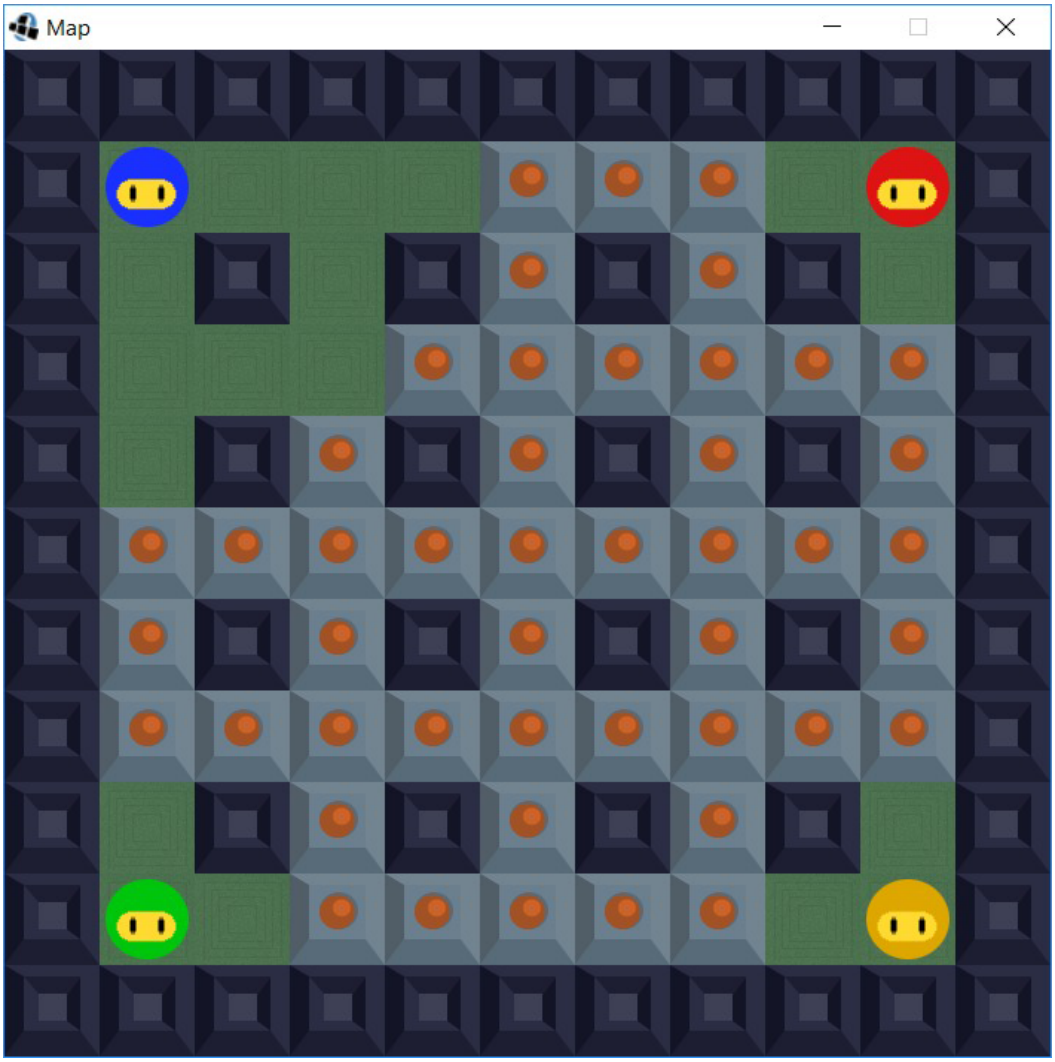
Außerdem haben wir die verschiedenen Möglichkeiten die Software zu benutzen in Use-Case-Diagrammen dargestellt um alle Eventualitäten abzudecken.



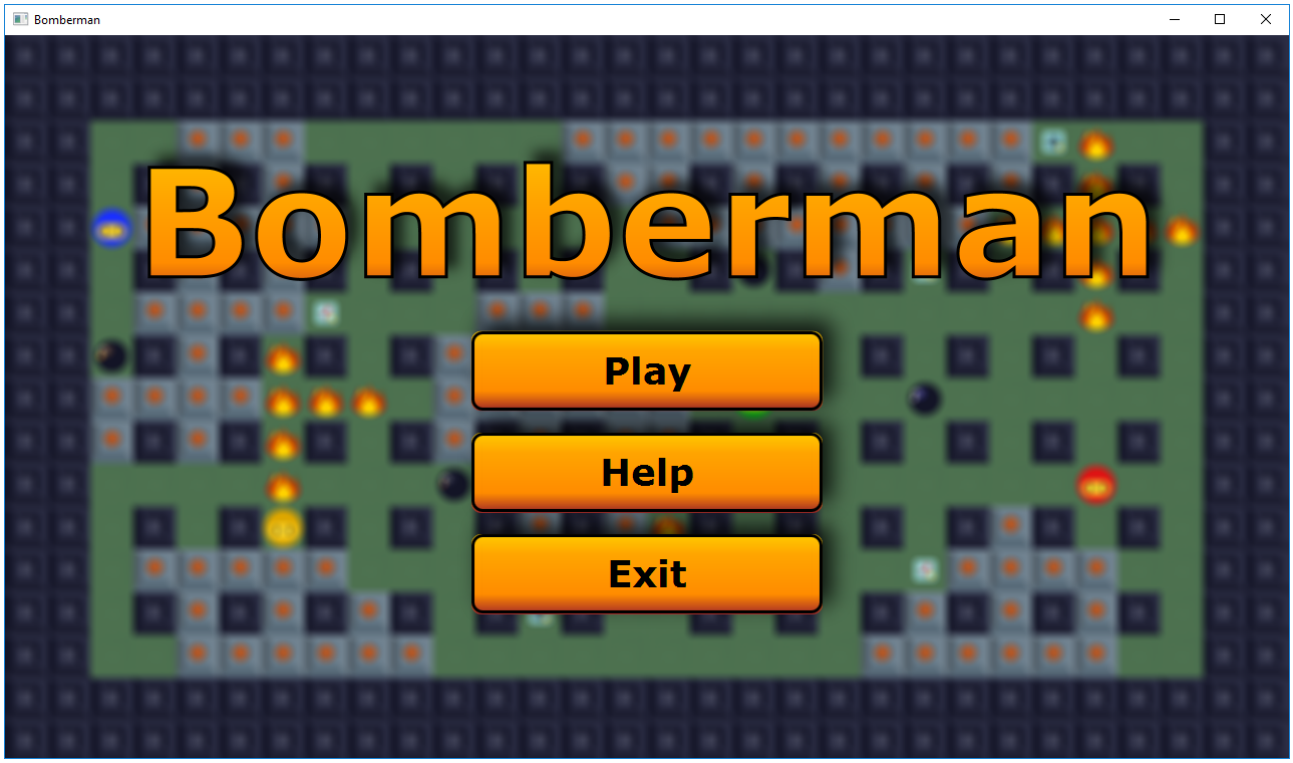
Die einzelnen Szenarien finden Sie im Pflichtenheft.

2.3. Grafische Benutzeroberfläche

Als grafische Benutzeroberfläche haben wir uns für ein einfaches grafisches Menü mit den nötigsten Bedienelementen entschieden, welches mit der Maus bedienbar ist. Im Spiel selbst öffnet sich ein separates Fenster mit alles nötigen Anzeigen.



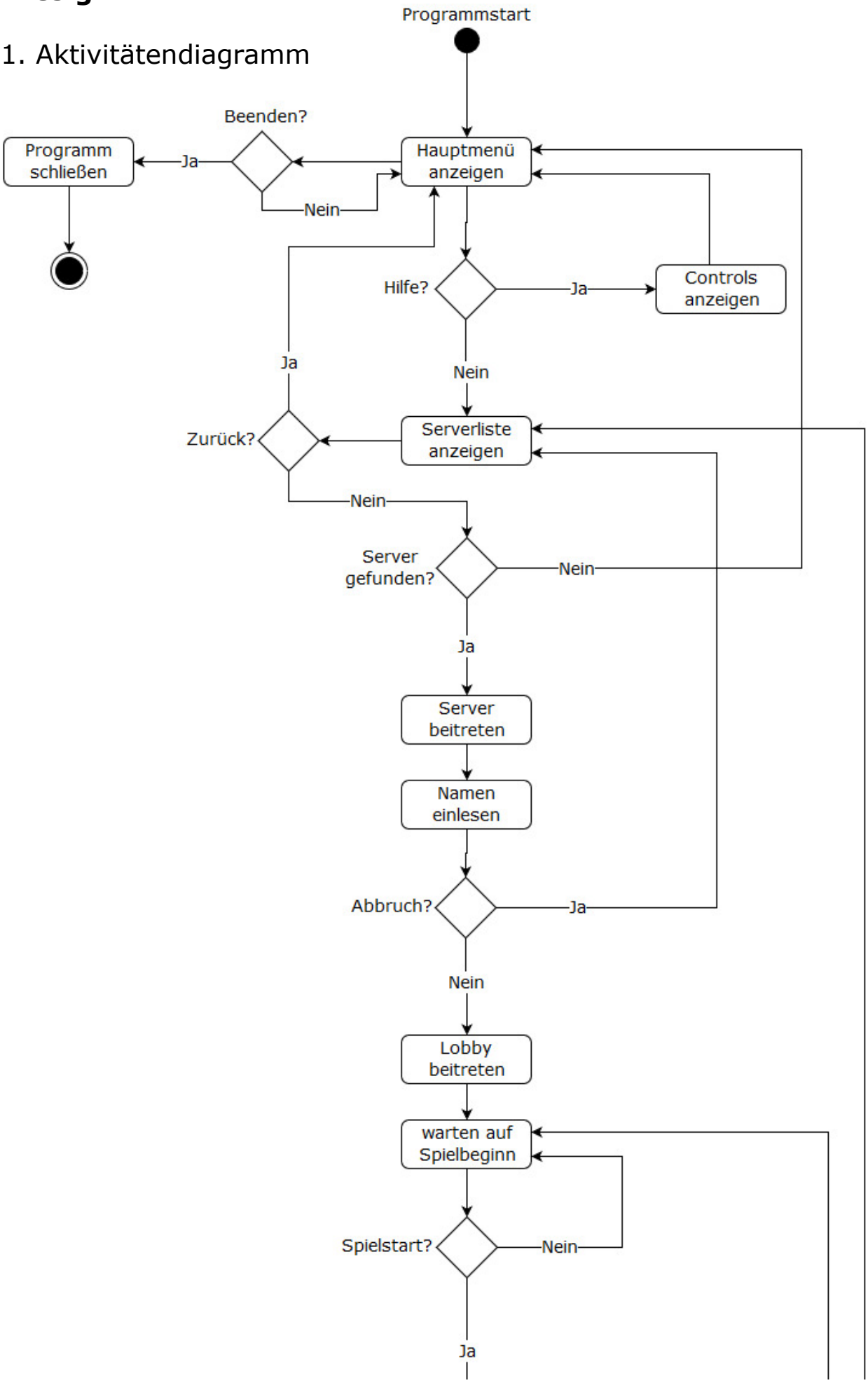
(im Spiel)

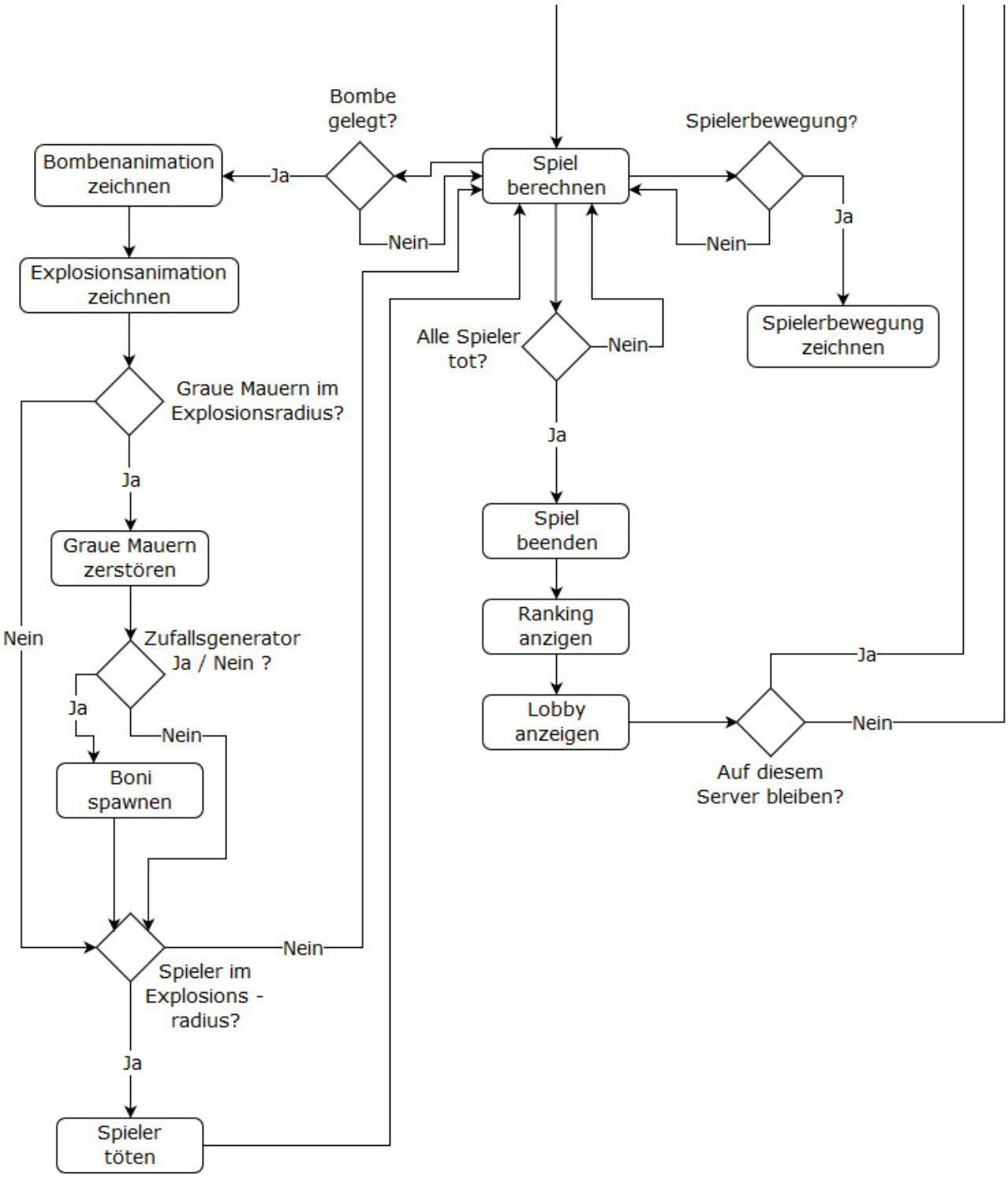


(im Menü)

3. Design

3.1. Aktivitätendiagramm





Dieses Aktivitätendiagramm deckt alle Aktivitäten der Software ab. Man sieht den gesamten Verlauft vom starten des Programms, über das Menü und den Spielverlauf, bis zum beenden.

3.2. Architektur

Die Softwarearchitektur von unserem Bomberman haben wir in einem UML-Klassendiagramm festgehalten. Hier werden alle Klassen, Attribute, Methoden und Verbindungen der Software übersichtlich dargestellt.

Das Klassendiagramm finden Sie in einer separaten JPEG Datei.

3.3. Tests

Des weiteren haben wir uns mit den Tests auseinandergesetzt. Dazu haben wir zu einigen relevanten Methoden JUnit Tests geschrieben. Jedoch sind Tests im gesamten Entwicklungsprozess etwas untergegangen. Weitere Methoden-Tests und Systemtests werden in zukünftigen Updates folgen.

4. Implementierung

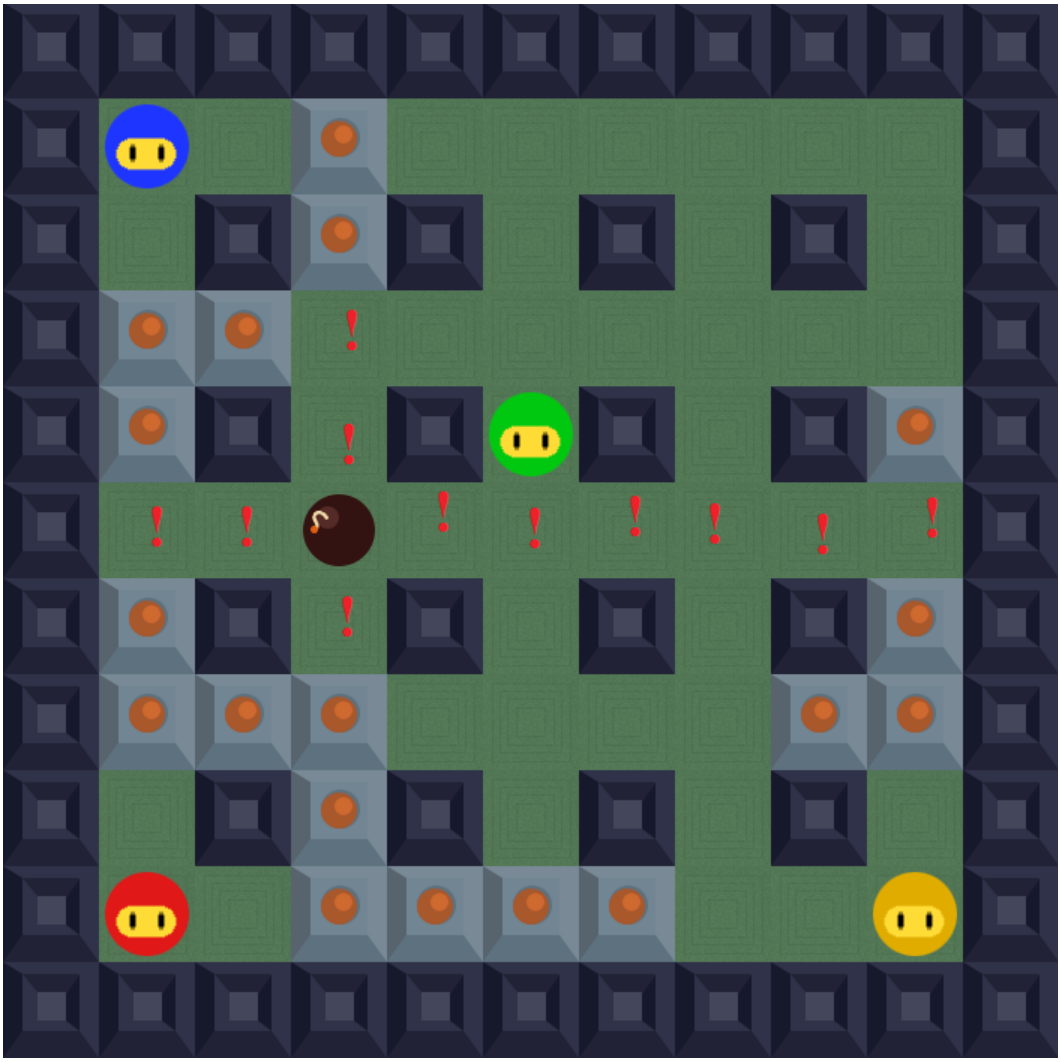
4.2. Spiel

Es wurde zu Beginn das Bombermanspiel offline implementiert, um es dann später in die Server-Client-Architektur einzubinden. Dabei wurde auf Standartbibliotheken von Java (z.B. Java AWT) zurückgegriffen, um bestimmte Funktionen wie Zeichnen der Matrix oder Tastatureingaben über einen Keymanager, etc. zu realisieren.
Zuerst wurde Version A des Spieles realisiert und grundlegende Funktionen des Bombermans wie Laufen, Bombe legen, Mauern zerstören usw. implementiert. Außerdem kamen die ersten Boni dazu
(Armor, größerer Explosionsradius, mehr Bombern legen).
Schließlich wurde die Variante B noch hinzugefügt, was fünf weitere Boni beinhaltet.

4.2. Künstliche Intelligenz

Die Ki Besteht aus 3 Hauptfunktionen „AIMain", die den Hauptablauf der KI abarbeitet, „GenerateMap", die eine Karte, mit Gefahren zur Navigation und „DetermineAction", die auf Basis der kreierten Karte bestimmt welches angrenzendes Feld das sicherste ist und ob Bomben gelegt werden. Zusätzlich gibt es noch 5 Testfunktionen die Informationen über die Karte geben.
„CheckDanger" prüft nach Bomben oder Explosionen auf Feld r, c und „CheckTop"(Bot, Left und Right jeweils mit anderer Richtung)

prüft ob das Feld oberhalb Betretbar und Sicher ist.
„GenerateMap“ sucht die Karte nach Bomben ab und markiert alle Felder zu denen die Explosion dieser Bombe gelangen würden mit Gefahr(im Code ID „999“; siehe Abbildung Ausrufezeichen).
DetermineAction prüft je nachdem ob der Bomberman sich in Gefahr befindet oder nicht an welchem angrenzendem Feld die meisten sicheren Felder liegen und schickt den Bomberman dort hin. Falls er sicher sein sollte prüft die Funktion ob der Bomberman eine Bombe legen kann und sich an einen sicheren Ort zurück ziehen kann, falls ja legt sie eine Bombe.



Markierung der Gefahren

4.3 Spielserver

Der Spielserver ist eine zentrale Einheit des gesamten Spiels. Dieser startet das Spiel und ermöglicht eine Interaktion der einzelnen Spieler über das Spielfeld. Die Implementierung erfolgt als Klasse „BombermanGameServer," deren Implementierung im folgenden Abschnitt kurz erläutert werden soll.

Hauptmethoden:

Sobald eine Instanz der Klasse „BombermanGameServer" angelegt ist werden mehrere Funktionen ausgeführt.

Als erstes startet die Funktion „startBombermanGameServer()". In dieser Funktion werden alle wichtigen Größen angelegt die der Spielserver zur Kommunikation mit den Clients braucht. Dazu gehören das ServerSocket „socketBombermanGameServer", welches den Verbindungsaufbau zu den Clients ermöglicht. Dann wird die LinkedList „msgQueue" angelegt. Diese beinhaltet die „heartbeats" und die jeweils dazugehörige „ClientID" der einzelnen, mit dem Server verbundenen Clients. Die „ClientID" bekommt jeder Client über einen, quasi ihm zugewiesenen, „BombermanGameClientHandler". Diese ist mit dem „heartbeat" wichtig, damit der Server weiß welcher Client noch aktiv ist. Hört der Empfang des „heartbeat" vom Client X auf bedeutet das, dass die Verbindung nicht mehr besteht. Danach wird die „writer_list" angelegt, die alle Writer der einzelnen Clients enthält, damit ist es möglich allen Clients die relevanten Information über das

Spielgeschehen zusenden zu lassen.

Nachdem die ersten Funktion durchlaufen wurde startet nun die Funktion „listenForClients()“. Diese Funktion legt einen „ExecutorService“ an. Also einen Threadpool in den max. vier Threads passen. Damit wird sichergestellt, dass nie mehr als erlaubt Spieler sich auf dem Spielserver befinden.
Nun werden so viele Verbindungen zu Clients aufgebaut wie Spieler auf dem Spielserver zugelassen sind, also min. 2 und max. 4. Die Sockets „toClientSocket = *socketBombermanGameServer*.accept()“ werden als erstes benutzt um einen Writer zu erstellen, der dann in die „writer_list“ hinzugefügt wird. Zum Senden wird das TCP und UTF-8 benutzt. Mit TCP wird der Empfang sichergestellt und mit UTF-8 können wir keine wichtigen Daten verlieren, da wir sie als Strings verpackte Jsonobjekte senden. Dem Threadpool wird dann ein neuer Thread vom Typ „BombermanGameClientHandler“ übergeben an den wiederum das Socket und eine „ClientID“ übergeben wird. Danach wird die Variable „ClientID“ um eins erhöht. Somit hat jeder zum Spielserver gehörende Client eine eigene ID und der Spielserver weiß von wem welche Eingabe kam, und kann entsprechend handeln.
Ist die max. Anzahl der Spieler erreicht, wird die Variable „gameStart“ auf true gesetzt, damit können nun alle „BombermanGameClientHandler“ von den Clients empfangen. Der Spielserver startet das Spiel und begibt sich in eine „While“-Schleife, in der der „heartbeat“ mit der dazugehörigen „ClientID“ empfangen wird. Die Schleife läuft solange bis die Variable „gameOver“ auf true gesetzt wurde.

Wird die Schleife verlassen, wird die Funktion „closeBombermanGameServer()“ ausgeführt. Hier schließt der Spielserver seinen ServerSocket und beendet sich somit.

Weitere Methoden:

Eine weitere Methode ist die Funktion „readMsgQueue()“. Diese liest immer das erste Element der „msgQueue“, sofern diese nicht leer ist und gibt diesen Wert zurück.
Die Funktion „broadcastToClient()“ wird benutzt um die Initial-Matrizen zu senden. Diese sind die „Playermatrix“ und die „Gamematrix“. Mit Hilfe eines Iterators wird die „writer_list“ durchlaufen und alle Clients bekommen die Initial-Matrizen. Die Matrizen werden der Funktion als Jsonobjekte übergeben.
Die letzte zu nennende Funktion ist „sendToAllClients()“. Diese wird von anderen Klassen verwendet um sowohl Änderungen in der „Playermatrix“ als auch in der „Gamematrix“ an die einzelnen Clients zu versenden. Sie braucht ebenfalls ein Jsonobjekt, dass sie versenden kann.