TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik
Professur für Medieninformatik

Teamorientiertes Praktikum

Entwicklung eines Bomberman-Spiels für einen Tabletop

Tobias Gall Björn Naumann Sebastian Arndt
Michael Swora

Chemnitz, den 21. Januar 2016

Betreuer Dipl.-Inf. Michael Storz

Inhaltsverzeichnis

1 Einleitung **1**

2 Paper Prototype **3**

2.1 Erste Spielversuche 5

3 Implementierung **7**

3.1 Grundgerüst 7

3.1.1 Spielfeld 8

3.2 Menü 9

3.3 Steuerung 11

3.3.1 Erste Versuche der Entwicklung mit Processing 12

3.3.2 Eingliederung des Processing-Prototyps in Java 14

3.3.3 Erweiterung und Anpassung der Steuerung 14

3.3.4 Bewegung der Spielfigur 15

3.4 Bewegungsanimation 16

3.4.1 Bilder Entstehung 17

3.4.2 Umsetzung 18

3.4.3 Nachteile/Probleme 19

3.5 Upgrades 19

3.6 Entwicklung der Bomben & deren Animation 21

3.6.1 Bombenanimation 22

3.6.2 Algorithmus zum Wegsprengen der Blöcke/Bomberman . . 23

3.6.3 Nachteile/Probleme 24

3.7 Sterben und Siegen 24
3.7.1 Sterbeanimation und Auswirkungen des Sterbens 24
3.7.2 Siegbedingungen 25

4 User Tests **27**

4.1 Erster Test 27
4.1.1 User Feedback 27
4.2 Zweiter Test 28
4.2.1 User Feedback 29

1 Einleitung

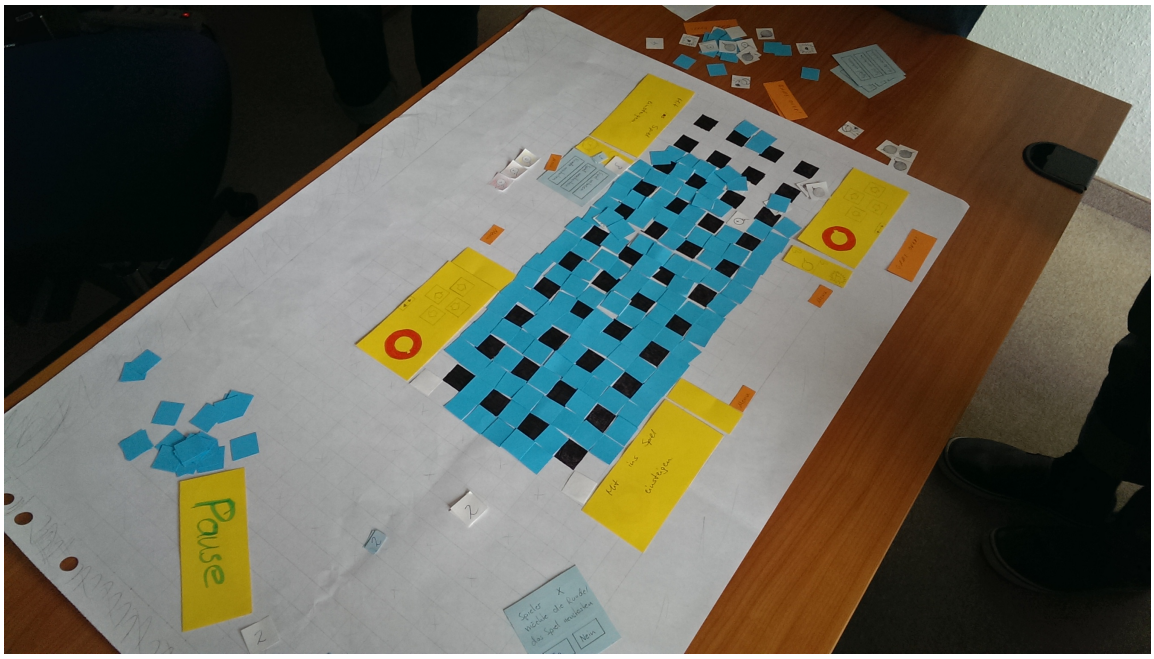
Im Rahmen unserer Studienordnungen ist ein teamorientiertes Praktikum zu erbringen. Bei diesem wurde uns die Aufgabe gestellt, eine kurzweilige Anwendung für einen Multitouch-Tisch zu entwickeln, die Menschen zu einer gemeinsamen Interaktion zusammenbringt. Zusätzlich sollte die Fähigkeit des Tisches, mehrfache Toucheingaben verarbeiten zu können, integriert werden. So begannen wir mit der Suche nach geeigneten Anwendungen und nach etwas Überlegung entschieden wir uns für die Eintwicklung eines Spiels, da Spiele für diese Aufgabe wie gemacht sind. Spielideen gab es im Team Einige. Doch am Ende entschieden wir uns für einen Klassiker - Bomberman. Bis zu vier Spieler sollten hierbei gegeneinander antreten und um den Sieg kämpfen.

2 Paper Prototype

Der erste Schritt in der Entwicklung des Bomberman-Spiels sollte, nach der Entscheidung für eine Spieleentwicklung und deren kurze Vorstellung bei unserem Betreuer, ein spielbarer Entwurf aus Papier, der sogenannte Paper Prototype, sein. Dieser soll als erster Schritt in der Entwicklung von Softwareprojekten aufzeigen, was möglich ist und die Vorstellungen aller Teammitglieder erstmals zusammenbringen. Danach kann die Machbarkeit des Projekts überprüft und ein Fahrplan für die Entwicklung und Umsetzung entworfen werden. Dabei geht es um Fragen wie: Was sind die elementaren Bestandteile des Projekts?, Was muss zuerst entwickelt werden und was ist optional? Und so fingen auch wir mit dem Entwurf des Paper Prototypes an.

Die grundlegende Frage dabei war: Welche Elemente soll das Spiel einmal enthalten? Eine nicht ganz triviale Frage, denn einige Teammitglieder hatten schon konkretere Vorstellungen des Endproduktes als Andere. Und während einer anschließenden Recherche wurden sogar noch andere Erweiterungsmöglichkeiten und Spielformen gefunden. Einige Elemente waren jedoch schon von Anfang an klar und gehören einfach zu einem Bomberman-Spiel hinzu. So beispielsweise die Jagd der Gegner durch das Legen von Bomben, dass Bomben eine gewisse Reichweite haben, man sich den Weg zu den Gegnern selbst erarbeiten muss und dass man die ersten beiden Parameter durch das Einsammeln entsprechender Items erhöhen kann. Außerdem sollte eine gewisse Anzahl an Leben verfügbar sein, um nicht sofort nach der ersten erfolgreichen Attacke eines Gegners aus dem Spiel auszuscheiden. Eine während der Recherche gefundene Erweiterung war

Abbildung 2.1: Paper Prototype



die Idee, die Geschwindigkeit der Spielfigur durch ein Item erhöhen zu können und auch die Anzahl der Leben flexibler zu gestalten. Da diese beiden Dinge gut im Team ankamen, wurden sie mit in die Implementierung aufgenommen und letztendlich in das Spiel integriert. Da sich das Team nun über einen groben Umfang an Ideen einig war, wurde in den folgenenden Teamtreffen mit der Arbeit am Paper Prototype begonnen. Es wurde gezeichnet, geschnitten, gemalt und die Ideen aller zu einem Ganzen verarbeitet. Während der Entwicklung des Prototypes fiel die Entscheidung erst mal auf ein einzelnes Spielfeldlayout, welches rechteckig sein sollte und dennoch genug Möglichkeiten für ein lustiges Spiel offen hält. Außerdem sollte jeder Spieler in seiner Ecke ein Bedienfeld erhalten, mit dem der Bomberman über den gesamten Tisch geführt werden kann und welches die wichtigsten Parameter seiner Spielfigur anzeigt. Das bedeutete zuerst nur die Anzahl der Bomben und deren Sprengkraft zu zeigen, wurde jedoch im fertigen Spiel noch erweitert. Damit die Spieler mit ihren Bomberman dabei nicht den Überblick verlieren, war eine Unterscheidung dieser nötig. Eine farbige Kennzeichnung der

Bomberman sollte dieses Problem lösen und auch die Steuerelemente sollten diese Farbe tragen, um ein Durcheinander der Spieler auszuschließen. Aufgrund der farblichen Unterscheidung ist so ein bequemes Spiel vom eigenen Platz aus möglich. Ein weiteres nötiges Element war ein Menü, über welches der Spieler aus dem Spiel aussteigen, einen Neustart iniziieren, oder das gesamte Spiel komplett abbrechen kann. Nach dem Ende der Arbeiten am Paper Prototype sahen wir erstmals das vorläufige Produkt. Die Steuerung war hierbei das einfachste Element, die eigentliche Bewegung der Bomberman über das Spielfeld das Schwierigste. Denn in Papierform mussten alle Felder einzeln aufgedeckt, jede Bombe einzeln gelegt und jeder Block eigenständig bedient werden. Für den ersten Nutzertest des Prototypen bedeutete das eine Menge Arbeit und viele Dinge, auf die es zu achten galt.

2.1 Erste Spielversuche

Der erste Test des Spiels in der Papierversion fand mit zwei Testergruppen aus nicht involvierten Spielern statt und verlief trotz des großen Aufwands recht erfolgreich. Gelobt wurden unter Anderem das leicht verständliche Grundprinzip des Spiels und die Idee im Allgemeinen. Alle Tester konnten sofort etwas mit dem Spiel anfangen und wussten recht schnell, was das Ziel des Spiels sein soll. Lediglich die manuelle Echtzeitumsetzung in Papierform wurde bemängelt, da einfach zu viele Objekte gleichzeitig von uns gesteuert werden mussten. Doch aufgrund der positiven Rückmeldungen konnte im Anschluss die Umsetzung des Bomberman-Spiels in die digitale Form beginnen.

3 Implementierung

3.1 Grundgerüst

Die Implementierung des Grundgerüstes für das Bomberman Spiel basiert auf dem Javaframework Processing mit der Erweiterung des Simple Multi-Touch(SMT) Frameworkes. Processing ist ein Framework zur Entwicklung von grafischen Anwendungen. Es kann mittels einer eigenständigen Entwicklungsumgebung oder direkt in Java verwendet werden. Das SMT Framework ermöglich zusätzlich die Verwendung von touchfähigen Monitoren.

Der Einstiegspunkt zur Implementierung einer grafischen Anwendung mit Processing ist die Klasse `PApplet`. Die eigentliche Implementierung erfolgt in der Klasse `Field`, welche von der Klasse `PApplet` erbt. Innerhalb dieser Klasse müssen zwei Methoden definiert werden. Zum einen die `setup`-Methode. Sie definiert alle Rahmenbedingungen der grafischen Anwendung, wie beispielsweise die Auflösung. Processing ruft diese Methode einmalig beim Starten der Anwendung auf. Die zweite Methode ist die `draw`-Methode. Diese Methode wird von Processing in jedem Frame aufgerufen. Sie definiert, welche grafischen Elemente auf dem Bildschirm angezeigt werden.

Neben der `setup`-Methode wurde in der Implementierung auch der Kontruktor der Klasse verwendet, um die in der `setup`-Methode verwendeten Attribute der Klasse zur Verfügung zu stellen. Der Konstruktur ließt dazu die `properties` -Datei mit definierten Einstellungen für das Spiel ein. Er erstellt zudem eine Hashmap mit allen in der Anwendung benötigten Grafiken, welche als Texturen für

einzelne Elementie dienen, ein. Über diese Hashmap können andere Funktionen konstanter Zeit auf die Grafiken zuzugreifen, ohne dass zusätzliche Dateisystemoperationen notwendig sind.

In der `setup`-Methode wird als Erstes die Auflösung der Anwendung und gleichzeitig die Rendering Engine definiert. Letztere wird auf `SMT.RENDERER` festgelegt, um das SMT-Framework benutzen zu können. Die Auflösung für die Anwendung wurde während des Entwicklungsprozesses fest definiert. Für den produktiven Einsatz lässt sich die Auflösung mittels der Varibalen `this.displayWidth` und `this.displayHeight` an die Auflösung des Bildschirms anpassen.

Die Methode initialisiert des Weiteren das Spielfeld, ein Raster von Blöcken, auf denen sich die Spielfiguren bewegen. Der Abschnitt "Spielfeld" beschreibt die Details dazu genauer. Neben dem Spielfeld werden auch die Bereiche für die Steuerung für die Nutzer angelegt, die so genannte `UserArea`. Diese Klasse, die von der SMT-Klasse `Zone` erbt, wird für jeden Spieler angelegt. Jeder `UserArea` wird dabei direkt eine Spielfigur, eine Instanz der `Bomberman`-Klasse zugeordnet. Der Nutzer kann über die Steuerungsmöglichkeiten der `UserArea` direkt mit seiner Spielfigur interagieren.

Um die Zugänglichkeit des Spieles für vier Spieler zu ermöglichen, wurden jeweils zwei Steuerungen auf der langen Seite des Bildschirmes positioniert. Zwei Steuerungen befinden sich gespiegelt auf der gegenüberliegenden Seite des Bildschirmes.

3.1.1 Spielfeld

Die Klasse `Blocks` stellt das grundlegende Spielfeld dar. Es generiert die zweidimensionale Blockaufteilung, wie sie im Paper Prototype bereits dargestellt war. Dazu verwaltet die Klasse eine Matrix aus einzelnen Elementen der Klasse `Block`. Diese Klasse ist für die Darstellung eines Blockes verantwortlich. Dazu gehört das Erstellen einer Shape, mit dem Processing Element `PShape`, mit einer bestimmten Textur und die Anzeige des Blockes. Dabei ist die Erstellung, das Rendering,

von der Anzeige getrennt. Ein Rendering in jedem Frame würde sich negativ auf die Performance des gesamten Spieles auswirken. Diese Trennung von Rendering und Anzeige wird von allen Elementen benutzt. Die größe eines Blockes berechnet sich dynamisch anhand der Displaygröße und der Dimensionen des Spielfeldes. Dadurch skaliert das Blockraster mit der Auflösung des Bildschirmes. Die Textur des Blockes wird anhand eines Types bestimmt. Folgende Typen existieren: verdeckter Block, leerer Block, Block mit Gegenstand, statischer Block, Block mit Bombe.

Während des Spielstartes werden unter den verdeckten Blöcken zufällig Gegenstände verteilt. Mehr dazu im Abschnitt “Upgrades”. Statische Blöcke werden nur in jeder zweiten Zeile und Spalte des Spielfeldes generiert, sodass zwischen zwei statischen Blöcken mindestens ein verdeckter oder leerer Block existiert.

Wie im Paper Prototype bewegt sich der Bomberman innerhalb des Blockraster. Dabei ist es ihm nur möglich sich auf leeren Blöcken zu bewegen. Verdeckte und statische Blöcke verhindern die Bewegung der Spielfigur. Um der Spielfigur beim Spielstart einen Raum zur Bewegung zu geben, wird um seine Startposition, jeweils in einer Ecke des Blockrasters, ein horizontaler und vertikaler Block ausgespart und als leerer Block dargestellt. Von dieses Blöcken aus muss sich der Spieler seinen Weg durch das Spielfeld sprengen. Wobei er nur verdeckte Blöcke sprengen kann. Wird ein verdeckter Block gesprengt, wird entweder ein unter ihm versteckter Gegenstand oder ein leerer Block gerendert und angezeigt.

3.2 Menü

Das Menü kann über eine kleine Touchzone neben der User Area erreicht werden. Jedem Spieler ist ein eigenes Menü zugeordnet. Über das Menü kann per Touchevent ein Submenü geöffnet und geschlossen werden. Zum Submenü gehören 3 Buttons(Beenden, Neues Spie, Aufgeben). Mit einem Touchevent auf einen dieser Buttons wird eine Checkbox zugehörig zum vorher gedrückten Button geöffnet

und kann auch wieder geschlossen werden. Diese Checkboxen liefern dem Spieler eine Beschreibung der Aktion und enthalten zusätzlich einen Button “Ja” und einen Button “Nein”. Diese beiden Buttons sollen verhindern das der Spieler eine Aktion ohne weitere Bestätigung ausführt. Der “Ja” Button löst die Aktion aus, der “Nein” Button schließt die Checkbox und beide Bestätigungbuttons wieder.

Deklariert wird das Menü innerhalb der Userarea in der Funktion `touch()`. Bei der initialisierung des Menü´s wird vorher geprüft auf welcher Seite sich der Button befindet, denn die Spielerfarben rot und violett erfordern eine Spiegelung. Die Übergabe Werte für das Menü umfasst die x und y Koordinate für den Start, die Größe des Buttons, die Eigenschaft ob sich das Menü auf der Seite von rot/violett befindet (1) oder bei orange, blau (2) und die Bomberman Spielfigur.

In der Klasse `Menu` werden die 3 Buttons (`button_close`, `button_new`, `button_surr`) des Submenü´s durch die Klasse `Submenu` deklariert. An die Submenübuttons werden die Koordinaten des Startpunktes, die Größe, der Text des Buttons. die Seite (rot/violett oder orange/blau) und die Spielfigur übergeben. Um das Menü öffnen und schließen zu können wird die private boolean Variable `pop` angelegt und mit false deklariert (das Menü ist anfangs geschlossen), welche beim Touch auf `!pop` gesetzt wird. Danach wird je nach Entscheidung der if else Schleife das Submenu angezeigt (`pop=true`) oder geschlossen (pop=false). Das schließen im else Zweig übernimmt die SMT Funktion `clearChildren()`.

Innerhalb der Klasse `Submenu` gibt es ebenfalls wieder eine private boolean Variable `pop`, die für das Schließen und Öffnen der Checkbox verantwortlich ist und auf die gleiche Art und Weise funktioniert wie in der Menüklasse. Eine if else Schleife stellt fest, auf welcher Seite die Submenübuttons erstellt werden müssen, wodurch der x-Achsen Startpunkt geändert wird. Der Text des Buttons wird in der Menüklasse mit übergeben und hier durch `button_name` dargestellt. Da jeder Submenübutton eine andere Checkbox öffnet, wird innerhalb des Touchevents nach Typ des Buttons (Neues Spiel, Beenden, Aufgeben) unterschieden.

Innerhalb der Klasse `Checkbox` werden die Buttons für “Ja” und “Nein” de-

klariert und nebeneinander angeordnet. Dem `button_yes` wird noch zusätzlich eine ID übergeben, von welchem Submenübutton er aufgerufen wurde, um auch einzelne Aktionen für die Buttons bereitzustellen.

In der Klasse `ButtonNo` wird innerhalb des Touchevents der Aufruf für das Schließen der Checkbox bereitgestellt. Dies geschieht mit den SMT Funktionen `this.getParent().getParent().clearChildren()`. Innerhalb der Klasse `ButtonYes` werden per switch die Funktionen der Submenübuttons bereitgestellt. Für das Aufgeben, wird per if geprüft, ob der Bomberman noch lebt und wenn ja wird die Funktion `bomberman.lostgame()` aufgerufen.

3.3 Steuerung

Am Anfang des Spielentwurfs stellte sich die Frage, wie eine Steuerung der Spielfiguren wohl am Besten zu realisieren wäre, damit das Spiel recht einfach und trotzdem ohne ständiges Hinsehen so gut wie möglich spielbar ist. Zur Auswahl standen folgende Möglichkeiten: eine direkte Steuerung durch Berühren der Figuren, eine Steuerung durch Pfeiltasten, oder die Steuerung mittels eines simulierten Trackballs. Der Punkt, welcher eine Entscheidung in diesem wichtigen Teil des Spiels mit entschied, ist die Tatsache, dass man an einem Tabletop kein haptisches Feedback auf seine Eingaben erhalten kann. Deshalb musste die Steuerung so einfach wie möglich gehalten werden, auch ohne ständiges Hinschauen spielbar sein und ein Chaos unter den Mitspielern verhindern. Also wurden Argumente für und gegen die verschiedenen Varianten gesammelt. Eine direkte Steuerung der Spielfiguren durch Berühren dieser fiel recht schnell aus, denn das würde zu dem genannten Chaos unter den Spielern führen. Man müsste die ganze Zeit in Touchreichweite zur eigenen Figur sein, was eine ständige Bewegung mit Selbiger um den Tisch bedeuten würde. Diese Möglichkeit war somit untauglich. Die nächste Option der Pfeiltasten klang schon besser, doch auch hier gab es Bedenken. Denn Pfeiltasten kann man auf einem Touchscreen zwar darstellen, jedoch ist auch hier

die Umsetzung einer Reaktion bloß ohne ein haptisches Feedback möglich. Es wäre zwar sichtbar, welche Taste gerade berührt wird, doch fühlen lässt es sich nicht, wodurch man wohl erneut zum öfteren Hinsehen gezwungen wäre. Außerdem sind bei virtuellen Pfeiltasten mehrere Finger zum Spielen nötig, die Finger müssen recht genau platzieren werden und jeder Spieler hat dabei wohl eine andere Geschicklichkeit. Also fiel die Entscheidung zugunsten eines einzelnen beweglichen Elements, was man auch in anderen Spielen für Smartphones und Tablet findet. Eine Art Ball, den man in einem gewissen Rahmen über den Bildschirm zieht, einen virtuellen Trackball. Diese Art der Steuerung sollte nicht sonderlich schwierig sein, benötigt bloß einen Finger zur Eingabe und ermöglicht trotzdem eine Bewegung in alle Richtungen. Denn hat man den Trackball einmal mit dem Finger berührt, so folgt dieser in die Bewegungsrichtung des Fingers und ist somit quasi blind bedienbar.

3.3.1 Erste Versuche der Entwicklung mit Processing

Auf der Suche nach einem passenden Tool für die Entwicklung des Spiels wurde, wie oben schon erwähnt, zuerst eine Programmiersprache in Betracht gezogen, die auf Grafiken und Animationen ausgerichtet ist. Processing ist ein quelloffenes Projekt und wurde am Massachusetts Institute of Technology entwickelt. Da in dieser Sprache Beschreibungen von Formen und Objekten möglich sind, erschien sie anfänglich perfekt für das Spiel. Jedoch stellte sich später heraus, dass Java vielleicht doch die geeignetere Programmiersprache ist. Da wir die Entwicklung des Projekts jedoch in Processing begonnen hatten, fiel auch die erste Entwicklung einer Steuerung auf dieses Tool. Am Anfang galt es zu überlegen, wie eine einfache Steuerung denn wohl aussehen müsste, damit sie als solche auch wahrgenommen würde und trotzdem intuitiv bedienbar ist. Was wäre da einfacher als ein Kreis, der geviertelt ist? Jedes Viertel steht für eine Bewegungsrichtung und überträgt diese Richtung auf den Bomberman. Und genau das wurde auch der erste Entwurf. Um einen Kreis in Processing zeichnen zu können, muss man zu-

nächst in einer Setup-Funktion eine Leinwandgröße festlegen und anschließend können in der Draw-Funktion Befehle eingefügt werden, die in jeder Frame wiederholt zeichnet. Für einen Trackball wird dabei eine Linienfarbe und eine Füllfarbe festgelegt und im Anschluss die eigentliche Form gezeichnet. Im konkreten Projekt bedeutete dies einen großen Kreis ohne Füllung zu zeichnen (nachfolgend: Außenkreis), zwei Diagonalen hindurch zu ziehen, damit Viertel entstehen, einen kleinen Kreis in der Mitte als Ruheposition zu platzieren und über all das einen beweglichen, farbigen Kreis (nachfolgend: Trackball) zu zeichnen. Dieser sollte nun dem Mauszeiger, später dann dem eigentlichen Finger in seiner Bewegung folgen und dadurch die Bewegungsrichtung anzeigen. Bei der Umsetzung dessen war dann ein wenig Mathematik gefragt, denn zeichnet man einen Trackball wie gerade beschrieben, so folgt dieser der Maus bzw. dem Finger über die gesamte Leinwand. Das ist jedoch nicht gewünscht, denn der Trackball soll sich nur innerhalb des Außenkreises, der als Grenze dienen soll, bewegen. Was man deshalb tun muss, ist die Position der Maus bzw. des Fingers zu prüfen und bei einem Überschreiten der Grenze des Radius des Außenkreises - dem Radius des Trackballs, dessen weitere Bewegung zu verhindern. Man führt quasi eine unsichtbare Grenzlinie ein, die sich aus dem Radius des Außenkreises subtrahiert mit dem Radius des Trackballs beschreiben lässt. Und sobald die Maus diesen unsichtbaren Kreis verlässt, bewegt sich der Trackball nur noch auf dieser unsichtbaren Grenzlinie. Processing bietet für diese Aufgabe einen einfachen Weg an, denn man kann sich die Position der Maus mit den Funktionen mouseX und mouseY zurückgeben lassen, mittels PVector einen Vector aus diesen beiden Angaben erzeugen und diesen Vektor am Mittelpunkt des Außenkreises ansetzen. Nun lässt sich mittels dist-Befehl die Entfernung zwischen dem Außenkreismittelpunkt und der Mausposition berechnen und sollte diese größer als die unsichtbare Grenze sein, so wird der aufgestellte Vektor einfach normalisiert und mit der maximal möglichen Position des Trackballs multipliziert. Schon bleibt der Trackball nur noch innerhalb des Außenkreises und verlässt diesen nicht mehr, selbst wenn die Maus bzw. der

Finger es tut. Zur Kontrolle, ob der Trackball auch die richtige Richtung der Bewegung darstellt, lässt sich noch eine einfach Ausgabe der Richtungen in Textform einbauen, die im jeweiligen Viertel des Außenkreises aktiviert wird. Und neben dem Trackball gehörte auch noch ein Knopf zum Bombenlegen (nachfolgend: Bombknopf) zum ersten Entwurf der Steuerung.

3.3.2 Eingliederung des Processing-Prototyps in Java

Nachdem die Entwicklung des Prototyps der Steuerung beendet war, war inzwischen auch die Programmiersprache unseres Projekts eine Andere. Es wurde nun in Java weiterentwickelt, was eine Migration der Steuerung nötig machte. Glücklicherweise ist Processing auf Java aufgebaut und ermöglicht einen Export der geschriebenen Projekte in Java. Dies ersparte uns etwas Arbeit und nach dem Exportieren wurde der erzeugte Code mit minimalen Anpassungen in eine neue Java-Klasse eingefügt, wo er nun problemlos lief.

3.3.3 Erweiterung und Anpassung der Steuerung

Nach dem erfolgreichen Einbau der Steuerung in Java, mussten die bis jetzt einzeln entwickelten Teile zusammengefügt werden. Das heißt, dass nun echte Funktionen der Bomberman angesprochen werden mussten, sobald sich der Trackball bewegt und auch die Bomben müssen von den Spielfiguren gelegt werden können. Zum Erreichen dessen, musste nur die bisherige Ausgabe der Bewegungsrichtung mit je einem Funktionsaufruf ersetzt werden und auch das Touchevent des Bombknopfs wurde mit der entsprechenden Funktion, eine Bombe zu legen, versehen. Schon konnte erstmals richtig mit der Steuerung getestet werden, was auch in diesem Fall sehr gut funktionierte. Als folgender Arbeitsschritt galt es noch, die ganz am Anfang genannten Wünsche einer Anzeige zu integrieren, was direkt in Java geschah. Dazu wurde die Position des Steuerkreises inklusive Trackball ein wenig weiter nach rechts verschoben, der Bombknopf ein wenig links und

somit entstand dazwischen ein Raum für die Anzeige. Doch welche Items sollen überhaupt gezeigt werden? Wie anfänglich schon im Paper Prototype festgestellt, ist eine Anzeige der Bombenanzahl, sowie deren Reichweite sinnvoll. Weitere Möglichkeiten waren die Anzahl der Leben, oder die Geschwindigkeit, mit der der Bomberman über das Spielfeld läuft. Da man Letztgenanntes nicht unbedingt wissen muss, da man es während des Spiels selbst erfährt, wurde die Anzahl der Leben noch mit aufgenommen. Diese Information ist deshalb für den Spieler interessant, da er so weiß, wie vorsichtig er sein sollte und wie es um die anderen Spieler bestellt ist. Außerdem kann man durch das Einsammeln eines Items seine Anzahl an Leben wieder erhöhen und dadurch die eigene Siegchance wahren. Zur Anzeige der verschiedenen Parameter wurden die entsprechenden Bilder der Items in Mittellage des Steuerpanels platziert und in deren Vordergrund ein Zahlenwert geschrieben. Der Spieler kann sich dadurch einen schnellen Überblick über den Zustand seines Bomberman verschaffen und begegnet bekannten Symbolen, weshalb ein Verständnis leicht fallen sollte. Zur weiteren Unterstützung des Spielers wurde noch der Außenkreis um den Trackball in der jeweiligen Spielerfarbe eingefärbt. Somit weiß jeder Spieler, welcher Bomberman zu ihm gehört, falls dies einmal vergessen werden sollte.

3.3.4 Bewegung der Spielfigur

Die Spielfiguren werden vom Spieler über die UserArea gesteuert. Der Spieler hat die Möglichkeit, seine Figur innerhalb des Blockrasters auf den begehbaren Blöcken zu bewegen. Dabei kann die Bewegung in vier Richtungen erfolgen: oben, unten, links und rechts. Die Eingabe über die UserArea ruft die Bewegungsmethoden in der Klasse Bomberman auf. In dieser Klasse existiert eine eindeutige Zuordnung zu einer Koordinate im Blockraster und somit eine eindeutige Zuordnung zum Spielfeld. Mit jeder Bewegung der Spielfigur wird diese Rasterzuordnung neu berechnet. Der Spieler bewegt seine Spielfigur pixelweise horizontal und vertikal durch die Bewegungsmethoden. Die Figur bewegt sich dabei auf vir-

tuellen Linien, die die Mittelpunkte der Blöcke miteinander verbinden. Mit jedem Frame wird überprüft, ob die Spielfigur sich zu einem weiteren Block bewegen kann. Dabei wird geprüft, ob sich der Rand der Spielfigur in einen Block bewegen würde, der betreten werden darf. Darf dieser Block nicht betreten werden, so kann sich die Spielfigur in Richtung dieses Blocks nicht weiter bewegen. Die Abfrage beschränkt sich zudem auf die der Spielfigur direkt angrenzenden Blöcke oder den Spielfeldrand. Durch die Bewegung auf der virtuelen Linie müsste sich die Spielfigur direkt im Mittelpunkt eines Blockes befinden, um von einer vertikalen in eine horizontale Bewegung, oder umgekehrt, überzugehen. Mit Hilfe eines Toleranzbereiches um den Mittelpunkt des Blockes herum, kann die Spielfigur zwischen einer vertikalen und horizontalen Linie übergehen. Ein zu kleiner Tolleranzbereich hemmt dabei aber den Spielfluss, da es für den Spieler zu schwierig ist, seine Figur in diesen Bereich zu steuern.

3.4 Bewegungsanimation

Die Animation sollte insgesamt eher minimalistisch werden, jedoch trotzdem ansprechend wirken. Damit überhaupt eine Animation erkennbar ist, braucht man mindestens zwei verschiedene Bilder, die abwechselnd angezeigt werden. Es wurden also für jede Bewegungsrichtung zwei verschiedene Bilder (mehr dazu im nächsten Absatz „Bilder Entstehung“) entworfen, die in einem bestimmten Takt abwechselnd angezeigt werden sollten, und zusätzlich ein Bild für den Ruhezustand, wenn die Figur gerade nicht bewegt wird. Da auch geplant war, ein Upgrade einzubinden, was die Figur schneller laufen lässt, musste sowohl die Animation als auch die interne Positionsbestimmung von der aktuellen Geschwindigkeit der Figur abhängig gemacht werden.

3.4.1 Bilder Entstehung

Als Grundlage für den Entwurf der Animationsbilder wurde der Platzhalter genommen: ein einzelner Bomberman in der Frontalansicht, ohne erkennbare Beine. Eine Bewegungsanimation ist jedoch am ehesten an einer Bewegung der Beine zu erkennen. Der erste Schritt war also, den Bomberman so zu verändern, dass eine Bewegung der Beine erkennbar war. Dazu wurde einfach der „Rock“, den unser Platzhalter Bomberman an hat, etwas angepasst, und das Resultat war mit etwas Toleranz durchaus als Beine erkennbar. Dieses Bild (Blickrichtung unten, beide Beine am Boden) wurde dann das Bild für den Ruhezustand. Da der Bomberman quasi „nach unten“ schaut, wurden zuerst die Animationsbilder für die `movedown()` Funktion entworfen. Hierzu wurde wieder unser Ruhezustand Bild genommen, und daraus zwei neue Bilder entwickelt: auf einem Bild ist der linke Fuß angehoben, und auf dem anderen Bild der rechte Fuß. Wenn die Bilder abwechselnd angezeigt werden, war bereits einen simpler Bewegungsablauf erkennbar. Als nächstes wurden die Bilder für die `moveup()` Funktion erstellt, also die Bewegung nach oben. Hierzu wurde das Ruhezustand Bild einfach horizontal gespiegelt, und die Augen und der Mund des Bomberman entfernt. Anschließend sind wieder zwei Versionen entstanden, mit jeweils einem Bein angehoben.Für die seitlichen Bewegungen (also `moveleft()` und `moveright()`) mussten einige Anpassungen vorgenommen werden: Der Docht, der aus dem Kopf des Bomberman ragt, musste entsprechend positioniert werden, ebenso wurde jeweils nur ein Auge verwendet, welches in Laufrichtung blickt. Der ursprüngliche Platzhalter, der als Vorlage benutzt wurde, hatte jedoch nur einen linken Arm, mit dem er eine Bombe hält. Wenn sich der Bomberman nach rechts bewegt, konnte man also keine Arme sehen, da der linke Arm vom Kopf verdeckt wurde. Bei den Bildern für `moveright()` wurde dann kurzerhand ein kleiner verstümmelter Arm eingefügt, als kleiner makabrer Hinweis, dass Bomben kein Spielzeug sind. Auflösungsbedingt war der verstümmelte Arm aber kaum erkennbar auf dem Spielfeld. Die Beine der Bomberman wurden für die seitwärts Bewegungen kurzerhand aus

zwei Dreiecken entworfen, die einmal geschlossen, und einmal geöffnet waren. So war auch bei der Seitwärtsbewegung eine Bewegungsanimation erkennbar. Da jetzt alle 9 nötigen Bilder vorhanden waren, wurden die Bilder auch noch für die Farben der anderen 3 Bomberman erstellt. Hier tauchte aber recht schnell ein Problem auf: wenn in Photoshop die Bereiche einfach mittels Farbeimer eingefärbt wurden, wurden die Übergänge zum transparenten Hintergrund nicht passend umgefärbt, und es konnten bei den Bildern Treppenkanten erkannt werden. Die Bomberman wirkten also grob pixelig. Dieses Problem wurde jedoch recht schnell mittels der Farbwertkorrektur gelöst, allerdings gab es jetzt leichte Abweichungen zu den ursprünglich gewählten Farben. Die neuen Farben wurden beibehalten.

3.4.2 Umsetzung

Die benötigten Bilder waren alle erstellt, und die Grundidee zum Ablauf stand auch. Es konnte also damit begonnen, die Animation im Quellcode einzubinden. Grundsätzlich sollte die Berechnung der Position getrennt von der Animation stattfinden. Daher wurden die move Funktionen nur geringfügig angepasst. Eines der Attribute der Bomberman beinhaltet die aktuelle Geschwindigkeit (Attribut „speed“) des jeweiligen Bomberman, und entspricht der Anzahl Pixel, die sich der Bomberman bei einem move Befehl pro Frame in die gewünschte Richtung bewegt. Die Veränderung der Position in den move Funktionen wurde also statt von einem festen Wert einfach von dem Geschwindigkeits Attribut abhängig gemacht. Die Bomberman haben ihre eigene draw Funktion, in der bestimmt wird, was auf dem Bildschirm ausgegeben wird. Vom Touchpanel kommt immer eines von 5 verschiedenen Signalen: direction=0 für stehend, direction=10 für Bewegung nach oben, direction=20 für Bewegung nach unten, direction=30 für Bewegung nach links und direction=40 für Bewegung nach rechts. In der draw-Funktion unserer Bomberman wurden also abhängig von der direction unterschiedliche Bilder ausgegeben. Bevor die Bilder gezeichnet wurden, wurden sie einmalig gerendert und in einer Hashmap gespeichert. Wenn die direction 0 war, der Bomberman

also einfach nur da stand, wurde einfach nur das Bild für den stehenden Bomberman gezeichnet.Wenn allerdings eine Bewegung dargestellt wird, mussten in einem bestimmten Takt zwei verschiedene Bilder auf dem Bildschirm gezeichnet werden. Um dies zu realisieren, wurde in der Bomberman Klasse ein weiteres Attribut namens „count“ angelegt. Count ist ein float Wert, und soll Werte zwischen 0 und 50 annehmen. Wenn count kleiner als 25 ist, wird Bild 1 der entsprechenden Richtung gezeichnet, wenn count größer gleich 25 ist, wird Bild 2 gezeichnet. In beiden Fällen wird zudem der entsprechende move-Befehl ausgeführt. Wenn count größer gleich 50 war, wurde count wieder auf 0 resettet. In der ersten Version zählte count einfach bei jedem Aufruf um 1 nach oben. Damit sich bei höherem speed (nach einsammeln der entsprechenden Upgrades) die Bewegungsgeschwindigkeit auch optisch erhöht, wurde count ab der zweiten Version immer um den aktuellen speed Wert erhöht. Auf diese Weise verkürzt sich das Intervall, in dem sich die Bilder abwechseln.

3.4.3 Nachteile/Probleme

Da die Animation komplett abhängig von den Frames ist, (also der fps (frames per second)) statt von der vergangenen Zeit, wirkt die Bewegung bei niedriger fps (bedingt durch z.B. zu langsamer Rechner für die gewählte Auflösung) deutlich langsamer, als bei höherer fps. Dieses Problem könnte behoben werden, indem eine Variante entwickelt wird, die nur von der Zeit abhängig ist.

3.5 Upgrades

Wie der Abschnitt “Spielfeld” schon angeschnitten hat, existieren im Spiel Gegenstände, die von der Spielfigur eingesammelt werden können. Die sogenannten Upgrades. Upgrades sind Blöcke mit einer bestimmten Textur. Je nach Upgrade werden verschiedene Texturen gerendert und angezeigt. Beim Spielstart sind die Upgrades allerdings nicht zu sehen. Sie werden zufällig unter den verdeckten

Blöcken versteckt. Dazu wird in der Instanz der Klasse `Block` das Attribut `type`, welcher den Gegenstand identifiert, und das Attribut `covered` gesetzt. Das Attribut `covered` bewirkt, dass der Block mit der Textur des verdeckten Blocks angezeigt wird. Es gibt verschiedene Gegenstände: ein zusätzliches Leben, eine zusätzliche Bombe, erhöhung der Sprengreichweite der Bombe und Erhöhung der Geschwindigkeit der Spielfigur.

Die Gegenstände haben verschiedene Auswirkungen auf die Spielfigur. Das einsammeln eine zusätzlichen Lebens bewirkt, dass die Spielfigur einmal mehr sterben kann, bevor sie aus dem Spiel ausscheidet. Die zusätzliche Bombe bewirkt, dass die Spielfigur eine Bombe mehr zur Verfügung hat, die sie legen kann. Die Sprengreichweite der Bombe kann ebenfalls durch einen Gegenstand erhöht werden. Je nach Anzahl der eingesammleten Gegenstände zur Erhöhung der Reichweite, erhöht sich der Umkreis der betroffenen Felder bei der Detonation einer Bombe um eins. Der letzte Gegenstand der eingesammelt werden kann, dient zur Erhöung der Geschwindigkeit der Spielfigur. Diese kann sich so schneller durch das Spielfeld bewegen.

Um die Gegenstände zu erreichen, muss der Spieler die verdeckten Blöcke mit einer Bombe wegsprengen. Blöcke, die im Umkreis der Sprengreichweite des Bombe liegen werden beim zünden der Bombe neu gerendert. Wenn ein Block mit einem Gegenstand versehen ist, wird dieser angezeigt. Ansonsten wird ein leerer Block angezeigt. Die Spielfigur ist dann in der lage den Gegenstand durch einfaches herüberlaufen über den Gegenstand auszusammeln. Wurde der Gegenstand aufgesammelt, wird auch dieser Block als leerer Block dargestellt.

Damit ein Spieler erkennt, dass seine Spielfigur einen Gegenstand eingesammelt hat, wurde eine Animation implementiert. Die Animation läst den Gegenstand über das Spielfeld zur `UserArea` des Nutzers fliegen und verändert an dieser Stelle die Anzeige über vorhandene Gegenstände. Sie ist so umgesetzt, dass der Gegenstand sich anhand eines berechneten Vektors, von der aktuellen Blockposition zur `UserArea`, bewegt. Dabei wird er mit einer definierten Anzahl Pixel

in die Richtung des Vektors verschoben. Die Animation bricht ab und entfernt das Bild des Gegenstandes, wenn die Position in der `UserArea` erreicht ist.

3.6 Entwicklung der Bomben & deren Animation

Da die Programmierung der Bomben voraussichtlich recht umfangreich werden würde, wurde entschieden, eine extra Klasse zu erstellen, statt es mit in die Bomberman Klasse einzubinden. Ein Objekt der Klasse sollte immer genau eine Bombe darstellen, und alle damit verbundenen Funktionalitäten (runterticken des Countdowns, danach alles im Explosionsradius zerstören, bei weiteren Bomben diese sofort auslösen etc.) erfüllen. Die nächste Frage war, wie es jetzt möglich war, dynamisch Objekte dieser Klasse erstellen zu können. Damit dieser Aspekt möglichst simpel umgesetzt werden konnte, wurde zu Spielbeginn eine festgelegte Anzahl Bombenobjekte erzeugt, und diese dann immer wieder zu benutzt.Damit es auf dem Spielfeld nicht zu unübersichtlich wird, wurde festgelegt, dass ein Spieler maximal 5 Bomben gleichzeitig ablegen kann. Um dies zu realisieren, wurde ein Feld in der `field`- Klasse angelegt, welches 20 Objekte der Bombenklasse verwalten kann. Jeder Bomberman hat 5 festgelegte Feldelemente, die seine Bomben abspeicherten (z.B. Spieler rot die Feldelemente 0-4). Jede Bombe hat Attribute für ihre Position, ob sie aktuell genutzt wird (also ob sie auf dem Spielfeld liegt), ihre Reichweite bei der Explosion, und die aktuelle Zeit, bis wann sie explodiert. Wenn vom Touchpanel der Befehl zum Legen einer Bombe kam, wurde zuerst überprüft, ob der Spieler noch eine Bombe verfügbar hat. Danach wurde in dem Bombenfeld im entsprechenden Intervall (bei Spieler rot z.B. 0-4) nach einer ungenutzten Bombe gesucht, und diese bekam die aktuelle Position des Spielers zugewiesen, wurde als genutzt markiert, und der Countdown wurde initialisiert.

3.6.1 Bombenanimation

Die Animation der Bomben muss wieder minimalistisch werden, jedoch sollte es möglich sein, auf den ersten Blick grob abschätzen zu können, wie lang die Bombe ungefähr noch benötigt, bis sie explodiert. Die Bombe benötigt ungefähr 3 Sekunden (also 180 Frames bei 60 fps), bis sie selbstständig explodiert. Die entsprechende Zahl ist auf der Bombe erkennbar (die 3 in grün, die 2 in gelb, die 1 in rot), und zeigt an, wie lange es noch dauert. Zudem ist eine kleine Animation eingebunden, um zu erkennen, dass die Bombe aktiv ist. Hierzu wurden 7 verschiedene Bilder entworfen: Zwei Bilder mit einer kleinen Bombe, die mit einer grünen 3 in der Mitte versehen sind, und sich nur durch ihre Farbe am Dochtende unterscheiden (bei einem Bild war die Flamme am Dochtende innen rot und außen gelb, bei dem anderen Bild war es umgedreht). Diese beiden Bilder wechseln sich in der ersten Sekunde in einem angebrachten Takt ab. Für die zweite Sekunde wurden 3 Bilder entwickelt. Die ersten 2 Bilder waren wie die Bilder für die erste Sekunde, statt der grünen 3 war diesmal allerdings eine gelbe 2 auf den Bomben zu sehen. Für das dritte Bild wurde die Bombe dann deutlich vergrößert. Die Idee dahinter war, dass sich in den ersten 1 1/2 Sekunden nur die Flamme des Dochtes und die Zahl ändert, und in den zweiten 1 1/2 Sekunden des Countdowns sich zusätzlich noch die Größe der Bombe verändert, um zu verdeutlichen, dass sie gleich explodiert. Für die dritte Sekunde entstanden 2 Bilder: eine kleine Bombe mit einer roten 1, und eine große Bombe mit ebenfalls einer roten 1. Diese beiden Bilder werden dann in einem schnellen Intervall abwechselnd angezeigt. Im Quellcode wurde die Animation dann folgendermaßen eingebunden: Die Bilder wurden wieder einmalig gerendert und in einer Hashmap gespeichert. In jedem Frame wurde der Countdown um 1 verringert, und alle 10 Frames wurde ein anderes Bild angezeigt, nach oben beschriebenem Muster.Zudem enstand für die Bomben noch ein weiteres Bild. Eine Flamme, die die Zerstörung bzw. deren Ausbreitung darstellen soll.

3.6.2 Algorithmus zum Wegsprengen der Blöcke/Bomberman

Nach Ablauf des Countdowns explodiert die Bombe und alles in ihrem Radius wird zerstört. Eine der ersten Designentscheidungen, die bezüglich der Bomben getroffen wurde, war, dass nur der erste Block in den jeweiligen Richtungen weggesprengt werden soll, und nicht mehrere Blöcke hintereinander. Die Explosion entfaltet sich bei „freier Bahn“ (kein BLOCK/ STATIC/coveredItem/Feldende in den 4 Richtungen in der Reichweite) komplett bis zu dem Radius, ansonsten stoppt sie entsprechend früher. Die Flammen der Explosion bleiben ungefähr 1 Sekunde bestehen, also 60 Frames. Es wurde also einen Algorithmus entworfen, der mittels einer Schleife in jede der 4 Richtungen (+ natürlich den Koordinaten, an dem die Bombe vorher lag) folgendes abprüft: Bis die Explosion auf ein Hindernis stößt (STATIC/BLOCK/coveredItem/Feldende) wird in jedem Feldelement eine Flamme angezeigt, und es wird abgeprüft, ob sich ein Bomberman oder eine Bombe auf eben diesem Feld befindet. Wenn sich ein Bomberman auf diesem Feld befindet, wird dessen `startDie` Methode aufgerufen, und für eine Bombe wird auf diese Koordinaten die `bombexplode` Methode ausgeführt. Auf die `startDie` Methode wird im Kapitel Sterbeanimation genauer eingegangen. Da die Bomben nicht direkt mit dem Feld verknüpft sind (es wird lediglich vermerkt, dass eine Bombe an der entsprechenden Stelle liegt, aber nicht welche), wird in der `bombexplode` Methode das Bombenfeld nach den passenden Koordinaten durchsucht, und deren Countdown auf 1 reduziert. Dadurch explodiert diese Bombe im nächsten Frame ebenfalls. Wenn sich in der Reichweite ein BLOCK oder coveredItem befindet, wird an dieser Stelle ebenfalls eine Flamme erzeugt. Es wird allerdings erst im letzten Frame, in dem die Flamme angezeigt wird, auch der BLOCK auf empty gesetzt, bzw. das covered auf false, damit ein leeres Feld oder das entsprechende Upgrade erscheint. Wenn bereits zu Beginn die Blöcke entfernt werden, würden im nächsten Schleifendurchlauf auch noch dahinter liegende Blöcke entfernt. Da aber nur der erste erreichte Block in jede Richtung weggesprengt werden darf, musste dies unterbunden werden.

3.6.3 Nachteile/Probleme

Wie bereits bei der Bewegungsanimation ist auch die Bombenanimation komplett von den frames per second abhängig, statt von der Zeit. Um ein flüssigeres Spielen bei niedriger fps zu ermöglichen, muss die Animation komplett umgeschrieben werden, und nur von der vergangenen Zeit abhängig gemacht werden.

3.7 Sterben und Siegen

3.7.1 Sterbeanimation und Auswirkungen des Sterbens

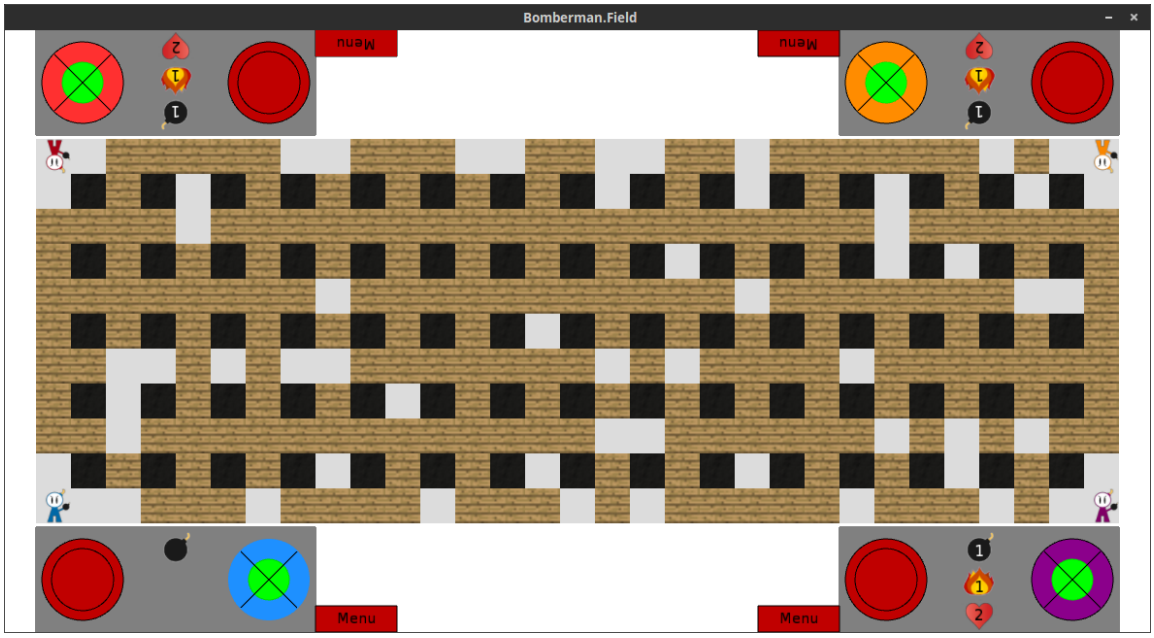
Was muss alles passieren, wenn ein Spieler in einer Bombenexplosion ist? Einfach die Position des Bomberman wieder auf Ausgangsposition zu setzen, reicht nicht aus. Es muss erkennbar sein, dass die Spielfigur gerade gestorben ist. Dazu wurde eine Verzögerung von 300 Frames eingebaut, bevor die Position zurückgesetzt wird. Während diesen 300 Frames wird die Steuerung deaktiviert, und eine Animation abgespielt. Diese Sterbeanimation muss wieder so simpel wie möglich sein. Der erste Entwurf enthielt 2 Bilder, die wieder abwechselnd angezeigt werden: Die Frontalansicht des Bomberman (statt den normalen Augen wurden Kreuze angezeigt), und das gleiche Bild mit einer weißen Umrandung. Dies verdeutlicht sowohl das Sterben, und zeigt auch an, dass die Figur aktuell unverwundbar ist. Bei einem ersten Test wurde dann allerdings festgestellt, dass eine weiße Umrandung auf weißem Hintergrund nur schwer erkennbar ist. Die Umrandung wurde daraufhin in schwarz umgeändert. Sollte ein Bomberman sterben und keine weiteren Leben mehr besitzen, wird nach den 300 Frames die Spielfigur nicht mehr angezeigt, die Steuerung deaktiviert und die Steuerfläche mit dem Hinweis versehen, dass der Spieler leider verloren hat.

3.7.2 Siegbedingungen

In der Field Klasse wurde eine neue Funktion angelegt, die abprüft, ob nur noch ein Spieler von den 4 Spielern übrig ist. Hierzu wurden für die Bomberman Klasse zwei neue Attribute festgelegt: playing und alive. Beide Werte sind standardmäßig false, und werden erst bei Spieleinstieg durch den Konstruktor auf true gesetzt. Das Attribut alive wird genutzt, um festzustellen, ob der Spieler noch aktiv am Spiel teilnimmt. Sobald der Bomberman aktiv ist, wird der Wert auf true gesetzt, und sobald der Spieler sein letztes Leben verliert, wird der Wert wieder auf false gesetzt. Das Attribut playing wird nach dem Setzen auf true durch den Konstruktor dauerhaft auf true gelassen, um ermitteln zu können, ob der Spieler bzw. Bomberman irgendwann am Spiel teilgenommen hat. Wenn also sowohl alive als auch playing false sind, ist der Bomberman noch nicht aktiviert worden. Sind beide Variablen true, ist der Bomberman aktiv und noch im Spiel. Ist alive false, und playing true, hat der Bomberman am Spiel teilgenommen, ist inzwischen aber bereits ausgeschieden. In der ersten Version wurde die Funktion für die Siegbedingung alle 60 Frames aufgerufen. Die Funktion hat dann überprüft, ob alle 4 Spieler am Spiel teilgenommen haben (bei allen musste playing true sein), und ob nur noch ein Spieler am Leben ist (bei 3 Spielern alive auf false). Als schnelle Notlösung war diese Version in Ordnung, hatte jedoch noch einige Probleme: Es konnte nur ein Sieger gefunden werden, wenn 4 Spieler teilnahmen. Starben die letzten zwei Spieler zudem innerhalb des 60 Frames Intervalles, gab es keinen Sieger. Einer der Tester machte den Vorschlag, die Funktion nur aufzurufen, wenn ein Spieler ausscheidet. Dieser Vorschlag wurde dann auch umgesetzt. Die Änderung erleichterte die Abfrage, und ermöglichte es zudem auch, einen Sieger zu ermitteln, wenn nur 2 oder 3 Spieler gespielt haben. Wenn ein Spieler ausscheidet, wird sein alive Attribut wieder auf false gesetzt. Wenn jetzt nur noch ein weiterer Spieler am Leben ist, wird er zum Sieger ernannt. Das playing Attribut wurde trotzdem weiterhin benutzt, und dadurch konnte ermittelt werden, ob die Spieler einfach nur nacheinander allein gespielthaben. Wenn bei allen Spielern playing

auf true und alive auf false gesetzt war (was nur passieren kann, wenn ein Spieler allein spielt und sich selbst oft genug tötet, bevor der nächste Spieler dazu stößt), kommt statt einer Siegmeldung der Hinweis, dass man doch bitte gemeinsam, und nicht nacheinander spielen soll.

Abbildung 3.1: Screenshot des Bomberman-Spiels



4 User Tests

4.1 Erster Test

4.1.1 User Feedback

Steuerung

Um die praxisnauglichkeit unseres Bombermann Spiels zu testen, gab es einen User Test. Dadurch sollten wir die Einschätzung von unbeteilichten Personen erhalten, was uns aufschlüsse über die Spiellogik und die Funktionsfähigkeit geben sollte. Der erste Usertest wurde von einer Testgruppe, bestehend aus 4 Personen, durchgeführt. Anfangs wagten sich die Tester nur sehr vorsichtig an die Bedienung des Trackballs und des Bombuttons und nutzten Anfangs nur eine Hand, im weiteren Verlauf dann doch beide Hände.

Spiel

Das Sprengen von genügend Blöcken für einen Kontakt der Spieler dauerte sehr lang und aufgrund der geringeren Entfernung kam es nur zu Zweikämpfen direkt gegenüberliegender Spieler. Power Ups die dem Spieler ermöglichen mehrere Bomben gleichzeitig zu legen wurden nicht direkt erkannt und es wurde eher Zufällig entdeckt. Das gezielte abbiegen/ um die Ecke laufen erwies sich als große Herausforderung, weshalb der Außenrand nur selten verlassen wurde. Dadurch wurden auch die Speed Power Ups gemieden, welche es aufgrund der höheren Bewegungsgeschwindigkeit ebenfalls erschwerten abzubiegen. Die meisten Tode

kamen durch selbst Verschulden zustande und weniger durch gegnerischen Einfluss. Dies führte auf dem fast vollständig freien Feld am Ende dazu, dass die Spieler sich auf dem Außenrand bewegten und alle Bomben hintereinander legten. In dieser Phase war es fast nicht möglich einen Gegner mit eigenen Aktionen gezielt auszuschalten.

Zusätzliches User Feedback

Ein Tester hatte Probleme die Übersicht über seine Spielfigur zu behalten, als er auf der gegenüberliegenden Seite unterwegs war. Die Lebensanzeige war nicht deutlich und es kam die Frage auf, ob die Anzeige die 0 entahlten sollte oder nicht. Die Anfangsphase war sehr langsam und träge, weil die Dauer bis man eine neue Bombe legen konnte sehr lang war.

Anschließende Änderungen

Um Verbesserungen an der Spieldauer vorzunehmen, wurde der Radius gelegten Bomben auf maximal 9 erhöht und die Zeit bis man wieder eine Bombe bekommt auf 5 Sekunden verringert. Um Spieler durch das legen eigener Bomben bekämpfen zu können wurde eine Änderung vorgenommen die es verhinderte das ein Spieler über Bomben laufen kann. Die Blöcke werden mit zufälligen Lücken verteilt, wodurch ca 30% weniger Blöcke auf dem Spielfeld sind. Das soll die Zeit verkürzen bis Spieler aufeinander treffen. Die Empfindlichkeit für das Abbiegen wurde verringert. Für das Problem der Lebensanzeige entschied man sich für die allgemeine Umsetzung, das eine Anzeige von „0 Leben“ zulässig ist und dann letzte Leben darstellt.

4.2 Zweiter Test

Im zweiten User Test wurde wieder mit einer 4er Gruppe getestet. Auch hier wurden beide Hände nicht von Anfang eingesetzt, nach einer kurzen Eingewöh-

nungszeit arbeitet dann doch alle Tester aktiv mit beiden Händen. Die Tester waren vollständig auf das Spielkonzentriert und es gab keine Überraschenden Vorfälle, die das Spielgefühl negativ beeinflusst hätten.

4.2.1 User Feedback

Die empfindlichkeit der Steuerung beim abbiegen an einer Ecke war zwar immernoch etwas unzuverlässig, aber nach kurzer Eingewöhnungszeit durchaus annehmbar. Die Umsetzung des Spielkonzeptes funktioniert und macht Spaß. Da der Test die praxistauglichkeit des Bombermann Spiels zeigte, wurden anschließend keine Veränderungen mehr am Spiel selbst vorgenommen.