Technical Design Document

Team

- Anne-Lena Simon (Team lead)
- Svenja Handreck
- Björn Golla
- Sebastian Rohde

Milestones

Meilenstein I (16.05.2013)

- Erstellen von UML-Diagrammen für das TDD
- Skripten von Events: Einarbeiten
- Design von UI-Elementen
- Dokumentation der Architektur
- Fertigstellung des TDD

Meilenstein II (14.06.2013)

- 3D Modellierung
- Implementierung der Spielobjekte
- Szenenmanager implementieren
- Schokolinsen und Belohnungssystem implementieren
- Protagonisten implementieren
- Implementierung der Kamera
- Umsetzen von Speichern und Laden

Meilenstein III (12.07.2013)

- Testlevel "einfache" Rätseltypen
    - Testlevel Schieberätsel erstellen
    - Testlevel Rutschrätsel erstellen
- Testlevel Höhlenrätsel erstellen
- Testlevel Spurwechsel erstellen
- Prototyp Bosskampf
- Soundeffekte einbinden
- Shader implementieren
- UI-Elemente integrieren

Meilenstein IV (12.08.2013)

- Leveldesign Schieberätsel, Rutschrätsel, Bosskampf
    - Levelumsetzung Schieberätsel, Rutschrätsel, Bosskampf
- Leveldesign Höhlenrätsel, Spurwechsel
    - Levelumsetzung Höhlenrätsel, Bosskampf
- Eventmanager implementieren
    - Events einbauen

- 3D-Modellierung und Animationen

Einreichung des Spiels (01.09.2013)

- Tests, Korrekturen, Feinschliff

Zusatz (optionale Features - kein Termin)

- Hintergrundmusik (Priorität: hoch)
- Bonuselemente, die der Spieler gegen Schokolinsen erwerben kann (Priorität: hoch)
- anpassbare Sprungweite
- Challenge-Modus
- Speichern&Laden mit mehreren Spielständen
- weitere Spielelemente für bessere Level

Das Spiel

Kurze Beschreibung

In "Save the Sweets" spielt der Spieler einen Einwohner von Candyland, der sich auf eine Reise begibt, um seine Welt zu retten. Dabei muss er mehrere Aufgaben (in Form von Rätseln/Labyrinthen) meistern. Auf seiner Reise begegnet er einem weiteren Bewohner Candylands, dem er hilft und der sich im daraufhin anschließt.
Am Ende stellen sie sich gemeinsam der Lakritze entgegen, die versucht, Candyland zu zerstören.

Systemanforderungen

- Windows Betriebssystem mit Unterstützung des xna-Frameworks (empfohlen wird Windows 7)
- Xbox-Controller
- Balance-Board (empfohlen aber optional)

Entwicklung

In diesem Abschnitt wird genauer auf die Tools und Ressourcen eingegangen, die in dem Projekt Verwendung finden.

Verwendete Software

- Microsoft Visual Studio 2010 als IDE
- Tortoise Git zur Versionierung
- Google Drive zur Dokumentation (ggf. stattdessen Projektmanagementsystem angeben)
- Blender zur Erstellung der 3D-Modelle und ihrer Animationen
- Projektmanagement über http://www.clockingit.com/

Externer Code/Bibliotheken

- Nutzung des xna-Frameworks für C#
    - .NET Framework 4 Full wegen IntermediateSerializer zum Speichern des Spiels
    - HLSL für Shader
- Ruminate XNA/MonoGame 4.0 GUI
- Nutzung der DLL, die zur Einbindung der Balance-Boards zur Verfügung steht

Spielmechanik

Architektur

In diesem Abschnitt werden die Module des Spiels und der Zusammenhang zwischen ihnen dargestellt.

Module sind:

- Grafikmanager
    - hier nutzen wir die von XNA gebotenen Methoden
    - Erweiterung durch Shader für Cel-Shading-Look
    - bietet die Methoden, die zum Zeichnen der Szene erforderlich sind
- Audiomanager
    - hier nutzen wir die von XNA gebotenen Methoden
    - bietet die Methoden, die zur Einbindung von Audio-Elementen erforderlich sind
- Interaktionsmanager
    - hier nutzen wir die von XNA gebotenen Methoden
    - Erweiterung zur Anbindung der Balance-Boards erforderlich
    - bietet die Methoden, die zur Kollisionserkennung und zum Erkennen von User-Input erforderlich sind

- Szenenmanager
    - Verwaltung Kamera, EventManager, Areas, Spieler
    - besitzt ein Objekt "UpdateInfo", in dem alle relevanten Daten gespeichert und das in den einzelnen Update-Methoden aktualisiert wird
- Eventmanager
    - Hier werden die Events des Spiels verwaltet und gesteuert. Alle Ereignisse wie Dialoge, Informationen über Schaltergruppen oder Zwischensequenzen werden hier verwaltet. Auch gescriptete Inhalte wie die Flucht des Endgegeners werden hier gespeichert.
- Area
    - Die Welt ist in Bereiche (Areas) aufgeteilt, die jeweils mehrere Level enthalten.
    - Es wird nur für die aktive Area die Update-Methode aufgerufen
- Level
    - Level enthalten die verschiedenen Spielobjekte
    - Kollisionserkennung (mithilfe der Methoden des Interaktionsmanagers) wird immer nur für das aktive Level durchgeführt (ggf. ist hier aus Performanzgründen auch noch weitere Aufteilung nötig)
    - auch "Korridore" sind Level

Grafik

In diesem Abschnitt wird detailliert, welche Anforderungen an die Grafik des Spiels gestellt werden und wie diese erfüllt werden sollen.

- einfache, niedlich wirkende Formen
- bunte Farben, kein Realismus
- Cel-Shading (erfordert eigenen Shader, Priorität eher gering)

Audio

Für die Hintergrundmusik und Soundeffekte sollen nach aktuellem Stand freie Ressourcen verwendet werden. Eingebunden werden diese über xna-eigene Mechanismen.
Soundeffekten sollte eine hohe Priorität gegeben werden, da sie ein wichtiger Feedback-Mechanismus sind. Hintergrundmusik hat dagegen eine niedrige Priorität.

KI

Das Spiel kommt weitestgehend ohne KIs aus. Nur die Lakritze benötigt die Fähigkeit, in gewissem Rahmen mit dem Spieler zu interagieren.
In Phase 1 (Flucht) ist die Bewegung komplett geskriptet. In Phase 2 wird die Lakritze durch eine einfache KI gesteuert.

Lakritze-KI: Wird in der zweiten Phase des Kampfes in einem Geschützturm auf den Spieler schießen. Sie muss dabei den Spieler verfolgen und auf bzw. vor ihn zielen und schießen, also in einem gewissem Maße den Weg des Spielers vorahnen können und dementsprechend versuchen durch Projektile den Weg zu blockieren. Die zweite spielbare Figur wird versuchen können die Lakritze aufzuhalten. Nach erfolgreicher Aktion wird die zweite Spielfigur deshalb von der Lakrize für einige Zeit blockiert, bzw. unbeweglich gemacht.

Multiplayer

Es wird keinen Multiplayer-Modus geben.

Physiksimulation

Für Sprünge wird ggf. eine einfache Physik-Simulation verwendet werden, um halbwegs natürlich wirkende Sprünge unterschiedlicher Länge zuzulassen. Es handelt sich hierbei allerdings um ein Feature mit geringer Priorität.

Spielobjekte und Spiellogik

Alle Spielobjekte basieren auf einer Grundklasse "Object". Diese Klasse beinhaltet bereits die Komponenten "Grafik" und "Interaktion".

Die Welt wird aus einzelnen Bausteinen aufgebaut, es gibt kein unterliegendes Terrain. Die Welt setzt sich aus folgenden Elementen zusammen:

- Plattform
  - einfach
  - Schalter
    - permanent
    - nur aktiv, während Objekt auf Schalter steht
    - Schaltergruppen für Kombinationen
  - bewegend? (Drehung, "durchschieben")
- Hindernis
  - fest
  - überspringbar
  - zerstörbar
  - verschiebbar
- "Waffen"
  - Lakritzstücke im Spurwechsel-Level (fallen vom Himmel)
  - Lakritzprojektile der Maschine (werden geschossen)
- Hauptquartier der Lakritze (Turm)
- Hintergrund (Skybox)

Charaktere gibt es in dem Spiel verhältnismäßig wenige. Es gibt zwei Kategorien von Charakteren, die jeweils auf ihrer eigenen von "Object" abgeleiteten Basisklasse basieren:

- spielbar
  - Protagonist
  - Begleiter
- NPC
  - Lakritze (nutzt Komponente "KI")
    - Variante Flug (bei der Flucht)
    - Variante Maschine (Endkampf)
  - Händler
  - Guide

Weitere Spielobjekte sind:

- Belohnungen
- Schokolinsen
- integrierte UI-Elemente
  - Dialoge
  - Karten
  - Speichern
  - Einkaufen

In den einzelnen Rätselleveln werden Interaktionen durch die einzelnen Objekte selbst behandelt. Dafür wird die Komponente "Interaktion" benötigt, zum Spielende auch die Komponente "KI". Zusätzlich gibt es eine Reihe von gescripteten Events, die die Geschichte voranbringen. Für diese (und für komplexere Beziehungen zwischen verschiedenen Spielobjekten) ist der Event-Manager zuständig.

Datenverwaltung

Das Speichern und Laden der Spieldaten soll mithilfe des IntermediateSerializers geschehen. Dabei werden alle entsprechend markierten Objekteigenschaften und -strukturen in einem XML-Format gespeichert und aus diesem auch wieder ausgelesen.

User Interface

- Menüs (Screen)
  - Neues Spiel
  - Spielstand laden
  - Optionen (Steuerung, Schwierigkeitsgrad, Lautstärke)
  - Belohnungen (Ansehen von Conceptart, Auswahl von Skins, Ansehen von Animationen)
  - Credits
  - Spiel beenden
- Menüs (integriert)
  - Reden
  - Einkaufen
  - Speichern
  - Reisen
- Andere Screens
  - Title Screen
- HUD
  - Dialoge
  - Übersichtskarte(n) (optional)
  - Schokolinsenanzeige
  - aktive Spielfigur
  - Verweis auf das Hauptmenü

Technische Risiken

- Aufgrund von mangelnder Erfahrung
  - können in der Planung grobe Fehler (Fehleinschätzung des Zeitbedarfs, fehlende Arbeitspakete) enthalten sein
  - können einige Elemente uns in der Implementierung große Schwierigkeiten machen
  - kann die bisherige technische Planung (vor allem Architektur, Spielobjekte) sich als nicht zielführend herausstellen und größere Umstrukturierungen erfordern
- Der Zeitaufwand, den wir für die Universität aufbringen müssen (Übungen, Prüfungsvorbereitung, Praktika, Seminare) könnte die aktuelle Einschätzung (weit) übersteigen und damit könnten große Teile der Entwicklung in die prüfungsfreie Zeit verschoben werden müssen. Im schlimmsten Fall wird das Projekt nicht fertig.

Daher gibt es in jedem Meilenstein Pufferzeiten und der letzte Meilenstein sieht nur noch Testen, Korrekturen und Nacharbeiten vor.