



Spielkonzept und Ausarbeitung

Jonas Temmen
Matrikelnr 1849460
WiSe 06|07 HS Bremen
Media Engineering
Mirko Haalck

EINLEITUNG	3
GRUNDÜBERLEGUNGEN	3
Story.....	3
Spielprinzip	4
Anforderungen	4
KONZEPTION	5
Zielsetzung.....	5
Zielgruppe.....	5
Techniken	6
Gestaltung.....	6
Umsetzung	8
Server	8
Client	9
PRODUKTIONSZEIT UND PROJEKTVERLAUF.....	11
AUSBLICK	13
ANHANG.....	14
Klassendiagramme	14

Einleitung

Erstellt werden soll ein interaktives Spiel, das sich durch eine innovative Steuerung abgrenzen, und so eine neue Form von Spielspass ermöglicht. Dabei sollen vor allem die sensorischen Fähigkeiten des Spielers angesprochen werden. Für die Eingabe wird dabei primär der „Sudden Motion Sensor“ benutzt, der bei neueren Notebooks in deren Festplatten integriert wurde. Dieser hat eigentlich die Aufgabe die Neigung des Notebooks im Raum zu ermitteln, um im Falle eines Sturzes die Leseköpfe von der Festplatte zu nehmen, um eine Beschädigung zu verhindern. In diesem Fall dienen die Daten des Sensors dazu, die X & Y Neigung zu ermitteln um so die Bewegung der vom Spieler gesteuerten Figur zu realisieren. Als sekundäre Eingabe wird die Akustik, also ein Mikrofon verwendet. Aktionen die der Spieler auslösen möchte, sind von dem von ihm verursachten Geräuschpegel abhängig. Überschreitet er einen definierten Wert, wird die Aktion ausgelöst, unterschreitet er ihn, wird die Aktion gestoppt bzw. nicht ausgelöst.

Der Spieler schlüpft dabei in die Rolle eines schwer bewaffneten Ufos, das sich über eine Farmlandschaft durch die bereits erwähnte Steuerung fliegen lässt.

Um die eigentliche Machbarkeit zu prüfen, ist im Vorfeld bereits ein Test durchgeführt worden ob Daten, die vom Sensor kommen, schnell genug verarbeitet werden können, um daraus realistische und erwartungskonforme Bewegungen zu erstellen.

Grundüberlegungen

Die Spielidee basiert im Wesentlichen auf einem klassischen Prinzip: Außerirdische greifen die Erde an. Der Spieler bezieht diesmal seine Position an der Seite der angreifenden Aliens. Diese haben einen neuen Plan, um ihre Bemühungen endlich zu einem erfolgreichen Abschluss zu bringen.

Story

Nachdem Außerirdische seit Jahrzehnten immer wieder einen Frontalangriff versucht haben und dabei stets von furchtlosen Helden in pixeligen Raumschiffen vernichtend geschlagen wurden, wurde nun ein neuer Plan zur Unterwerfung der Erde entwickelt. Anstatt den Feind direkt anzugreifen und somit einen massiven Gegenschlag zu riskieren, sollen vereinzelte Angriffe auf wenig besiedelte Farmlandschaften den Feind aushungern. Dabei gilt es vorrangig, das Wesen zu eliminieren dem Aliens die größte Antipathie entgegenbringen: Die Kuh. Der Grund ist einfach: Menschen mögen Kühe.

Spielprinzip

Das Prinzip ist unkompliziert. Je Level gibt es eine Aufgabe, die zu Beginn kurz angezeigt wird. Diese gilt es zu erfüllen, um das Level zu beenden und zum nächsten zu gelangen. Die Aufgaben beschränken sich dabei auf die gezielte Eliminierung von Gebäuden oder Lebewesen.

Sobald der Spieler das erste Objekt beschossen, und damit seine Absichten deutlich gemacht hat, wird er selbst durch Raketen angegriffen. Da nur eine begrenzte Menge an Lebensenergie (Schutzschild) zur Verfügung steht, gilt es diesem Beschuss auszuweichen. Dabei muss der Spieler feinfühlig reagieren denn auch eine Kollision mit der Randbegrenzung (Wald) verringert die Lebensenergie. Ist diese aufgebraucht, gilt der Angriff als gescheitert und das Spiel kann von vorne begonnen werden.

Das Spiel verzichtet auf Faktoren wie Punkte, Highscore usw. Die Motivation soll durch das Spielprinzip an sich und durch witzige Levelziele erreicht werden und so für einen lang anhaltenden Spielspass sorgen.

Anforderungen

Um den Spieler bei seiner Mission zu unterstützen und die gute Benutzbarkeit zu steigern, sind für das Interface folgende Komponenten geplant:

- Eine visuelle Darstellung des Sensors. Hier sollte nach Möglichkeit eine Anzeige darüber Aufschluss geben, in wie weit der Spieler das Notebook neigt
- Eine ähnliche visuelle Darstellung für den Lautstärkepegel, den das Mikrophon zum gegenwärtigen Zeitpunkt misst. Des Weiteren sollte der Schwellenwert der die Waffe auslöst angezeigt werden, der zudem noch vom Spieler einstellbar ist um auf eventuelle Umgebungsgeräusche reagieren zu können. Ferner muss das Mikrophon kurzzeitig per Tastendruck komplett abschaltbar sein.
- Ein Balken, der die Lebensenergie darstellt.

Für die Spielbarkeit und den Spielspaß sind im Vorfeld folgende Überlegungen festgesetzt worden, die später in die Realisierung einfließen:

- Der Sensor, durch den gesteuert werden soll, unterliegt einem ständigen Lernprozess. In diesen kann nicht eingegriffen werden. Jeder Sensor „lernt“ die benutzerspezifische Art mit seinem Notebook umzugehen und verlagert dadurch den Ruhezustand. Des Weiteren ist er sehr empfindlich. Daher sollte zunächst bei jedem Start der Anwendung eine festgelegte Zeit lang der Sensor

- abgefragt werden, um einen Durchschnittswert zu ermitteln, der dann für die Dauer des Spiels als Ausgangslage verwendet wird. Dies muss bei jedem Start geschehen, da der Sensor, wie bereits erwähnt, stetig „lernt“.
- Um eine realistische Flugsimulation zu erhalten, darf das Ufo nicht direkt die Daten des Sensors übernehmen, sondern diese müssen in eine Beschleunigung des Körpers unter Verwendung einer Masse umgerechnet werden.
 - Die spätere Implementierung muss es zulassen, dass die Level verschiedene Ziele haben können.
 - Erweiterungen sollten berücksichtigt werden. D.h. es sollten Basisklassen geschrieben werden, von denen andere abgeleitet werden können, um Funktionen hinzuzufügen.
 - Insbesondere die Level sollten so implementiert werden, dass problemlos mehr bzw. unterschiedliche erzeugt werden können.
 - Sollte die Hardware des Rechners, auf dem die Anwendung gestartet wird, nicht ausreichend sein (kein Sensor / Mikrofon vorhanden), so sollte ein Start des Spiels verhindert werden, um Verwirrung zu vermeiden.
 - Abweichend von dem Ziel des jeweiligen Levels sollte jedes Objekt auf dem Spielfeld angreifbar sein.
 - Um den Aufwand der Levelgestaltung zu minimieren, sollen die Level einigermaßen selbstbestimmend sein. D.h. ein Level sollte per Skript zugewiesen bekommen, was es darzustellen hat, die Position der Objekte sollte aber frei gewählt werden, je nach Platz.

Konzeption

Zielsetzung

Das fertige Produkt soll eine für den Benutzer einfach zu startende Anwendung sein. Dabei sollte er keinerlei Einstellungen vornehmen müssen.

Da die Zielgruppe von vorn herein durch die Wahl des Sensors als Eingabegerät beschränkt wird, kann speziell für diese Plattform gearbeitet werden. Es werden nur Techniken eingesetzt, die keinerlei zusätzliche Software voraussetzen, als die Standardumgebung es zulässt.

Zielgruppe

Zu diesem Zeitpunkt kann mit Sicherheit gesagt werden, dass nur das Apple iBook, Powerbook, MacBook sowie das MacBook Pro über Festplatten mit dem *Sudden Motion Sensor* verfügen und auch mit einem

eingebauten Mikrophon ausgestattet sind. Daher beschränkt sich die Zielgruppe auf Besitzer eines solchen Notebooks.

Es wird versucht von diesen eine möglichst hohe Zahl anzusprechen. Dies soll über eine ansprechende Gestaltung und natürlich vor allem durch die innovative Steuerung erreicht werden.

Als Zielgruppe werden alle Menschen verstanden, die Spaß an einem Spiel haben, das ihre Fähigkeiten einmal auf andere Weise fordert, als es die Klassiker des Genres tun.

Die Altersgruppe des Spielers soll dabei möglichst nicht festgelegt sein. Vielmehr wird versucht durch die Gestaltung, eine Mischung aus Retro-Look und schlichter Oberfläche, ein großes Publikum anzusprechen.

Techniken

Für die erfolgreiche Umsetzung sind im Wesentlichen zwei Komponenten von Bedeutung:

- Ein Server der die Daten des Sensors ausliest und diese weiterleitet. Dieser wird in Java implementiert, da Java auf OSX-Systemen vorinstalliert ist. Für die Beschaffung der Daten des Sensors ist eine C-Klasse zuständig. Für diese existiert ein Java -Wrapper als OpenSource (siehe <http://www.shiffman.net/p5/sms/>).
- Durch die Implementierung dieser Klasse in einem ServerSocket auf einem freien Port werden die ermittelten Daten für einen Client zugänglich gemacht.
- Der Client wird in Flash als Projektordatei ausgespielt, somit ist auf dem Rechner des späteren Spielers nicht zwingend Flash notwendig.

Da laut Zielgruppe das Programm zunächst nur auf OSX-Systemen zum Einsatz kommt, wird der Client als Mac-Projektor ausgespielt. Die beiden eingesetzten Techniken erlauben es aber, bei einer Verbreitung der Sensorfestplatten auf andere Systeme eine angepasste Version auszuspielen. Auch dies war ein Grund für die Verwendung von Java in seiner Rolle als plattformunabhängige Programmiersprache.

Gestaltung

Generell soll das Design zwei verschiedene grafische Grundkonzepte haben. So soll das eigentliche Spielfeld, die darauf befindlichen Objekte sowie das Interface während des Spiels einem Retro-Look folgen. D.h. gespielt wird in einer Perspektive von Schräg oben (isometrisch bzw. Vogelperspektive). Die Objekte sind allesamt Sprites (Pixelbilder) und somit nicht texturiert. Dies dient vor allem dazu, die Realität zu verfälschen, um dem Spiel die Grausamkeit zu nehmen.

Im Gegensatz dazu sollen das Hauptmenü sowie die verschiedenen anderen Anzeigegrafiken (Gameover etc.) möglichst schlicht und futuristisch aussehen.

Die Wahl für Sprites als Darstellungsform für die Objekte ist im Wesentlichen auf die Verminderung der Produktionszeit (im Gegensatz zu 3D-Modellen) zurückzuführen. Des Weiteren eignen sich diese gut für die Frame-basierte Animation in Flash.

Die Vielzahl von Grafiken, die für die Realisierung gebraucht werden, werden alle mit Photoshop erstellt, lediglich die Animationsphasen der Tiere sowie die Gebäude und Bäume werden wegen Ermangelung von Kompetenz von Dritten eingekauft.

Umsetzung

Die Anwendung besteht aus zwei Komponenten: dem Server und dem Client. Der Client ist das eigentliche Spiel, der Server dient als Startpunkt und läuft im Hintergrund. Abbildung 1 zeigt den Verlauf Anwendung im groben Überblick

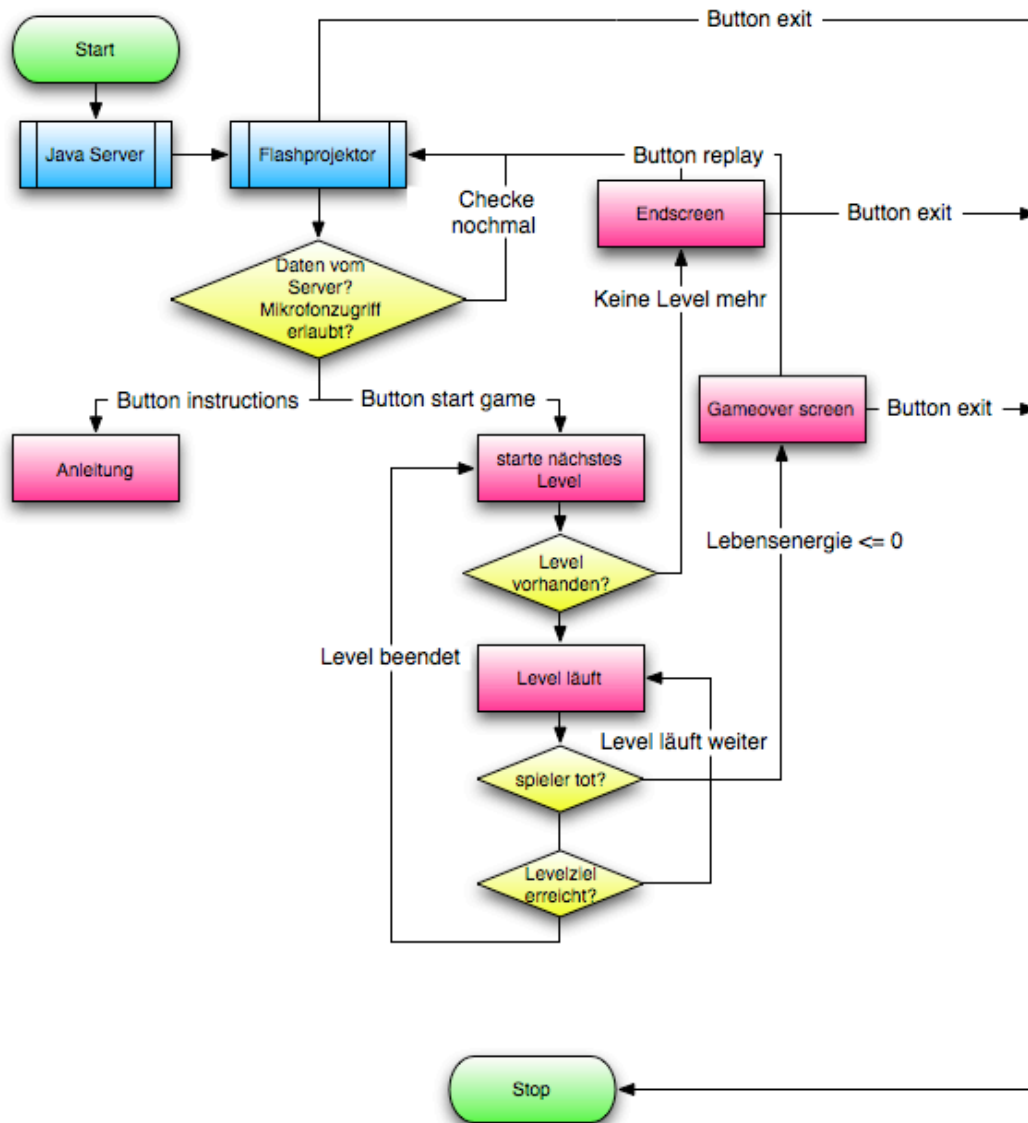


Abbildung 1 Programmablauf im Überblick

Server

Der Server ist zugleich der Startpunkt der Anwendung. Nachdem der Serversocket auf Port 6666 eingerichtet ist, startet er den Flashprojektor, um dem Anwender nicht das parallele Starten von zwei Anwendungen

zuzumuten. Wird der Projektor geschlossen, so wird auch automatisch der Server beendet.

Der Server wartet nach dem Start immer fünf ms auf eine eingehende Verbindung. Danach „schläft“ er für zehn Sekunden bevor er wieder reagieren kann, da die Methode *accept()* der Klasse *ServerSocket* blockierend ist. Das heißt: Im Falle eines dauerhaften Versuchs Verbindungen zu akzeptieren, würde der Server an diesem Punkt einfrieren und den Client nicht starten. Somit wird der Server als Thread umgesetzt und hat damit die Fähigkeit, die Ausführung des Programms für einen festgelegten Zeitraum zu unterbrechen. Sobald eine Verbindung erfolgt ist, wird der Server keine weiteren mehr akzeptieren und die Verbindungsprüfung überspringen. Stattdessen werden alle 100 ms die Daten des Sensors übermittelt.

Im Falle, dass der Client geschlossen wird, wird die dadurch ausgelöste *Exception* dazu benutzt, auch den Server zu beenden.

Client

Die Umsetzung erfolgt in einem objektorientierten Ansatz. Für jedes Objekt auf der Bühne, bei dem es Sinn macht, gibt es eine Klasse. Diese sind in Pakete unterteilt, um die Übersichtlichkeit für spätere Erweiterungen zu fördern. Nach Möglichkeit werden Klassenhierarchien gebildet, um bei mehreren ähnlichen Objekten (z.B. den Tieren) auf vorhandene Funktionen zurückzugreifen. Innerhalb der Frames wird kein Code verwendet, es sei denn, es handelt sich um Steuerungsanweisungen (*Stop()* bzw. *goTo()* Aufrufe).

Als zentrales Element des Spiels wird die Klasse *GameController* implementiert. Diese verwaltet die Levels, beginnt bzw. beendet sie. Von hier aus wird erkannt, ob ein Level wegen seines Zustands zu beenden ist oder er normal weiterläuft. Dazu wird pro Frame das zurzeit gestartete Level abgefragt.

Die einzelnen Level liegen in einem Array vor und werden zu Beginn der Anwendung initialisiert.

Alle Level werden durch eine Klasse repräsentiert, welche sich von der Hauptklasse *Level* ableitet. Die Klasse *Level* beinhaltet alle Funktionen, die für den Spielablauf benötigt werden. Diese müssen lediglich von den abgeleiteten Klassen aufgerufen bzw. für eine andere Levellogik überschrieben werden. So soll es beispielsweise das Standardverhalten eines Levels sein, dass alle Objekte eliminiert werden müssen. Die Klasse *Level* definiert dies in einer Funktion. Sollte das in einem konkreten Level nicht erwünscht sein, so muss die entsprechende Funktion überschreiben und ein eigenes Ziel angegeben werden. Dadurch bleiben die eigentlichen Levelklassen sehr schlank und übersichtlich, jedoch sind sie trotzdem anpassbar für Änderungen.

Neben den Funktionen, die ein Level aufrufen oder überschreiben kann, wird in der konkreten Klasse die Anzahl der jeweiligen, darzustellenden Objekte definiert und an eine darstellende Funktion der Vaterklasse übergeben.

Objekte, die auf der Oberfläche dargestellt werden und aktiv am Spielgeschehen teilhaben (Gebäude und Tiere), werden durch zwei Klassen repräsentiert: eine Darstellungs- und eine Logikklasse. Eine neue Instanz eines solchen Objekts wird durch den Konstruktor der Darstellungs-klasse erzeugt. Diese Klassen tragen den Namen der konkreten Implementierung (z.B.: Chicken, Cow, House usw.) Nach der Instanziierung holen diese den entsprechenden MovieClip aus der Bibliothek auf die Bühne, der mit der Logikklasse verlinkt ist. Das hat den Vorteil, dass auf diese Weise gleiche Objekte zusammengefasst werden können. So sind alle Tiere per se gleich, sie unterscheiden sich nur in Aussehen und Geschwindigkeit. Diese speziellen Merkmale werden in der Darstellungs-klasse definiert. Die Logikklasse übernimmt dann Aufgaben wie Kollisionsabfrage, Laufrichtung, Treffer durch den Spieler etc. Bei den Gebäuden verhält es sich nahezu identisch.

Um zu verhindern, dass die Tiere übereinander bzw. über die Gebäude laufen, wird nach der Initialisierung aller Objekte in der Levelklasse ein Array gebildet, in dem alle diese Objekte zusammengefasst werden. Dieses Array wird an alle Tiere weitergegeben. Pro Frame prüft jedes Tier selbstständig, ob eine Kollision mit einem anderen Objekt stattfindet. Sollte dies der Fall sein, so ändert das Tier seine Laufrichtung um 180° und bewegt sich für einen festgelegten Zeitraum (mindestens 10 Frames) in diese Richtung.

Alle Logikklassen leiten sich von der Hauptklasse *ObjectOnStage* ab, welche Funktionalitäten beinhaltet die für alle Objekte (Tiere wie Gebäude) gleich sind (Positionierung, Aktivierung etc.)

Das Ufo, das vom Spieler gesteuert wird, wird dann konsequenterweise auch in zwei Klassen (Darstellung und Logik) geteilt.

Bei Objekten, bei denen dies keinen Sinn macht, kann auf die Trennung verzichtet werden. Dazu gehören z.B. die beiden Klassen die die Sensoren repräsentieren. Hier ist es ausreichend, den entsprechenden MovieClip mit der Klasse zu verlinken, da er sich von Beginn an, die ganze Zeit auf der Bühne befindet.

Die Klasse Mikro prüft den momentanen Lautstärkepegel pro Frame. Diese Daten sind von außen per Funktion abrufbar. Dabei wird die Position des Schwellenwertreglers berücksichtigt. Die Funktion liefert so lange einen Pegel von 0 bis sich der ermittelte Lautstärkewert über der Schwelle befindet.

Zusätzlich gibt es die Möglichkeit zu prüfen, ob das Mikro eingeschaltet ist, um einen Start des Spiels ohne zu verhindern.

Die Aktualisierung der Darstellung des MovieClips, der die Daten visualisieren soll, kann direkt in dieser Klasse erfolgen.

Für die Darstellung der Bewegung werden zwei Klassen benötigt. Zum einen ist es notwendig eine Erweiterung zu der Flash-eigenen XMLSocket-Klasse zu schreiben, da mehr als nur die Standardfunktionen nötig sind. So sollte die Klasse nach erfolgreichem Kontakt zum Server zunächst einmal einen Durchschnittswert der Positionsdaten ermitteln, um sowohl das Lernen des Sensors als auch die momentane Position des Notebooks auszugleichen. Danach werden die empfangenen Positionsdaten in ein Objekt gespeichert, das von außen abgefragt werden kann. Dies geschieht in einer Klasse, die analog zu der Mikrofonklasse funktioniert. Wegen der dauerhaften Präsenz ist auch hier eine Trennung von Logik und Darstellung nicht notwendig. Je Frame werden die Daten aus der XMLSocket-Kasse geholt und die Visualisierung des verbundenen MovieClips angepasst. Von außen sind diese Daten per Funktion abrufbar. Zusätzlich ist auch hier eine Funktion implementiert deren Rückgabe angibt ob der Sensor Kontakt zum Server hat.

Beide Sensor-MovieClips sind der Logikklasse des Ufos bekannt. Je Frame werden beide abgefragt und die Daten benutzt, um die Handlung zu steuern. So wird je Frame eine Beschleunigung unter Berücksichtigung der Masse des Ufos aus den Bewegungsdaten errechnet und auf eine momentane Geschwindigkeit übertragen. Diese führt dann zu einer neuen Position des Ufos.

Sollte der Wert des Mikrofons über 0 liegen, wird die Waffe des Ufos ausgelöst und überprüft, ob etwas getroffen wurde. Dies geschieht wieder mit Hilfe des Arrays, in dem alle Objekte gespeichert sind.

Sollte ein Treffer vorliegen, kann direkt eine Funktion des Objekts aufgerufen werden, das die Sterbesequenz einleitet und das Objekt aus dem Array streicht. Parallel existiert ein anderes Array, in dem sich alle Objekte befinden, die zum Erreichen des Levelziels eliminiert werden müssen. Auch aus diesem Array wird das Objekt entfernt. Sobald dieses leer ist gilt der Level als beendet. In diesem Fall wird die Klasse GameController benachrichtigt. Diese starten dann den nächsten Level aus dem Array. Sollte es keinen weiteren Level geben, gilt das Spiel als beendet und eine entsprechende Grafik wird gezeigt.

Produktionszeit und Projektverlauf

Die Produktion erfolgt in mehreren Stufen, die alle mit einer sorgfältigen Testphase enden. Da die Anwendung viele Aktionen pro Frame

durchführt, muss auf die Performance geachtet werden und diese nach jeder Stufe überprüft und gegebenenfalls verbessert werden.

Die einzelnen Entwicklungsstufen im Überblick:

1. Server
2. Klasse zum Testen der Serververbindung
3. Objekt das sich auf Basis der Serverdaten bewegt
4. physikalische Korrektheit der Bewegung
5. Bewegung auf eingeschränkten Bereich begrenzen
6. Tiere, Gebäude (zunächst nur grob) und deren eigenständiger Bewegung, Platzierung, Kollision etc.
7. Mikrofon
8. Angriffslogik
9. Spielkontrolle (Levelstart)
10. Implementierung von zunächst zwei Leveln
11. Levelziele
12. Wechseln von Leveln
13. Lebensenergie
14. Gegner
15. Grafiken erstellen, einbinden und Animieren
16. Bugfixing
17. Feintuning an Aussehen, Spielmechanik, Motivationsfaktoren.

Die Phasen 1-8 bilden dabei das Grundgerüst der Spiellogik. Hier muss besonders sorgfältig gearbeitet werden, damit die späteren Erweiterungen problemlos implementiert werden können. Dafür wird ein Zeitrahmen von ca. drei Wochen notwendig sein.

Die eigentliche Steuerung des Spiels wird in den Phasen 9-14 hinzugefügt. Dabei wird der Code der bisherigen Stationen bei Bedarf verbessert um neuen Anforderungen gerecht zu werden. Der dafür vorgesehene Zeitrahmen beträgt vier Wochen.

Die Grafiken werden nach einer Liste, die während der ersten 14 Schritte gebildet wurde, erstellt. Dies sollte nicht länger als eine Woche dauern. Die Punkte 16 und 17 werden in einem Schritt abgehandelt. Durch die Tests nach jeder Implementierungsphase sollte es eigentlich zu keinen schwerwiegenden Fehlern im Programmablauf mehr kommen. Hier ist dann der Raum für neue Ideen, Verfeinerungen sowie eine Testphase mit freiwilligen Testspielern. Diese sollen Aufschluss darüber geben, wie das Spielprinzip ankommt, ob die Motivation ausreicht bzw. was verbessert werden muss. Diese Phase sollte noch einmal vier Wochen in Anspruch nehmen.

Danach lassen sich beliebig viele Level mit geringem Aufwand hinzufügen.

Insgesamt ergibt sich damit eine Produktionszeit von ca. drei Monaten.

Ausblick

Auch nach der Fertigstellung der ersten Version soll die Entwicklung weitergehen. So sind während der Konzeption bereits Ideen hinzugekommen, die wegen des knappen Zeitplans nicht in dieser Version umgesetzt werden können, aber hier trotzdem Erwähnung finden sollen.

Eine zukünftige Version soll um folgende Punkte erweitert werden:

- SoundFX
- Lebensenergie für Gebäude und Tiere
- Verschiedene Gegner
- Unterschiedliches Leveldesign (Eiswelten, Wüsten)
- Wolken, die die Geschwindigkeit des Spielers bremsen
- Levelsystem auf XML umstellen, um einen Leveleditor zu ermöglichen.
- Mehr Tiere

Anhang

Klassendiagramme

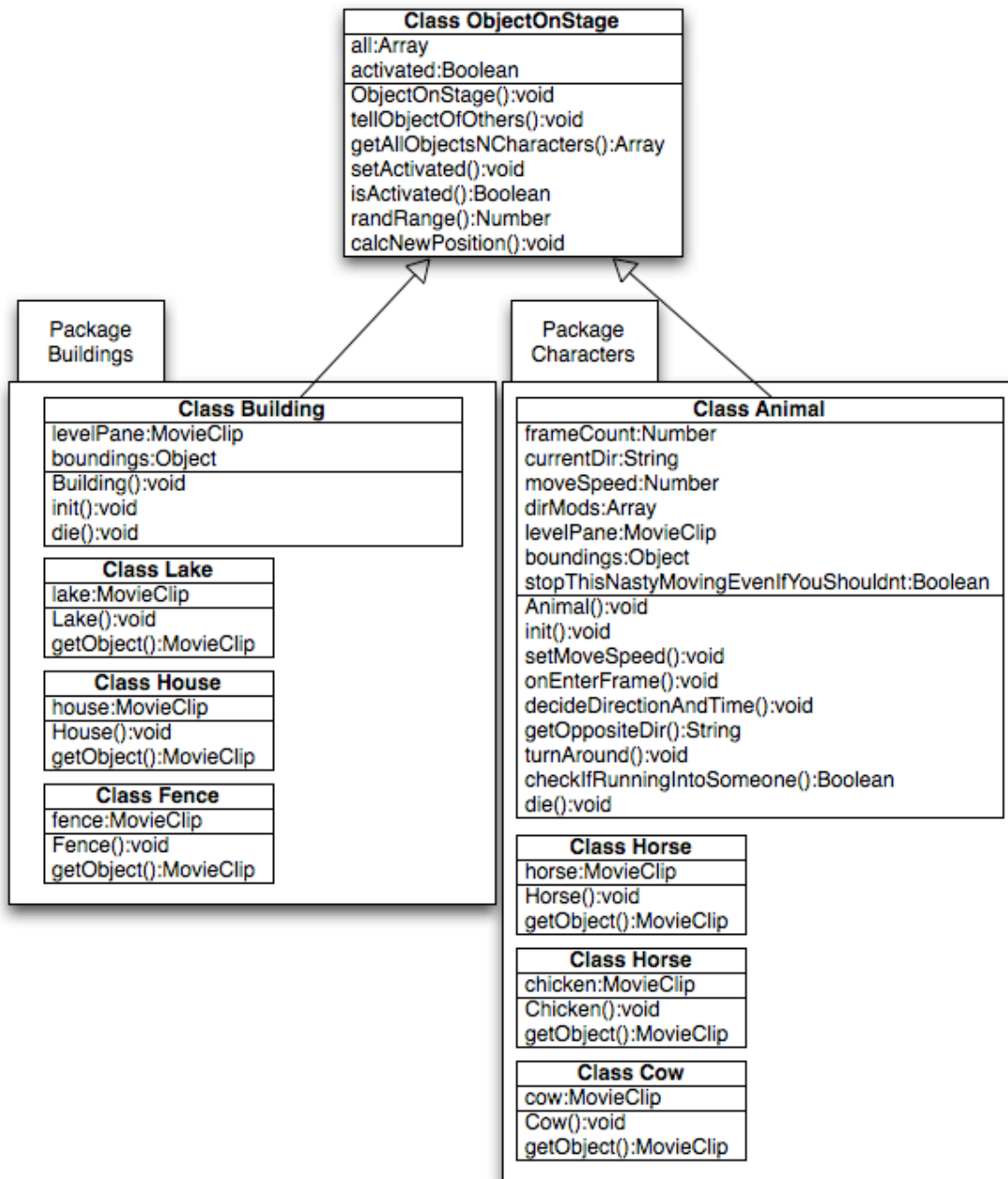


Abbildung 2 Packages Characters, Buildings und ObjectOnStage Klasse

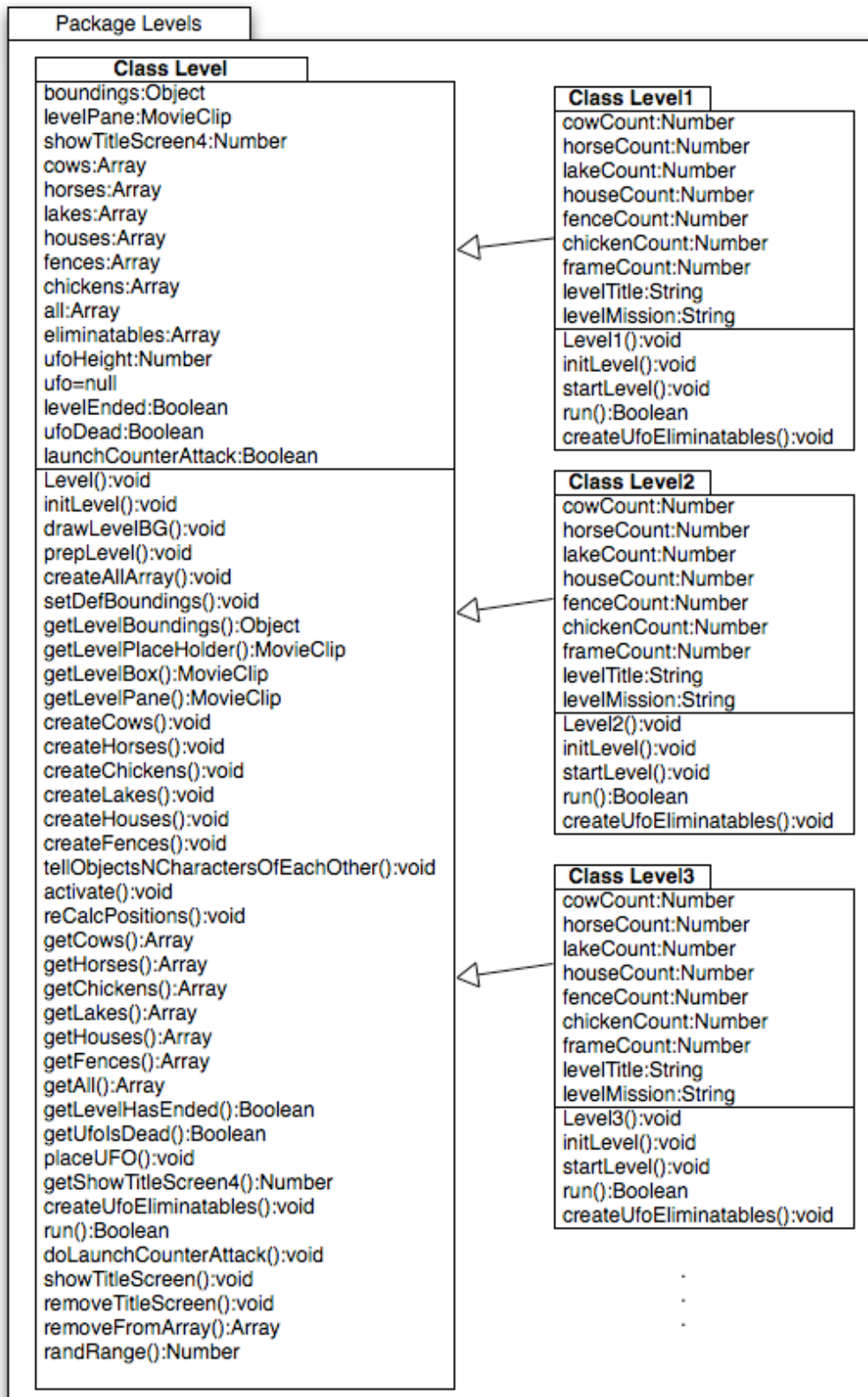


Abbildung 3 Package Levels

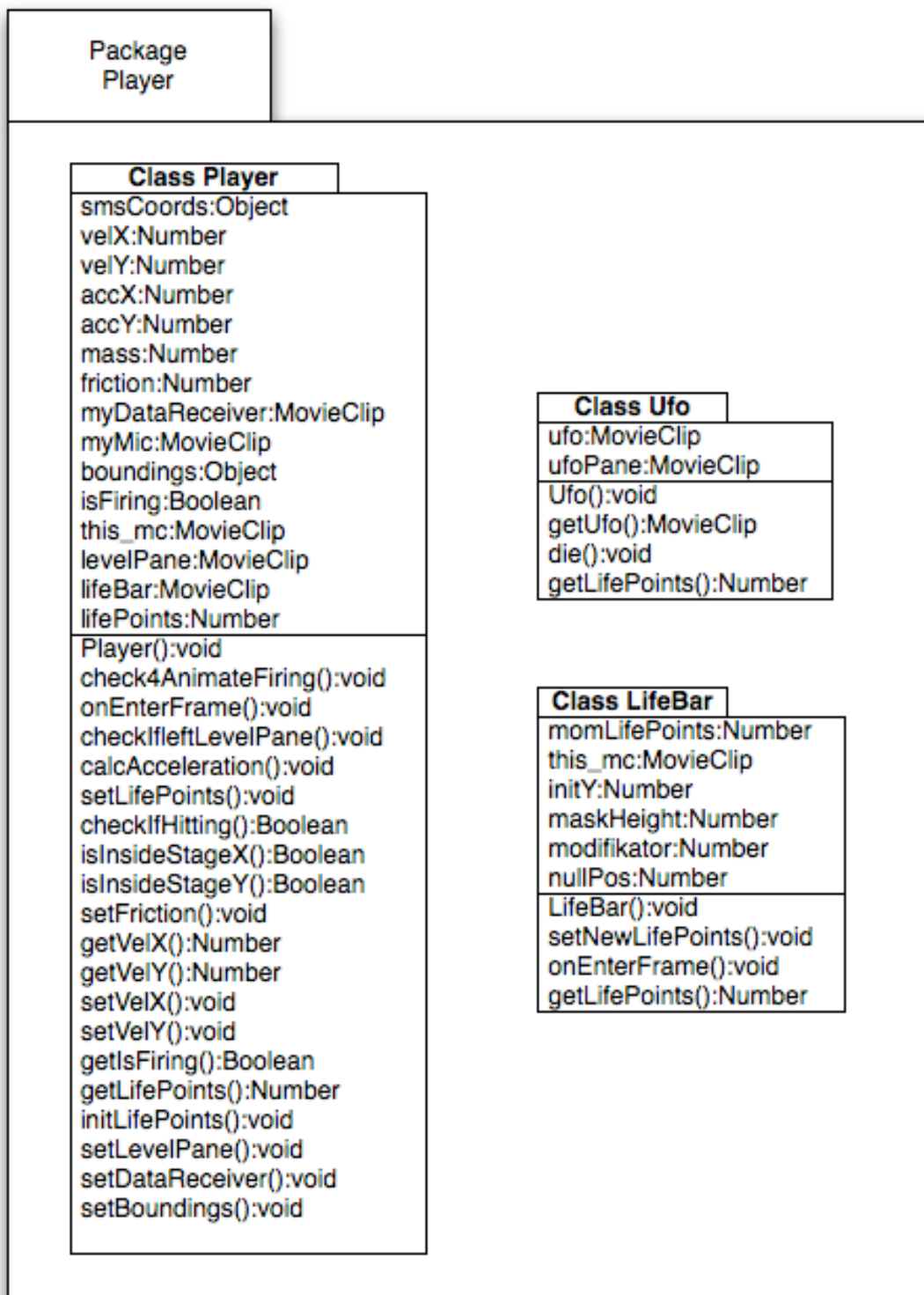


Abbildung 4 Package Player

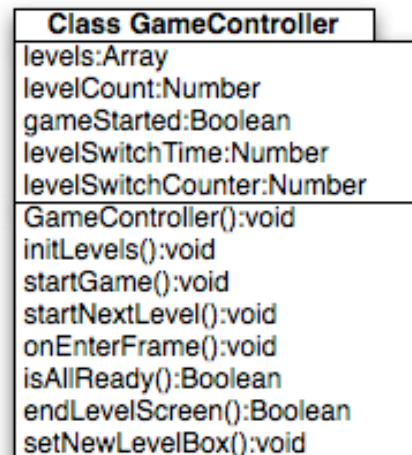
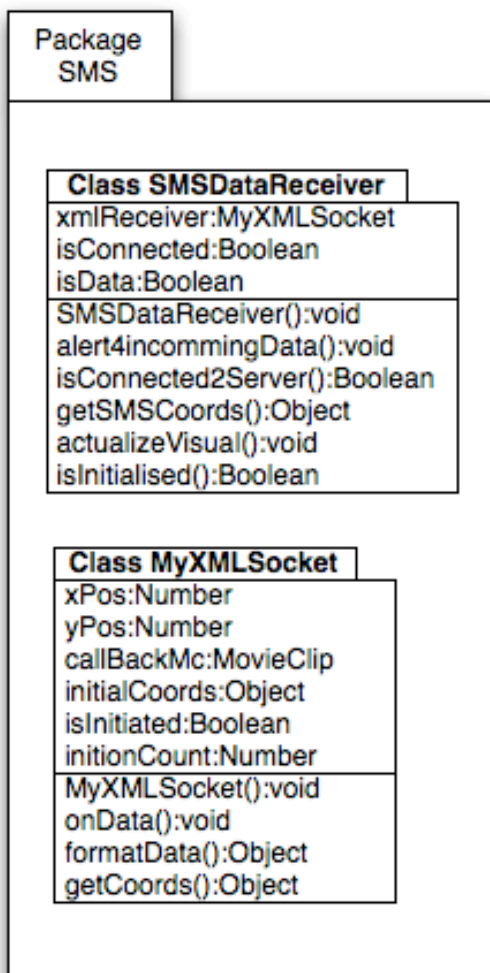
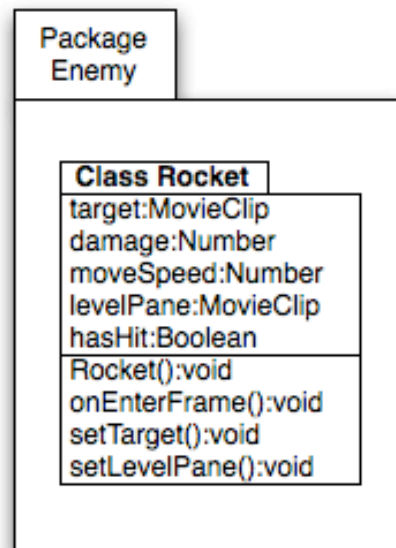
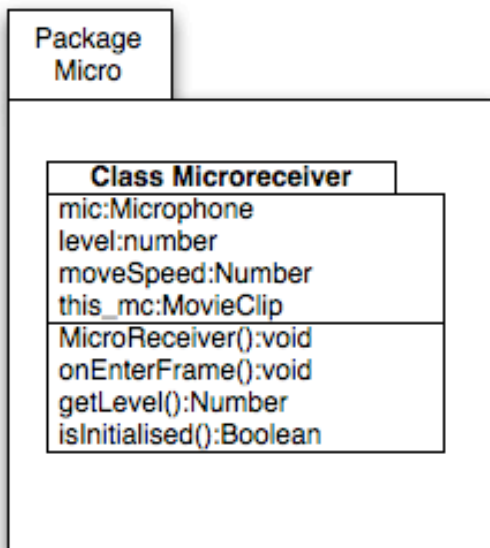


Abbildung 5 Packages SMS, Mikro, Enemy und Klasse GameController

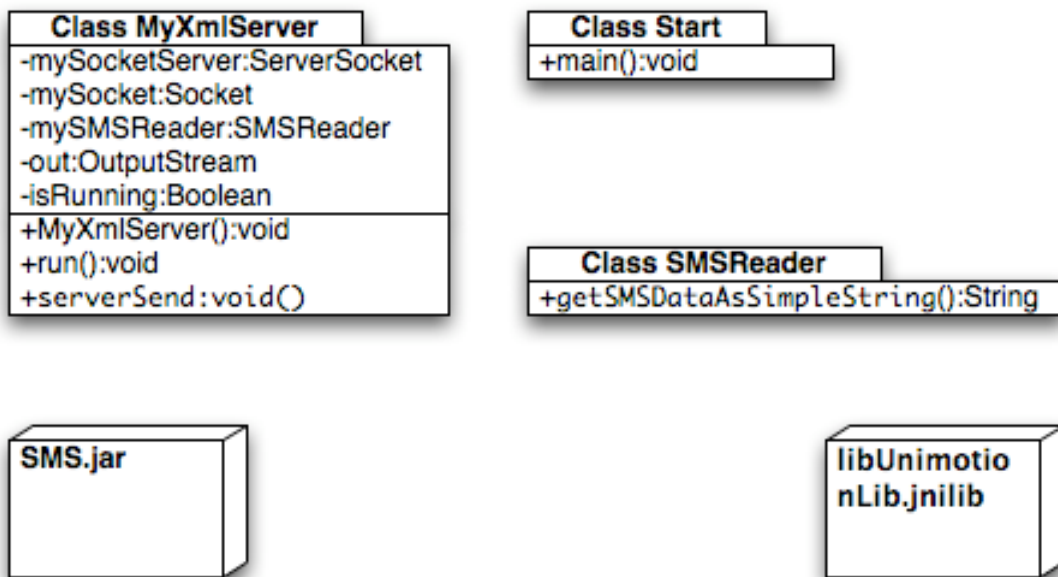


Abbildung 6 Server und Startpunkt der Anwendung