GAME DESIGN DOCUMENT

VON JULIA FÖRSTER UND ANNIKA REINHARDT

Unerfahrene Game Designer entwickeln häufig gute Ideen, geben aber oft zu schnell dem Reiz des Programmierens nach. In professionellen Studios muss eine Spielidee verschriftlicht und von verschiedenen Instanzen abgesegnet werden. Damit bei der Fülle an Informationen nicht der Überblick verloren geht, bietet es sich an, ein Game Design Document auszuarbeiteten.[1]

Das Game Design Document ist eine Art Bauanleitung eines Spiels. Es liefert den Entwicklern alle wichtigen Informationen, die für die erste Produktionsphase relevant sind, beschreibt detailliert Spielmechaniken, definiert die Zielgruppe, liefert Verkaufsargumente. Es ist dabei zu großen Teilen stark strukturierend verfasst, beachtet Formeln, Regeln, Werte etc. Des Weiteren beinhaltet es u.a. Umsetzungsvorschriften mit Blick auf Genre, Spielablauf, Charaktere, Level-Design, Sound und Musik sowie User-Interface.

Basierend auf der Spielidee wird zunächst ein Exposé erstellt. Aus diesem entwickeln alle Beteiligten ein Grobkonzept, bei dem Ideen, Features, Grafikstil etc. ausformuliert werden. Es folgt ein Prozess der Verfeinerung dieses Konzepts, an dessen Ende schließlich das Game Design Document als Ergebnis steht.

> The Game Design Document is based on the game designer's vision, research, and license or storyline, plus additional input by other team members during the preliminary design phase.[2]

Das Game Design Document setzt also alle Features in einen Zusammenhang, so dass alle am Entwicklungsprozess beteiligten Personen eine (mehr oder weniger) genaue Vorstellung des fertigen Spiels haben: »Der Vorteil eines Konzeptes dürfte ja allen klar sein: man hat schwarz auf weiß, wo es hingeht, was alles zu tun ist, hat eine Grundlage für die Planung«,[3] so Daniel Dumont.

Programmierer können dem Game Design Document beispielsweise entnehmen, welche Anforderungen das Spiel an die Game-Engine stellt und welche Daten sie anlegen müssen. Genauso müssen

> Grafiker […] nach der Lektüre des Dokuments in der Lage sein, sich die Welt, um die es geht, vorzustellen und Konzeptgrafiken anzufertigen (die dann wiederum Teil des Designdokuments werden). Musiker

1 Vgl. Ryan: »The Anatomy of a Design Document«.

2 Pedersen: Game Design Foundations, S. 356-357.

3 Dumont: »Creating Game Design Documents«, S. 1.

> etwa sollten beim Lesen des Dokumentes die Stimmung des Spieles einfangen können, um passende Musik dafür zu komponieren.[4]

Die Designer überarbeiten und ergänzen das Game Design Document stetig; ab einem bestimmten Zeitpunkt werden jedoch keine neuen Ideen mehr aufgenommen, sondern die bestehenden Komponenten nur noch verfeinert. Das Game Design Document sollte komplett ausgearbeitet sein, bevor der Prozess des Programmierens und Umsetzens der Spielidee beginnt.

> Durch die Regel: Nur was im Game Design Document steht, darf umgesetzt werden, erhält der Gamedesigner volle Kontrolle über das Gamedesign – egal von wem die Idee ursprünglich stammt.[5]

Am Ende eines Projektes kann auf Basis des Game Design Document eine Checkliste für die Qualitätssicherung erstellt werden. So kann überprüft werden, ob alle geplanten Features eines Spiels implementiert wurden und wie vorgesehen funktionieren. Jedes Game Design Document wird individuell auf ein bestimmtes Spiel gemünzt. Es gibt kein einheitliches Format oder feste Kategorien. Größere Studios geben ihren Designern zwar Vorgaben, jedoch unterscheiden sich diese von Studio zu Studio. Wichtig bei jedem Design Document ist die Übersichtlichtkeit. Alle Beteiligten müssen sofort erkennen, welche Textabschnitte für sie wichtig sind und welche sie nur überfliegen müssen.

> In writing the text of your document, you will want to break it up with lots of titels, headings, subheadings, and so forth. This will make it easier for readers to skim over the document and zoom in on the information they are seeking. Breaking your information into lists, either numbered or bullet, wherever possible will further allow readers to easily realize what different attributes a given part of the game will need to include.[6]

Auf den folgenden Seiten findet sich ein von Mike Dally (ehemaliger Game Desginer bei DMA) veröffentliches[7] Game Design Document zu einem Spiel mit dem schlichten Arbeitstitel »Race'n'Chase Game« – heute bekannt als GRAND THEFT AUTO (DMA Design, 1997).

4 Steinke: Spieleprogrammierung, S. 121.

5 Dumont: »Creating Game Design Documents«, S. 1.

6 Rouse: Game Design, S. 357.

7 Quelle: http://www.flickr.com/photos/mikedailly/sets/72157602022230830/with/5548285262/, 15.09.2011.

LITERATURVERZEICHNIS

Dumont, Daniel: »Creating Game Design Documents«, 2006, http://igda.dimajix.net/fileadmin/ressources/report-06/handouts/05_Summary_Dumont.pdf, 15.09.2011.

Pedersen, Robert E.: Game Design Foundations, Sudbury 2009.

Rouse, Richard: Game Design: Theory and Practice, Sudbury 2005.

Ryan, Tim: »The Anatomy of a Design Document, Part 1: Documentation Guidelines for the Game Concept and Proposal«, 1999, http://www.gamasutra.com/view/feature/3384/the_anatomy_of_a_design_document_.php, 15.09.2011.

Sorour, Sven: »Spieleentwicklung: Der Weg zum eigenen professionellen Design-Dokument«, 2009, http://www.game-inventor.de/design-dokument-2/, 15.09.2011.

Steinke, Lennart: Spieleprogrammierung, Heidelberg 2008.

Race'n'Chase
Game Design

Version 1.05
March 22, 1995

Revision List

Version	Author	Date	Comments
1.00	K.R. Hamilton	January 25, 1995	Initial version
1.01	K.R. Hamilton	January 30, 1995	Revision after meeting of 30/01/95
1.02	K.R. Hamilton	February 6, 1995	Revision after meeting of 06/02/95
1.03	K.R. Hamilton	February 20, 1995	Revision after meeting of 20/02/95
1.04	K.R. Hamilton	March 6, 1995	Revision after meeting of 06/03/95
1.05	K.R. Hamilton	March 22, 1995	Revision after meeting of 21/03/95

Table of Contents

1. Introduction

This document specifies a design for the gameplay of a game with the provisional title "Race'n'Chase". It is based on elements discussed in various meetings held since 23rd January 1995 and involving Dave Jones, Mike Dailly, Robert Parsons, Stewart Graham, Steve Hammond, Chink, Oz, Keith Hamilton and David Kivlin.

1.1 Scope

This document is intended to be read by programmers, artists and producers involved in the design, implementation and testing of Race'n'Chase.

1.2 Type Conventions

Things which have been discussed in a meeting are presented in this document using Times font, like this.
Things which have not been officially agreed on but which are suggested by the author are presented in Chicago font, like this.

2. References

[1] **CityScape Data Structure**
Version 3.10 - March 21, 1995 - DMA Design Ltd

[2] **Pedestrians in Race'n'Chase**
Version 1.00 - March 14, 1995 - DMA Design Ltd

[3] **Vehicles in Race'n'Chase**
Version 1.00 - March 14, 1995 - DMA Design Ltd

[4] **Traffic Control in Race'n'Chase**
Version 1.00 - March 15, 1995 - DMA Design Ltd

[5] **Screen Display in Race'n'Chase**
Version 1.00 - March 16, 1995 - DMA Design Ltd

[6] **Comms in Race'n'Chase**
Version 1.00 - March 17, 1995 - DMA Design Ltd

3. Target System

Race'n'Chase will be produced for the following platforms : PC DOS, PC Windows 95, Playstation, Saturn and Ultra 64. This document is primarily concerned with the PC versions.

3.1 DOS

The DOS version will use a DOS extender so that it can use the 32-bit flat memory model. It will require 4MB of RAM. This limit may have to be raised to 8MB to accomodate all of the graphics which will be required.
The game will run in either SVGA mode 101h (640*480) or VGA mode 13h (320x200). Both modes use 256 colours from an 18-bit palette. A very fast processor (e.g. Pentium) will be recommended for the SVGA mode.

3.2 Windows 95

The 32-bit Windows version will require 8MB of RAM. It will use 8-bit or 16-bit graphics, depending on the card fitted.

Both PC versions of the game will be supplied on one CD-ROM. There will be no floppy version.

4. Development System

4.1 Software

Race'n'Chase will use the overhead perspective engine developed by Mike Dailly.
The DOS version will be developed using Watcom C/C++ v10, Microsoft MASM 6.1 and Rational Systems DOS extender (DOS4GW) v 1.97.
The Windows 95 version will be developed using Visual C++ v2.0.

5. Specification

5.1 Concept

The aim of Race'n'Chase is to produce a fun, addictive and fast multi-player car racing and crashing game which uses a novel graphics method.

5.2 Story

5.2.1 Setting

The game will be set in a present-day world.

5.3 Game Structure

There will 3 cities with a different graphic style for each city (e.g. New York, Venice, Miami). There will be many different missions to be played in each city. This is so that players can get to know the routes through a particular city.
In each game type, it will be possible to progress to different cities only when certain goals have been attained.

5.4 Players

The PC game will be playable by multiple players across a network or by one player at a standalone machine. Console versions will allow two players at one machine. This facility may be added to PCs.

5.5 Action

Players will be able to drive cars and possibly other vehicles such as boats, helicopters or lorries.
Cars can be stolen, raced, collided, crashed (ramraiding ?) and have to be navigated about a large map.
It will also be possible for players to get out of their car and steal another one. This will mean controlling a vulnerable pedestrian for a short time. Trying to steal a car may result in an alarm being set off which will, of course, attract the police.

5.6 Objective

The objective of the game will vary depending on the game type. The player can choose to play one of 4 different games:

5.6.1 Cannonball Run

This a a straight race across the city where the winner is the first player to drive from 'A" to 'B', taking any route. Best times could be saved. Pole positions could be set depending on each players best time round a practice route.
Computer-controlled cars could be included in the race. Progressing to the next level would require finishing ahead of the computer cars.

5.6.2 Demolition Derby

The players can drive anywhere in the city. The aim is to cause damage to the other cars by crashing into them. The winner is the driver who survives the longest.

Alternatively, players whose cars get wrecked could get a new car back at the start, so the winner is the player who wrecks the most enemy cars.

5.6.3 Bank Robbery (Robber)

The player drives a getaway car and must escape from the police by reaching a particular location (e.g. cross safe line or get to safe house). There will be many different missions available - with varying start point, end point, police presence, etc.

When enough crimes have been completed, the player can move on to a different city.

The robber's game is up when he gets killed or is captured by the police.

5.6.4 Bank Robbery (Cop)

The player controls a police car and must stop a getaway car from escaping. Other police cars are controlled by the computer or by other network players.

5.7 Graphics

5.7.1 Landscape

The landscape will be viewed from directly above, with perspective. Different scenes are possible, e.g. urban, countryside, seaside, etc. The landscape will be built from 64x64x64 pixel blocks, which are drawn with a texture map for each visible face.

Some faces will be animated. Some faces will be transparent.

The screen will scroll left/right/up/down as the player's vehicle moves so that is is kept approximately centred. The screen will not tilt or rotate, but it will be possible to zoom in and out.

The screen will zoom to different levels automatically depending on the action. For example, it could zoom out as the car drives faster, or zoom in to show a crash in more detail. At maximum zoom in, individual people will be clearly visible. The maximum zoom out which is practical will depend on the speed of the PC.

5.7.1.1 Reduced Detail

A reduced detail mode will be included for PCs which are not fast enough to display the whole perspective view. In this mode, the sides of building will not be drawn and the tops will be drawn on the ground. The will result in a landscape looking something like the original Sim City.

5.7.1.2 Optmized Display

The perspective landscape must be drawn as fast as possible. Looping versus repetition for line draws is being investigated.

5.7.2 Objects

Objects (e.g. cars) will be drawn using some scaled & rotated sprites (see below for more details). Each will be around 64x64 pixels. Cars will always be seen from above.

All object graphics will be based on rendered models.

There will be around 20 different cars per style.

5.7.3 Screen Display

5.7.3.1 Dashboard

Overlaid on top of the landscape view will be the instruments of the current car. These will include:

- speedometer
- rev counter
- damage meter

5.7.3.2 Popups

Miniature popup windows could appear occasionally to show animations of what is happening, e.g. arrest being made, car being wrecked.

5.7.4 Weather

Different weather effects will be investigated , for example :

- snow (swirling effect white pixels)
- thunder & lightning (flash screen, darken palette)

5.8 Data Storage

5.8.1 Car Sprites

As the light source is directly overhead, rotation can be done in software. This means storing only 3 frames per car (the up/down rotation).
Deltas will be drawn onto the car sprites to show additional detail, such as brake lights, police lights, damage.

5.8.1.1 Compressed World

Some form of compression will be used to store the world data. This should aim to produce a world which takes at least 2 minutes to drive across but which can be held entirely in memory.
This may mean around 256x256x6 blocks.
Buffer loading could still be used, but only for major changes, e.g. crossing into a different state.
Compression ideas :

- store 2-byte pointer for each block, instead of 5 faces & type
- store 2-byte pointer for each column (no bridges then)
- store blocks run-length encoded upwards

These compression methods will be investigated.

5.8.2 Faces

Landscape face graphics will require:
Bytes per face : 64x64
Faces per style : 255
Total : 1,044,480 bytes

5.8.3 Code

Code space will amount to 1MB.

5.8.4 Sound

Space for sound (samples, etc.) will amount to 1MB.

6. Gameplay

6.1 World

The playing world will be very, very large - multiple screens.

There will be a number of clear landmarks to ease navigation.

A large printed map will be supplied as part of the package. It will be necessary to refer to this during gameplay.

Usage of the pause key may be restricted so that players cannot keep pausing the game to look at the map. Instead, they must park somewhere to stop and look at the map.

6.2 Landscape

The landscape will consist of:

- roads (small roads and freeways)
- pavements
- buildings
- water hazards
- bridges

The landscape is not fixed, and can be altered by player actions.

The landscape will include a number of levels so that, e.g., a road could be on a bridge across another road. This means that slopes will be necessary to get from one level to another.

The landscape will be, in a city, highly populated : there will be lots of incidental things to see like traffic, pedestrians, etc.

6.2.1 Roads

Roads will be constructed using a map editor from a number of fixed pieces, e.g. corners, junctions, straights, etc. Lanes will be 1 block wide.

6.2.2 Pavement

Pedestrians will normally walk on pavement blocks. Cars can drive on road or pavement. Pavements will be 1 block wide.

6.2.3 Buildings

Buildings will be constructed from cube-shaped blocks but can be any shape. It will be possible for cars to cause damage to buildings when they crash - e.g. plate glass window on side of building is seen to smash.

6.3 Ground Type

Types of ground will include:

- road
- pavement
- water
- building

6.4 Object Types

Objects which can appear include:

- cars

- road blocks
- road signs
- traffic lights
- pedestrians
- inanimate objects, e.g. bins

6.4.1 Cars

There will be player controlled cars, intelligent other cars, and simple drone cars.

It will be possible for cars to sustain damage and continue. The damage will be shown by the car slowing down, wobbling, pulling to one side, or emitting smoke. It could also be shown by damage deltas drawn onto the car sprite.

If a player-controlled car has a serious crash, it will blow up after a short time. Hence, the player must get out of the car and find another one.

Cars will not run out of fuel.

6.4.1.1 Car Intelligence

Intelligent cars will be able to navigate themselves about the city, plotting the shortest route to another car or to a particular location.

6.4.2 Lorries

Lorries will be treated as a seperate cab and trailer, where the trailer just has to always follow the cab.

6.4.3 Pedestrians

Pedestrians will be wandering about all of the time. They can be run over by cars. They will tend to walk on pavements.

Types of pedestrians could include :

- schoolchildren & lollipop lady
- dogs

6.5 Police

6.5.1 Communication

The police will communicate with each other by means of radio messages which the player will hear as sampled speech complete with radio crackles.

Getaway drivers will have a scanner which is tuned into these police broadcasts. Messages will be of the form "x seen on y street" and will be sent to all other police cars when the robbers are spotted.

The printed map will have to be used to see where the street in question is.

6.5.2 Guns

The police will be able to get out of their cars and shoot at the robbers.

6.6 Control

The game will be controlled by mouse or keyboard.

6.6.1 Direct Control

When using direct control (i.e. when driving one car), the controls will be:

- accelerate
- brake
- turn left

- turn right
- change gear - forward / reverse
- sound horn
- get in/out car

The radio-control car method will be used, i.e. directions are always relative to the car.
The steering will auto-centre. That is, it will tend to turn back towards straight ahead. The amount of turn will increase as the steering key is pressed.
A handling method must be developed which will permit the player to perform stunts with the car such as handbrake turns, spinning wheels, etc.

6.6.2 Hardware

The thrustmaster steering wheel, and possible other devices, will be supported.

6.6.3 Indirect Control

An indirect control method (using mouse to co-ordinate various intelligent cars) can be added to the game at a later stage if necessary.

7. Front End

7.1 Intro

There will be a pre-drawn/rendered animated introduction to the game.

7.2 Menus

The game will use a simple menu system, as in Doom, for selecting options.

8. Development Tools

8.1 Editor

The editor used for Race'n'Chase will produce a 3D array which can used by both the perspective and the isometric engines, so that it can also be used for other games.
It will consist of a grid editor which is used to place blocks on a grid, with a seperate grid for each level. The editor will allow any block to be placed at any level. Each block can be assigned texture maps for up to 5 faces (top and 4 sides). A standard data format will be used to represent this.

9. Team

Project Manager	: Keith H
PC Programming	: Keith H, Robert P
PS-X Programming	: Cameron R
Saturn Programming	: tba
U-64 Programming	: tba
Programming	: David K + 2 others
Art	: Chink + 3 others
Design	: Stewart G
Producer	: Dave

10. Time

Official Start Date	:	April 4, 1995
Complete Game Design	:	May 31, 1995
Milestone 1 - Engine	:	July 3, 1995
Milestone 2 - Look &Feel	:	October 2, 1995
Milestone 3 - 1st Play	:	January 3, 1996
Milestone 4 - Alpha	:	April 1, 1996
End of Project	:	July 1, 1996