Postmortem - Magneto

Gruppe 3

Mitglieder Christian Blackert (CB), Claus Liebenberger (CL), Stefan Reidlinger (SR)

Spielbeschreibung

Ziel des Spiels Magneto ist es, einen Ball mit Hilfe von Magnetfeldern durch ein Level (2d) zu steuern, um ein Ziel zu erreichen.

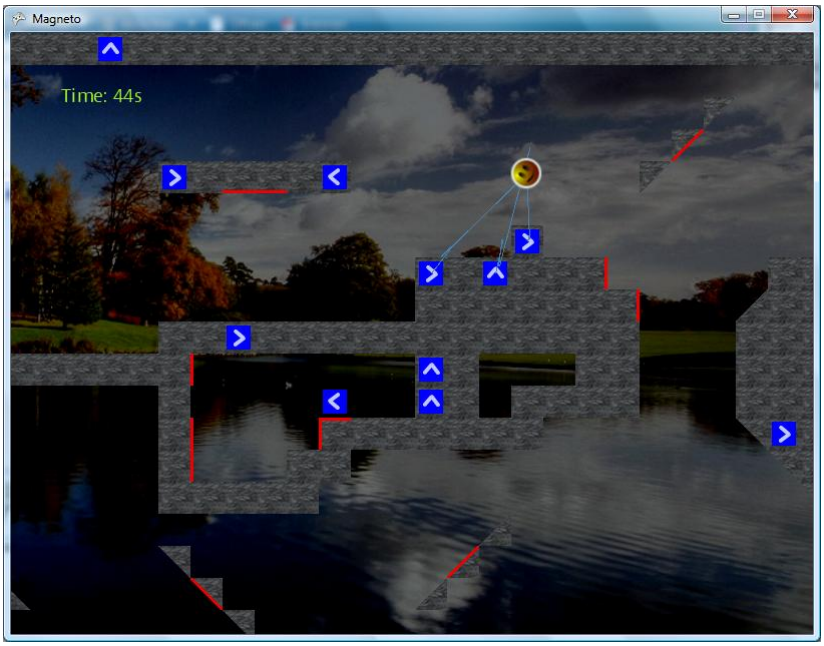

Abbildung 1: Magento Testlevel

Der Aufbau des Levels erfolgt über Tiles, welche am Bildschirm gescrollt werden, und einem statischen Hintergrundbild (mit geringerer Scrollgeschwindigkeit für 3D-Effekt).

Im Level gibt es mehrere Magneten, mit deren Hilfe der Ball gesteuert werden kann. Der Spieler kann über einen Tastendruck nur alle Magnete aktivieren oder deaktivieren.

Jedes Magnetfeld (in einem gewissem Umfeld um den Ball) zieht den Ball an und übt eine zusätzliche Kraft in eine der 4 Richtungen (oben/unten/links/rechts) auf den Ball aus. Zusätzlich wirkt in jedem Level grundsätzlich die Schwerkraft.

An bestimmten Positionen im Level sind "gefährliche" Gegenstände (Spitzen), die den Ball zerstören können. Deshalb muss der Spieler darauf achten, dass der Ball diese Gegenstände nicht berührt. Ist dies der Fall, wird der Ball wieder in die Ausgangsposition zurückversetzt, und der Level beginnt von vorne.

Durch Lösen eines Levels gelangt man zum nächsten, welches bis dahin gesperrt war.

Die Komplexität / der Schwierigkeitsgrad steigt mit jedem Level – die Anfangslevel dienen dazu, die Steuerung des Balls zu erlernen.

Die Zeit, die der Spieler benötigt, um ein Level abzuschließen, bestimmt den Score, der Score, bzw. der letzte freigeschaltene Level wird in einer Konfigurationsdatei gespeichert, sodass er beim nächsten Spielstart wieder verfügbar ist.

Entwicklungsumgebung, Plattform

als Entwicklungsumgebung wird das Microsoft XNA Framework 3.0 mit Programmiersprache C# verwendet.

Zielplattform: Windows XP oder höher mit entsprechenden .NET Framework 3.0.

Grafik / Leveleditor

Die Grafik im Spiel (zumindest im Level selbst) ist nicht sehr aufwändig, und kann mit einigen, wenigen Sprites gezeichnet werden. Nachdem wir als Spielvorlage "Magnetic Joe" bereits zur Verfügung hatten, war die Definition und Erstellung der Grafiken für die einzelnen Objekte im Level nicht besonders schwierig, und wurde schon vor Beginn der Implementierungsarbeiten gemacht. Eine ursprünglich geplante "Verschönerung" der Grafik wurde jedoch aus Zeitmangel in der Endphase nicht mehr durchgeführt.

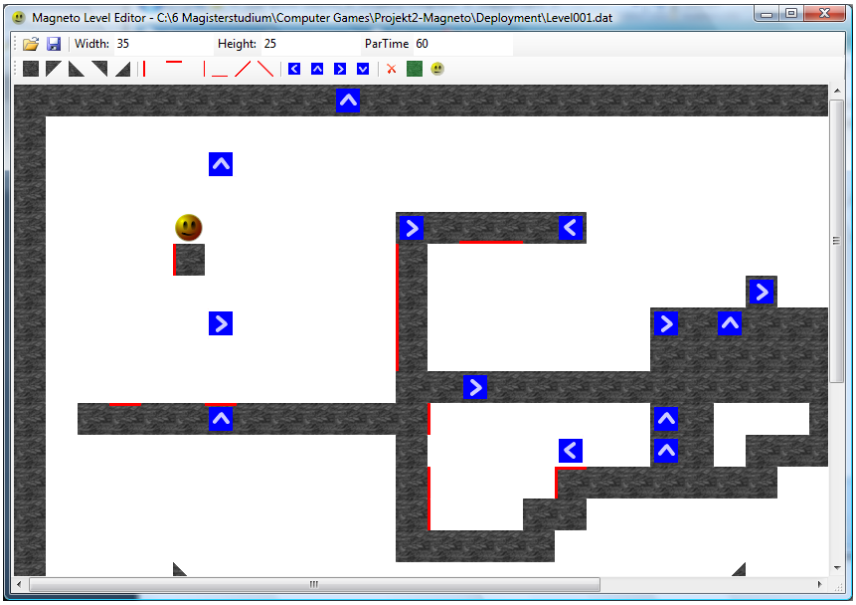

Abbildung 2: Level Editor

Die ursprünglich geplante Definition der Leveldaten wurde von uns jedoch schon ganz zu Beginn geändert, da wir sehr rasch erkannten, dass die manuelle Definition von Levels sehr aufwendig und mühsam würde. Deshalb entschlossen wir uns, noch vor Implementierung des Levelaufbaus im Spiel selbst, einen einfachen, aber für unsere Zwecke ausreichenden Level-Editor zu implementieren, der uns auch während der Testphase für die Spielkontrolle (insbesondere der Physik) gute Dienste erwiesen hat, da man relativ einfach spezielle Levels für Testzwecke erstellen und ändern konnte.

Menüs und Gamestate

Erheblich mehr Aufwand als ursprünglich angenommen, nahm die Implementierung der Spielelemente außerhalb des Levels selbst in Anspruch. XNA stellt hierfür zwar Templates zur Verfügung. Die Einarbeitung in diese Klassen ist jedoch aufwendig, und so entschlossen wir uns, den

Gamestate und die Menüdarstellung, sowie Levelauswahl und Feedback über Erfolg/Misserfolg selbst zu implementieren.

Abbildung 3: Spielemenü

Das Hauptproblem ergab sich hierbei durch das Fehlen einer Abfrage für einen Tastenklick in XNA. Es kann nur der momentane Zustand der Tastatur abgefragt werden. Ein "Klick"-Ereignis, wie z.B. unter Windows Forms wird jedoch nicht angeboten. Dadurch wird die Implementierung von Tastatureingaben unübersichtlich und aufwendig.

Sound

Die Hintergrundmusik und Soundeffekte sind grundsätzlich sehr einfach in XNA zu integrieren, obwohl die Wahl der "richtigen" Musik eine Herausforderung ist, die wir etwas unterschätzten. Letzlich "rippten" wir Musik von einer CD, da wir im Internet nicht fündig wurden. Auch die einzelnen Soundeffekte sind über das WWW relativ schwer aufzutreiben. Es gibt zwar sehr viele Webseiten, bei denen Soundeffekte angeboten werden – aus dem großen Angebot jedoch das richtige herauszufinden, bedurfte einiger Anstrengungen.

XNA kennt 2 Soundkonzepte: a) Hintergrundmusik und b) Soundeffekte. Leider sind die Dateiformate für die einzelnen Konzepte mit .mp3 für Musik und .wav für Effekte fix vorgegeben. Nachdem wir das anfänglich nicht wussten, mussten wir die Implementierung für die endgültige Version ändern, da die ursprünglich von uns verwendeten .wav Dateien für die Hintergrundmusik viel zu groß für eine Auslieferung gewesen wären.

Spiele-Physik

Der ursprüngliche Plan, die Physik des Spiels selbst zu programmieren wurde, nach relativ kurzer Internetrecherche für Collisiondetection zwischen Kreisen und Polygonen sehr rasch aufgegeben. Die Vorstellung der 3D Engine in der Lehrveranstaltung ermutigte uns nach existierenden Physik-Engines

zu suchen. Nachdem wir lediglich 2D Grafik hatten, erschien die Farseer Physics Engine (für die es auch eine Portierung für C# gab) vielversprechend.

Obwohl die Dokumentation der Engine ziemlich unvollständig ist, war die Verwendung der Bibliothek sehr schmerzlos, und abgesehen von ein paar Kleinigkeiten, wie der Positionierung von Objekten nach ihrem Masseschwerpunkt, gab es keinerlei Probleme.

Das Tuning der Parameter für die physikalischen Eigenschaften der einzelnen Objekte benötigte zwar einige Testläufe, funktionierte letztlich jedoch sehr gut.

Zusammenarbeit, Versionsverwaltung

Wir stellten fest, dass die Implementierung des Spiels dann am schnellsten voranschritt, wenn die Teammitglieder im selben Raum programmierten, und neue Funktionen so rasch wie möglich in die Versionsverwaltung aufgenommen wurden.

Die Abhängigkeiten der einzelnen Klassen untereinander war relativ hoch, und Probleme konnten meist nur in Zusammenarbeit von wenigstens zwei Teamkollegen entsprechend rasch gelöst werden. Eine Kooperation über Teleworking (Skype, PC Visit für Desktopsharing) half zwar über Zeiten hinweg, an denen man nicht an einem Ort gemeinsam arbeiten konnte, erzielte jedoch bei weitem nicht die selbe Performance, wie wenn alle Mitglieder zusammen in einem Raum waren. Dies gilt vor allem für die Initialphase der Implementierung – hier sollten alle Teammitglieder persönlich vor Ort sein.

Die doch schon relativ genaue Definition der Programmfunktionalität und die Tatsache, dass wir von vornherein versuchten die Lösung relativ einfach zu halten, half uns von vornherein auf die wesentlichen Aspekte des Spiels zu konzentrieren und machte grobe Programmrestrukturierungen überflüssig.

Es stellte sich jedoch heraus, dass die Schnittstellen zwischen den von den einzelnen Teammitgliedern zu implementierenden Klassen und der grobe Programmablauf auf Code-Ebene gemeinsam definiert werden müssen, damit alle das gleiche Grundverständnis für die Programmarchitektur haben.

Als Versionsverwaltung wurde SVN verwendet, für das es ein Open Source Plugin (Ankh) für Visual Studio 2008 gibt. Auf einem Arbeitsplatz wurde Visual-SVN verwendet. Obwohl mit unterschiedlichen Plugins gearbeitet wurde, verlief die Versionsverwaltung relativ problemlos.

Zeitplanung und Arbeitsaufwand

Mit Ausnahme dieses Postmortems konnten wir die vorgegebenen und ursprünglich geplanten Meilensteine einhalten.

Die Implementierung wurde bedingt durch Abwesenheit von Teammitgliedern erst in den letzten eineinhalb Wochen vor Präsentation gestartet. Mit einem Gesamtaufwand von ca. 120h verlief die Fertigstellung des Spiels in dieser Zeit jedoch relativ problemlos, obwohl auch wir erfahren mussten, wie man sich in "Crunch-Times" fühlt.