TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik
Professur für Medieninformatik

Teamorientiertes Praktikum

Entwicklung eines Bomberman Spieles für einen Tabletop

Tobias Gall Björn Nauman Sebastian Arndt
Micahel Swora

Chemnitz, den 16. Dezember 2015

Betreuer Dipl.-Inf. Michael Storz

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Praktikumsbericht

1.1 Grundgerüst

Basierend auf Processing und dem Simple Multi-Touch(SMT) Toolkit für Processing entstand das Grundgerüst für das Bomberman Spiel. Als Einstiegspunkt für die graphische Anwendung muss die Klasse `PApplet` aus Processing erweitert werden. Die Implementierung erfolgt in der Klasse `Field`, zudem muss diese Klasse zwei Methoden implementieren. Zum einen die `setup`-Methode, in der alle Rahmenbedingungen zur Erstellung der Anwendung definiert werden und zum anderen die `draw`-Methode, die kontinuierlich vom Framework aufgerufen wird um die graphische Anzeige zu generieren. Zusätzlich liest der Konstruktor der Klasse bereits die Einstellungen aus einer `properties`-Datei ein und er erstellt eine Hashmap mit allen verwendeten Bildern, um auf diese später ohne zusätzliche Dateizugriffe zuzugreifen.

In der `setup`-Methode wird als erstes die Auflösung der Anwendung und gleichzeitig die Rendering Engine definiert. Letzte 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 Bildschirmes 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

Steurung 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 direkt eine Spielfigur, eine Instanz der `Bomberman`-Klasse zugeordnet. Der Nutzer kann über die Steuerungsmöglichkeiten der `UserArea` direkt mit seiner Spielfigur interagieren. Um das Spiel für vier Spieler zugänglich zu machen, sind jeweils zwei Steuerungen auf der langen Seite des Bildschirmes positioniert. Die weiteren zwei Steuerungen befinden sich gespiegelt an der gegenüberliegenden Seite des Bildschirmes.

1.1.1 Spielfeld

Das grundlegende Spielfeld wird über die Klasse `Blocks` dargestellt. Die Klasse verwaltet das zweidimensionale Blockraster, in dem sich die Spielfiguren bewegen und ihre Aktionen ausführen. Jeder einzellne Block gehört zu der gleichnamigen Klasse. Die Blöcke untergliedern sich in verschiedene Typen: verdeckter Block, leerer Block, Block mit Gegenstand, statischer Block, Block mit Bombe. Auf leeren Blöcken kann der Nutzer die Spielfigur bewegen. Beim Start des Spieles existieren leere Blöcke um die Startposition der Spielfigur herum. Von dieser Startposition aus muss der Nutzer die verdeckten Blöcke wegsprengen, um sich weiter durch das Spielfeld bewegen zu können. Das Sprengen erfolgt durch das Legen einer Bombe. Der Block im Raster, in dem die Bombe gelegt wurde, wird als Block mit Bombe dargestellt und animiert. Nun hat der Spieler rund 3 Sekunden Zeit um seine Spielfigur aus dem Sprengradius der Bombe zu bewegen um zu überleben. Wird ein verdeckter Block verstört, kommt entweder ein Block mit einem Gegenstand oder ein leerer Block zum vorschein. Die Verteilung der Gegenstände unter den verdeckten Blöcken erfolgt zufällig. Dabei verteilt der Algorithmus im Bereich um die Startposition der einzelnen Figuren mehr Gegenstände als in der Mitte des Spielfelden. Neben den sprengbaren Blöcken existieren die statischen Blöcke. Sie werden über das Raster so verteilt, dass zwischen diesen immer ein leerer oder verdeckter Block liegt.

Ein einzellner Block wird in der gleichnamigen Klasse durch das Processing Element `PShape` dargestellt. Die größe eines Blockes berechnet sich dynamisch anhand der Displaygröße und der Dimensionen des Spielfeldes. Dadurch skaliert das Blockraster mit der Auflösung des Bildschirmes. Der Type des Blockes wird über sein Attribut `type` festgelegt. Anhand dieses Types wird der Block mit einer bestimmten Textur angezeigt. Mit Settern und Gettern, können zudem vom Block weitere Attribute abgefragt und gesetzt werden. So ist es für die Bewegung der Spielfigur wichtig, ob diese sich auf einen bestimmten Block bewegen darf oder nicht. Die Methode `isWalkable` berechnet diese Eigenschaft aus dem Typen des Blockes und ob dieser verdeckt ist.

1.1.2 Bewegung der Spielfigur

Die Spielfiguren werdem vom Spieler über die `UserArea` gesteuert. Der Spieler hat die Möglichkeit seine Figur innerhalb des Blockrasters auf den begebaren 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 virtuellen 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 überprüft, ob sich der Rand der Spielfigur in einen Block bewegt, 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 Tolleranzberieches 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.

2 Bewegungsanimation

2.0.3 Was war bereits vorhanden?

Als die Arbeit an der Bewegungsanimation begann, war bereits das Grundgerüst vorhanden. Es gab also schon verschiedene Blöcke auf dem Spielfeld, ebenso waren bereits Abfragen eingebunden, um zu ermitteln, welche Blöcke von einem Bomberman besucht werden können. Diese Abfragen waren in Funktionen eingegliedert, namentlich moveright(), moveleft(), moveup() und movedown(). Sie sollten ausgeführt werden, wenn man den jeweiligen Richtungsbefehl erhält. Da zu diesem Zeitpunkt allerdings noch keine Bewegungssignale von den Touchzones übermittelt wurden, wurde zu Testzwecken ein Bomberman jeden Frame in die gleiche Richtung laufen gelassen. Zudem gab es bereits einen Bomberman als Platzhalter, der sich ohne jede Animation mittels der oben genannten move Funktionen über das Spielfeld bewegen lässt.

2.0.4 Grundidee

Die Animation sollte insgesamt eher minimalistisch werden, jedoch trotzdem einen gewissen Charme versprühen. Damit man überhaupt eine Animation erkennen kann, 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.

2.0.5 Bilder Entstehung

Als Grundlage für den Entwurf der Animationsbilder wurde der Platzhalter genommen: ein einzelner Bomberman in der Frontalansicht, ohne erkennbare Beine. Eine Bewegungsanimation kann man jedoch am ehesten an einer Bewegung der Beine erkennen. Der erste Schritt war also, dem Bomberman Beine zu verpassen. 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 entworfen, 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 kur-

zerhand 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.

2.0.6 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 bewegen soll. 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 werden soll. Vom Touchpanel sollte immer eines von 5 verschiedenen Signalen kommen: direction=0 für stehend, direction=10 für Bewegung nach oben, direction=20 für Bewegung nach unten, di-

rection=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 der Bomberman sich allerdings in eine bestimmte Richtung bewegen sollte, 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.

2.0.7 Erste Tests

Beim ersten Nutzertest wurde die Bewegungsanimation als angenehm bewertet, es gab insgesamt keine weiteren Auffälligkeiten in diesem Bereich.

2.0.8 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 Entwicklung der Bomben & deren Animation

3.0.9 Was war bereits vorhanden?

In dem Grundgerüst war es bereits möglich, den einzelnen Feldelementen verschiedene Zustände zuzuweisen: „STATIC“ für einen nicht wegsprengbaren Block, „BLOCK“ für einen sprengbaren Block, „EMPTY“ für ein leeres, begehbares Feld, und es gab schon einen ersten Platzhalter für die späteren Upgrades. Zudem konnte man den einzelnen Feldelementen den Zustand „BOMBE“ zuweisen.

3.0.10 Grundidee/Umsetzung

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. Da in dieser Richtung kein wirklich schöner Lösungsansatz gefunden wurde, 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. Es wurde also ein

Feldbenötigt, welches 20 Objekte der Bombenklasse verwalten kann. Dieses Feld wurde in der field- Klasse erstellt. Jeder Bomberman hatte 5 festgelegte Feldelemente, die seine Bomben abspeicherten (z.B. Spieler rot die Feldelemente 0-4). Jede Bombe hatte 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.

3.0.11 Bombenanimation

Die Animation der Bomben sollte wieder recht 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 sollte ungefähr 3 Sekunden (also 180 Frames bei 60 fps) benötigen, bis sie selbstständig explodiert. Es sollte eine entsprechende Zahl auf der Bombe erkennbar sein (die 3 in grün, die 2 in gelb, die 1 in rot), die anzeigt, wie lange es noch dauert. Zudem sollte noch eine kleine Animation eingebunden werden, 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 sollen sich in der ersten Sekunde in einem angebrachten Takt abwechseln. 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 sollten dann in einem schnellen Intervall abwechselnd angezeigt werden. Im Quellcode wurde die Animation dann recht simpel 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.

3.0.12 Algorithmus zum Wegsprengen der Blöcke/Bomberman

Nach Ablauf des Countdowns sollte die Bombe explodieren und alles in ihrem Radius zerstören. 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 sollte sich bei „freier Bahn“ (kein BLOCK/ STATIC/coveredItem/Feldende in den 4 Richtungen in der Reichweite) komplett bis zu dem Radius entfalten, ansonsten entsprechend früher stoppen. Die Flammen der Explosion sollten ungefähr 1 Sekunde bestehen bleiben, 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 eingegan-

gen. 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. Der Grund dafür war recht simpel: wenn bereits zu Beginn die Blöcke entfernt wurden, wurden im nächsten Schleifendurchlauf auch noch dahinter liegende Blöcke entfernt. Da aber nur der erste erreichte Block in jede Richtung weggesprengt werden sollte, musste dies unterbunden werden.

3.0.13 Erste Tests

In den Usertests lief das Spiel jeweils in deutlich niedrigerer fps als die geplanten 60, wodurch die Animation der Bomben deutlich länger andauerte, als eigentlich geplant. Zudem kam von den Testern der Vorschlag, dass es nicht mehr möglich sein sollte, über gelegte Bomben zu laufen. Der letzte Vorschlag wurde bis zum zweiten Usertest eingebunden, und hat den Testern deutlich besser gefallen.

3.0.14 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.

4 Siegbedingungen

Die erste Version der Siegbedingungen entstand recht kurzfristig vor dem ersten Usertest. Die Variante war daher noch nicht wirklich gut durchdacht, funktionierte allerdings erst einmal. 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 sollte dazu dienen, um festzustellen, ob der Spieler noch aktiv am Spiel teilnimmt. Sobald der Bomberman aktiviert wurde, wurde der Wert auf true gesetzt, und sobald der Spieler sein letztes Leben verloren hatte, wurde der Wert wieder auf false gesetzt. Das Attribut playing wurde 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 inner-

halb 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.

5 die Ideenfindung

Nach dem ersten Kontakt der am Projekt interessierten Personen musste eine Idee für das teamorientierte Praktikum gefunden werden. Dazu wurde beim ersten Treffen eifrig überlegt, was für Anwendungen für einen Tabletop denn sinnvoll wären und was wir uns zutrauen würden. Die Ideenpalette reichte dabei von einem "Mensch-ärgere-dich-nicht"-Spiel und ... über ... bis zu einem Spiel, welches wir letztendlich realisieren wollten. Es ist ein Spiel, welches man gemeinsam an einem so riesigen Tisch spielen kann, welches Spaß macht und den sozialen Austausch der Spieler miteinander fördert. Kurz gesagt: ein bekanntes Spiel, bei dem der Spaß im Vordergrund steht. Es handelt sich um ein Bomberman-Spiel. In irgendeiner Art sollte jeder auch schon einmal Kontakt mit einem Bomberman-Spiel gehabt haben und deshalb auch ohne große Erklärung wissen, worum es hierbei geht. Doch selbst wenn dies nicht der Fall ist, so sollte das Spiel einfach zu verstehen und ohne große Erklärung spielbar sein.

6 der erster Entwicklungsschritt - ein 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 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 werden und ein Fahrplan für die Entwicklung entworfen werden. Was sind die elementaren Bestandteile des Projekts, was muss zuerst entwickelt werden und was ist optional? So fingen auch wir mit dem Entwurf eines Papier-Prototypen 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 man eine gewisse Anzahl an Leben haben, um nicht sofort nach der ersten erfolgreichen Attacke eines Gegners aus dem Spiel zu sein. Eine während der Re-

cherche gefundene Erweiterung war 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 dem Paper Prototype begonnen. Es wurde gezeichnet, geschnitten, gemalt und die Ideen aller zu einem Ganzen verarbeitet. Während der Entwicklung des Prototyps 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 Steuerungsfeld 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, was jedoch im fertigen Spiel noch erweitert wurde. Damit die Spieler mit ihren Bomberman nicht den Durchblick 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 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 Bastelei 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.

6.1 erste Spielversuche

Der erste Test des Spiels in der Papierversion fand mit nicht involvierten Spielern statt und verlief trotz des großen Aufwands recht erfolgreich. … So konnte im Anschluss die Umsetzung des Bomberman-Spiels in die digitale Form beginnen.

7 die Entwicklung der 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 nämlich die ganze Zeit in Touchreichweite zur 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 erhält man nicht wie mit einer Tastatur eine spürbare Reaktion auf die Eingabe. Man kann somit nicht fühlen, ob bzw. welche Taste man gerade betätigt und wäre so wohl zum öfteren Hinsehen gezwungen. Außerdem muss man bei virtuellen Pfeiltasten mehrere Finger benutzen, die Finger recht genau platzieren und jeder Spieler hat dabei wohl eine andere Geschicklichkeit. Also fiel die Entscheidung zugun-

sten 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.

7.1 erste Versuche der Entwicklung mit Processing

Auf der Suche nach einem passenden Tool für die Entwicklung des Spiels wurde zuerst eine Programmiersprache in Betracht gezogen, die auf Grafiken und Animationen ausgerichtet ist. Sie nennt sich Processing und wurde als quelloffenes Projekt am Massachusetts Institute of Technology entwickelt. Da sie eine Beschreibung von Formen und Objekten darstellt, 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 haben, 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 dabei 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 zunächst in einer Setup-Funktion eine Leinwandgröße festlegen und anschließend kann man in der Draw-Funktion Befehle einfügen, die in jedem Frame ausgeführt werden. Für einen Trackball muss man dabei eine Linienfarbe und eine Füllfarbe festlegen und kann nun im Anschluss die eigentliche Form zeichnen. Im konkreten Projekt bedeutete dies einen großen Kreis ohne Füllung zu zeichnen (nachfolgend: Außenkreis), zwei Diago-

nalen hindurch zu ziehen damit Viertel entstehen, einen kleinen Kreis in der Mitte zu platzieren um eine Ruheposition zu simulieren 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. Die Umsetzung dessen war auch gar nicht so schwer, jedoch verlangte sie ein wenig Mathematik. Zeichnet man nämlich einen Trackball wie gerade beschrieben, so folgt dieser der Maus 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 läuft 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.

7.2 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 auch einen Export der geschriebenen Projekte in Java, wodurch der Schritt nicht zu schwierig war. Nach dem Exportieren wurde der erzeugte Code in eine neue Java-Klasse eingefügt und mit minimalen Veränderungen lief das Ganze dann auch.

7.3 Erweiterung und Anpassung der Steuerung

Nach dem erfolgreichen Einbau der Steuerung in Java, mussten die bis jetzt einzeln entwickelten Teile zusammengefügt werden. Das hieß, dass nun echte Funktionen der Bomberman angesprochen werden mussten, wenn sich der Trackball bewegt und auch die Bomben müssen von den Spielfiguren gelegt werden können. Zum Erreichen dessen musste nur die bisherige Ausgabe der Richtung 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 man so weiß, wie vorsichtig man 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 seine 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.

8 User Tests

8.1 User Test 1

8.1.1 User Feedback

Steuerung

Die Bedienung der Steuerelemente (Trackball und Bombbutton) waren nicht direkt ersichtlich. Die Tester mussten durch probieren die Bedeutung der Buttons testen. Die Steuerung wurde teilweise nur zögerlich einhändig benutzt, dieser Umstand verbesserte sich nach einer kleinen Eingewöhnungszeit.

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 leg-

ten. In dieser Phase war es fast nicht möglich einen Gegner mit eigenen Aktionen gezielt auszuschalten.

Zusätzliches Tester 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.

- Steuerung nicht direkt klar
- teilweise Steuerung nur mit einer Hand $\rightarrow$ später beidhändig
- zu lange Anfangsphase / nur Zweikampf gegenüberliegender Panels
- lange Wartezeit für neue Bomben
- Möglichkeit mehrere Bomben nach Upgrade zu legen nicht genutzt
- „verloren gehen“ der eigenen Figur
- abbiegen/ um die Ecke laufen sehr schwer $\rightarrow$ Speedupgrades wurden gemieden
- Spielende nicht absehbar (Bombenradius zu klein, Möglich über Bombem zu laufen)
- Lebensanzeige nicht ganz eindeutig

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.

- Spielfeld wird nicht komplett mit Blöcken bedeckt
- Regeneration neuer Bomben verringert
- Empfindlichkeit für abbiegen verbessert
- Bomben können nicht mehr überlaufen werden
- Lebensanzeige: zeigt die tatsächlichen Leben an 0 Leben → letztes Leben

- Bombenradius auf maximal 9 erhöht
- eingesammelte Upgrades „fliegen“ zu dem Spieler der sie eingesammelt hat.

8.2 User Test 2

8.2.1 User Feedback

- Steuerung nicht ganz klar

- Trackball wurde nicht gehalten, sondern nur eine Toucheingabe gemacht, die keine Auswirkung hatte
- Erhöhung der allgemeinen Spielgeschwindigkeit sorgte wieder für Probleme beim abbiegen
- Anschließende Änderungen

9 Literaturverzeichnis