# Programmierprojekt - “BreakING”

von Stefan Neumann, Andreas Neumann und Michael Chen

# Inhaltsverzeichnis



# Programmbeschreibung

Grundlegend handelt es sich bei “BreakING” um ein Spiel, nach dem Prinzip des klassischen Spiels “Breakout”. Man bewegt im unteren Teil des Bildschirms eine kleine Plattform und muss mit dieser einen hoch- und runter fliegenden Ball entsprechend treffen, sodass er nicht den unteren Bildschirmrand berührt.
Ziel des Spiels ist es, Blöcke die sich im oberen Teil des Bildschirms befinden zu treffen, somit zu zerstören und “auszubrechen”.

Unser Programm beinhaltet neben dem Spiel an sich noch andere Ergänzungen, die sich in verschiedene Programmteile auslagern. Dazu gehört ein Menü zur Individualisierung der

Plattform und des Balls mit verschiedenen Grafiken, ein Menü für die Auswahl von verschiedenen Leveln (mit entsprechend höheren Schwierigkeitsstufen) und ein Menü für die Feineinstellung bestimmter Parameter (Ballgröße, Ballgeschwindigkeit,...) sodass das Grundspiel selber nochmal individuell verändert werden kann.

# Leistungsmerkmale

## 1. Übersichtliche Menüführung:

Das Benutzer-Interface soll einfach und schnell verständlich aufgebaut sein, sodass der Benutzer kein Probleme hat zwischen Menüs (intern Programmteilen) zu wechseln.

## 2. Guter und abgerundeter Spielfluss:

Das Spiel als Hauptteil, soll flüssig und gut laufen. Dazu gehört ein flüssiger Spielverlauf, eine gute und präzise Steuerung und ein forderndes, aber auch realistisches Tempo.

## 3. Highscores:

Während des Spiels wird ein gewisser Punktestand errechnet und dieser soll lokal mit einer individuellen Signatur gespeichert werden.

## 4. Levels:

Es gibt verschiedene Levels mit steigender Schwierigkeit, die mit der Zeit freigeschaltet werden können.

## 5. Dynamische Grafikveränderung:

Der Benutzer soll die Möglichkeit haben, gewisse Grafiken für den Ball oder die Plattform freizuschalten und diese in einem Untermenü zu verändern.

## 6. Einstellbare Spielparameter:

Der Benutzer kann in einem weiteren Untermenü verschiedene (eigentlich nur interne) Parameter selber feinjustieren. So kann er sich das komplette Spiel selber etwas neugestalten.

## 7. Möglichst wenig Fehler:

Das komplette Programm soll auf verschiedene Fehler getestet sein, sodass es nicht zu unvorhergesehenen Störungen kommen kann.

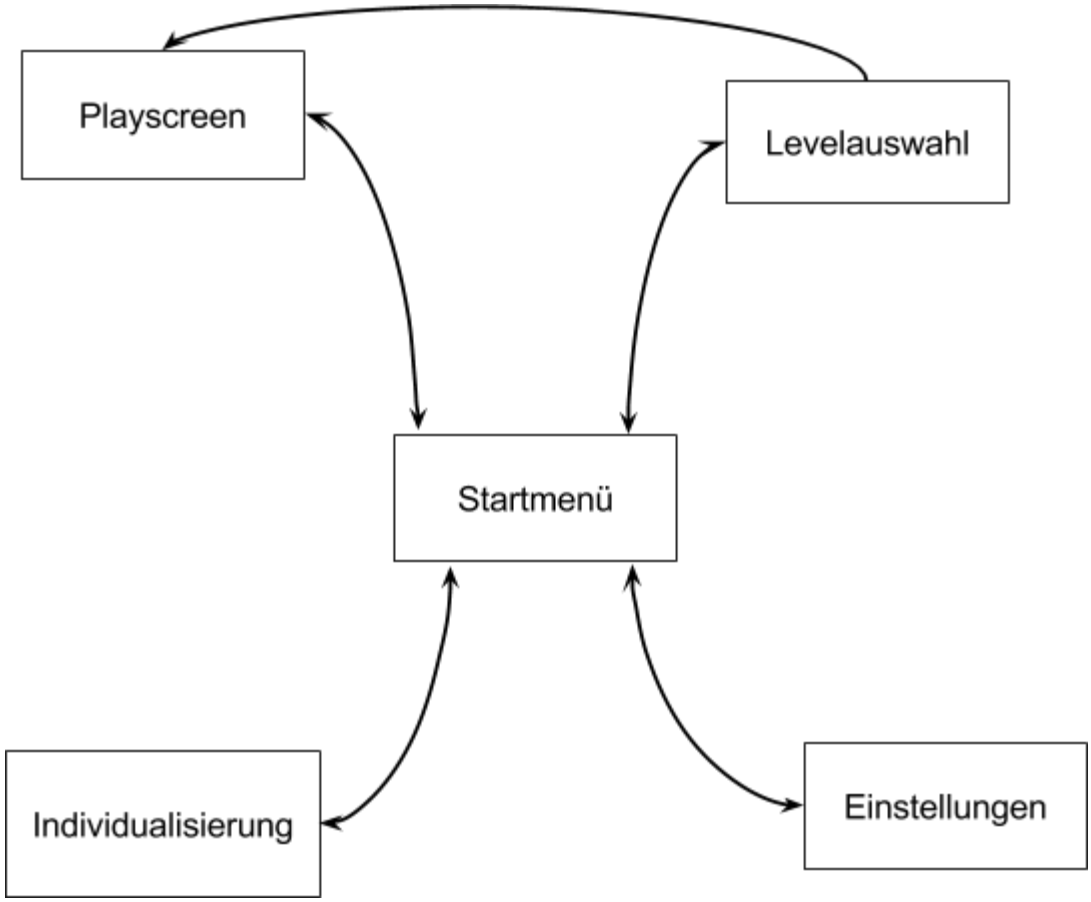
## 8. Töne:

Das Programm soll unterlegt sein mit Musik und Soundeffekten.

## 9. (nicht fest):

Geplant ist weiterhin eine Online-Highscore-System bei dem der Benutzer seinen eigenen Highscore hochladen kann und mit anderen vergleichen kann.

# Programmübersicht




# Programmstruktur

1. **Initialisierung**
    a. **SDL Initialisieren**
        i. **SDL_Windows erstellen**
        ii. **SDL_Renderer erstellen**
    b. **SDL Plugins Initialisieren (SDL_image, SDL_ttf, ...)**
    c. **Spieldateien laden**
        i. **Speicher für Arrays allozieren**
        ii. **Grafiken laden**
        iii. **Audiodateien laden**
        iv. **Schriftartdateien laden**
2. **Gameloop**

a. **Updates**
    i. **SDL Events abfragen**
        1. **Events verarbeiten (Mausklicks und Tastendrücke)**
            a. **Wenn im Menü**
                i. **Prüfen, ob Button gedrückt wurde**
            b. **Wenn im Spiel**
                i. **Prüfen, ob Spieler mit Paddle interagiert**
        2. **Wenn Alt-F4 Gameloop beenden**
    ii. **Wenn im Spiel:**
        1. **Spielobjekte aktualisieren**
            a. **Positionen aktualisieren**
            b. **Kollisionen abfragen:**
                i. **Objektgeschwindigkeit anpassen**
                ii. **Gameover abfragen**
                iii. **Spielstatus erneuern**
                iv. **Ggf. Audio abspielen**
        2. **Spielvariablen anpassen (Lives, Score, ...)**
b. **Rendering**
    i. **Wenn im Menü**
        1. **Textgrafiken rendern**
    ii. **Wenn im Spiel**
        1. **HUD rendern (Lives, Score, ...)**
        2. **Alle Objekte rendern**
c. **Wiederhole 2. Gameloop**

3. **De-Initialisierung**
    a. **Spieldateien entladen**
        i. **Grafiken zerstören**
        ii. **Audiodateien zerstören**
        iii. **Schriftartdateien zerstören**
    b. **SDL Plugins de-initialisieren (SDL_image, SDL_ttf, ...)**
    c. **SDL de-initialisieren**
        i. **SDL_Renderer zerstören**
        ii. **SDL_Windows zerstören**
        iii. **Speicher befreien (Arrays)**

# Zentrale Funktionen

## Top-Level Funktionen

```
int main(int argc, char * args[]); // Entry Point
void INITIALIZE(); // Siehe Struktur 1.
void UPDATE(); // Siehe Struktur 2.a.
void DRAWFRAME(); // Siehe Struktur 2.b.
void QUIT() // Siehe Struktur 3.
```

## Menüfunktionen

```
bool BUTTON_IsClicked(Button * btn);
// Wird im Eventhandler aufgerufen um zu prüfen, ob der Button
// im vergangenen Loop gedrückt wurde
// Zusatzfunktionen zum z.B. rendern: Siehe Strukturfunktionen
```

## Spielfunktionen

```
bool RECT_Collide(SDL_Rect rect1, SDL_Rect rect2);
      // Prüft, ob zwei SDL_Rect kollidieren
double distance(Vector v1, Vector v2);
      // Gibt den Abstand zweier Vektoren zurück
```

## Strukturfunktionen

```
void <STRUKTUR>_Initialize(SDL_Renderer * renderer
      [, type1 [*] Param1 [,type2 [*] Param2 [,...]]]);
      // Unterfunktionen von INITIALIZE() für Strukturen
<STRUKTUR> <STRUKTUR>_CreateDefault(type1 [*] Param1
      [,type2 [*] Param2 [,...]]);
      // Erstellt ein Standardobjekt von einem struct <STRUKTUR>
      // Unterfunktionen von INITIALIZE() für Strukturen
void <STRUKTUR>_Draw(SDL_Renderer * renderer, <STRUKTUR> * obj);
      // Unterfunktionen von DRAWFRAME() für Strukturen
void <STRUKTUR>_Update(<STRUKTUR> * obj
      [, type1 [*] Param1 [,type2 [*] Param2 [,...]]]);
      // Unterfunktionen von UPDATE() für Strukturen
void <STRUKTUR>_DestroyObject(<STRUKTUR> * obj);
```

```
      // Unterfunktionen von free() für Strukturen
void <STRUKTUR>_Deinitialize();
      // Unterfunktionen von QUIT() für Strukturen
// ggf. Zusatzfunktionen für Aktionen, die vom Spieler ausgelöst werden
```

## Zentrale Strukturierte Datentypen

```
struct vectorStruct {
    double x, y;
}; // Vektor für die Berechnung von Geschwindigkeiten, Kräften und
   // Beschleunigungen für Objekte

struct ballStruct {
    Vector Location, Momentum;
    SDL_Rect TargetRect;
    double Size, Rotation, RotationValue;
    int TextureIndex;
}; // Objekt für die Eigenschaften des Balls

struct paddleStruct {
    double XLocation; // Notice: Locked Y-Coordinate
    SDL_Rect TargetRect;
    int XSize, TextureIndex;
}; // Objekt für die Eigenschaften des Paddles

struct blockStruct {
    Vector Location;
    SDL_Rect TargetRect;
    int XSize, YSize, TextureIndex;
}; // Objekt für die Eigenschaften des Paddles

struct buttonStruct {
    SDL_Rect TargetRect;
    void (*OnClick)();
    // Funktionspointer der aufgerufen wird, wenn der Button geklickt wird
}; // Objekt für die Eigenschaften eines Buttons
```