

# C-Projekt Breakout

von Albrik Dürsteler und Erik Amgwerd

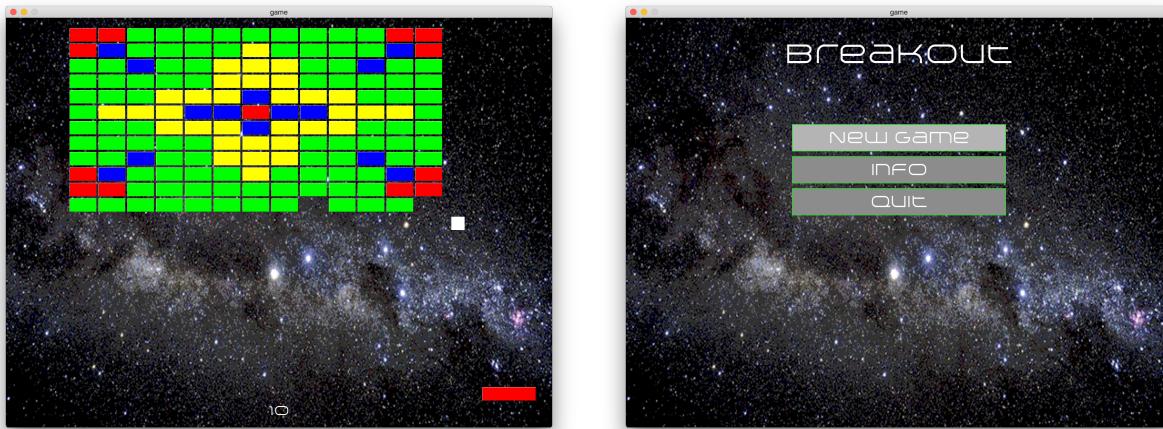
2.1.2017

## Inhaltsverzeichnis

<b>1 Beschreibung</b>	<b>3</b>
<b>2 Geschichte von Breakout</b>	<b>3</b>
<b>3 Spielablauf und Ziel</b>	<b>3</b>
3.1 Use Case . . . . .	4
3.2 Programmablauf . . . . .	4
3.2.1 Programmbeschreibung . . . . .	4
3.2.2 Ablaufdiagramm . . . . .	5
<b>4 Projekt- und Codestruktur</b>	<b>5</b>
4.1 Filestruktur . . . . .	6
4.2 Codeaufbau . . . . .	6
4.3 benötigte Allegro-Komponenten . . . . .	7
4.4 Aufbau des Level-Files . . . . .	7
4.5 Builds und Platform . . . . .	7
4.6 Retina Einstellung . . . . .	8
<b>5 Schluss</b>	<b>8</b>
5.1 Herausforderung des Projekts . . . . .	8
5.2 Schlusswort . . . . .	8

## 1 Beschreibung

Diese Dokumentation beinhaltet die Implementation und Definition des C-Programm Projektes „Breakout“, erarbeitet von Albrik Dürsteler und Erik Amgwerd.



## 2 Geschichte von Breakout

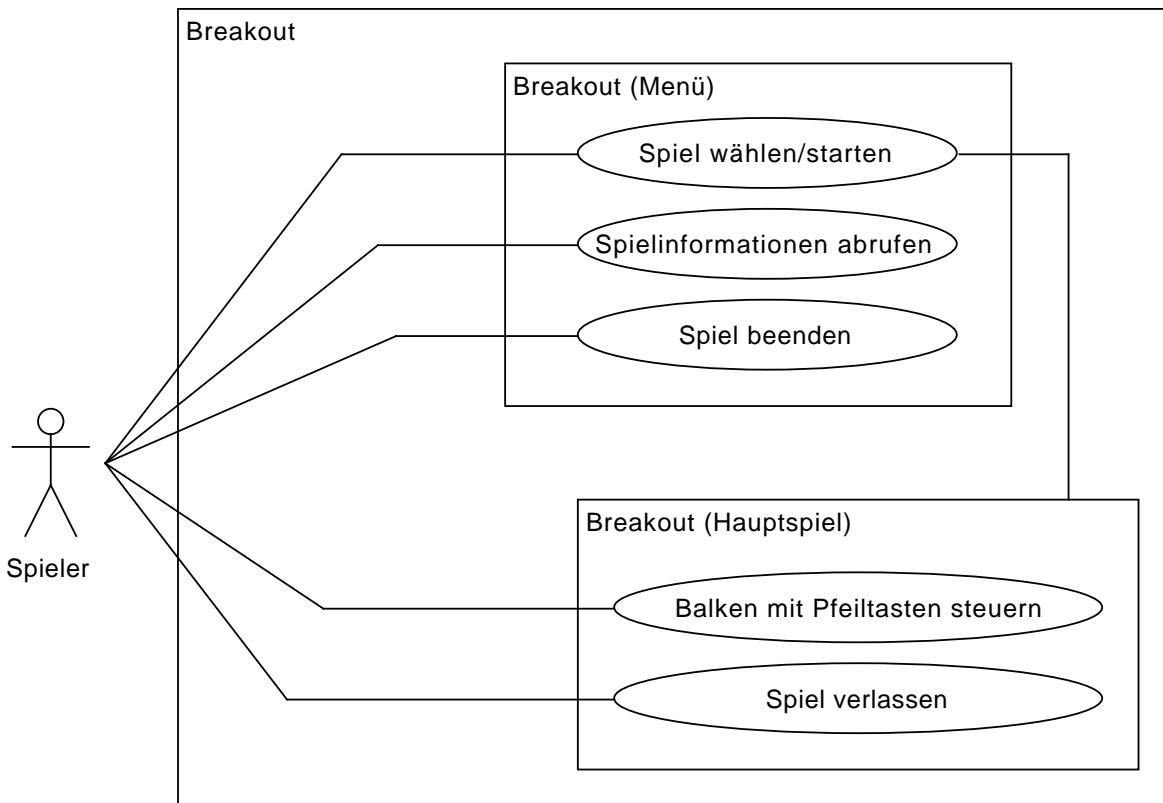
Breakout (oder Little Brick out) ist ein Programm (Computerspiel) das in den 70er Jahren von Atari entwickelt wurde.



## 3 Spielablauf und Ziel

Die Aufgabe ist es einen Ball so abzufeuern, dass man alle Mauersteine im Oberen Feld zerstört, ohne dass der Ball einmal herunter fällt. Speziell zu beachten gibt es das der Ball nur am Anfang abgefeuert wird und man dann, wie bei PingPong den Ball anhand einer Bank wieder zurückschleudern muss. Je mehr Mauersteine zerstört werden, umso mehr Punkte gibt es. Werden alle Mauersteine zerstört, fängt das Spiel wieder von vorne an. So ist es möglich immer mehr Punkte zu erzielen und den Highscore zu knacken.

### 3.1 Use Case

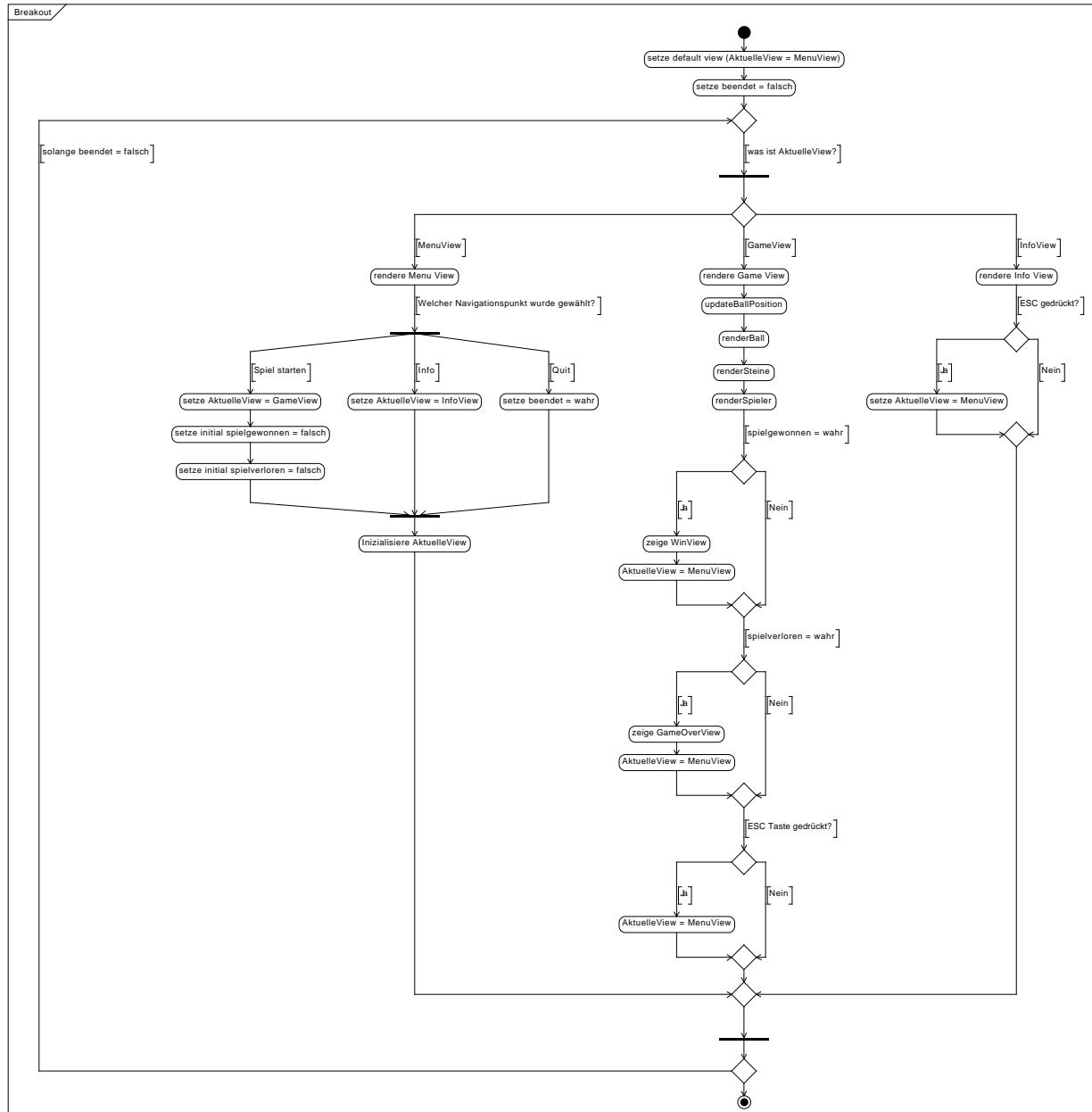


### 3.2 Programmablauf

#### 3.2.1 Programmbeschreibung

Der Benutzer startet das Spiel und steigt im Hauptmenü ein. Er hat die Möglichkeit "New Game", "Info" oder "Quit" mit den Pfeiltasten zu wählen und mit der Eingabe zu bestätigen. Wählt der Benutzer "New Game", startet das Spiel sofort. Es werden farbige abbaubare Blöcke, einen mit Pfeiltasten steuerbaren Balken sowie ein Spielball angezeigt. Wenn der Ball unterhalb des Bildschirms verschwindet, hat der Spieler das Spiel verloren. Game Over wird angezeigt. Baut der Spieler alle Blöcke ab, so hat er gewonnen. Dies wird ebenfalls angezeigt. Der Benutzer kommt auf die Menüansicht zurück, wenn der das Spiel beendet oder ESC gedrückt hat. Wählt der Benutzer "Info", wird ein Programminfo-Bildschirm angezeigt. Mit ESC gelangt der Spieler zurück ins Menü. Bei "Quit" wird das Spiel beendet.

### 3.2.2 Ablaufdiagramm



## 4 Projekt- und Codestruktur

Das Programm wurde komplett in der Programmiersprache C geschrieben. Die Game Library Allegro 5.1 wurde für die grafische und audiovisuelle Umsetzung, sowie für das Keyboard- und Event-Handling verwendet. Es muss Allegro 5.1 auf dem System installiert sein.

## 4.1 Filestruktur

- Arkitech\_Light.ttf (einzige verwendete Schrift)
- audio.ogg (einiger verwendetes Audio-Sample)
- background.png (einzig verwendete Grafik)
- build\_osx (build script für mac os x)
- constants.h (hier werden sämtliche Konstanten definiert)
- main.c (Hauptprogramm)
- map1.txt
- map2.txt (verwendete Karte)
- scenecontroller.c (Ansichten Kontroller/Switcher)
- scenecontroller.h
- scenes/gamescene.c (Hauptspiel)
- scenes/gamescene.h
- scenes/infoscene.c (Infobildschirm)
- scenes/infoscene.h
- scenes/mainscene.c (Hauptmenü)
- scenes/mainscene.h

## 4.2 Codeaufbau

- Der Code ist modular aufgebaut.
- Jede Scene befindet sich in einem separaten File.
- Jedes C file hat ein Header File mit deklarierten Prototypen und Variablen.
- Konstanten sind in einem separaten File ausgelagert.
- libraryimports sind im jeweiligen Headerfile definiert.
- es gibt keine globalen Variablen.
- Levels sind in .txt Dateien definiert.
- Jede View wird auf- und abgebaut durch einen SceneController.
- Jeder Funktionsname hat den Prefix des aktuellen Filenamens.
- Es gibt keine dynamische Speicherallozierung in diesem Programm.
- Der Code ist kommentiert
- Die Funktionsnamen sind grösstenteils selbsterklärend definiert.
- array, struct, int, float, char, string (, enum werden verwendet.

### 4.3 benötigte Allegro-Komponenten

```
#include <stdio.h>
#include <allegro5/allegro.h>
#include <allegro5/allegro_primitives.h>
#include <allegro5/allegro_font.h>
#include <allegro5/allegro_image.h>
#include <allegro5/allegro_ttf.h>
#include <allegro5/allegro_audio.h>
#include <allegro5/allegro_acodec.h>
```

### 4.4 Aufbau des Level-Files

#### Farbdefinitionen

char	Farbe	Punkte
r	Rot	5
g	Grün	5
b	Blau	5
y	Gelb	15

#### map2.txt Datei

```
rrggggggggggrr
rbgggggyggggbr
ggbgggyyyyggbgg
gggggyyyygggggg
gggyyybyyygggg
gyyybbrrbbyyyg
gggyyybyyygggg
gggggyyyygggggg
ggbgggyyyyggbgg
rbgggggyggggbr
rrggggggggggrr
gggggggggggggg
```

### 4.5 Builds und Platform

Das Programm wurde auf einem Mac OS X Sierra entwickelt und getestet. Ein statisches binary des Programms steht zur Verfügung.

## 4.6 Retina Einstellung

Es ist zu beachten, dass der Build für Retina optimiert ist. Um einen Nicht-Retina-Build zu erstellen, muss im `src/constants.h` der Skalierungsfaktor von 2 auf 1 geändert werden

```
#define SCREEN_RATIO 2
```

# 5 Schluss

## 5.1 Herausforderung des Projekts

- Ansichtenwechsel zwischen den verschiedenen Views
- Leveldatei strukturiert einlesen
- Keyboardevents mit verschiedenen Views
- mehrseitige Ballcollision mit Steinen (leider noch nicht so gut gelöst)
- statisches Build erstellen mit Allegro-Komponenten

## 5.2 Schlusswort

Die Entwicklung dieses Computerprogrammes bereitete uns viel Freude. Nicht oft hat man die Möglichkeit so etwas zu entwickeln. Obwohl die C-Erfahrung sehr eingerostet war, war es uns relativ schnell gelungen, ein Spiel auf Basis einer Dritten-Library zu realisieren.