Dokumentation: Spielentwicklung in Processing mit Max 8 Sound-Integration

Kurze Info: Falls der Punkt nach "data" nach dem Entzippen fehlt, füge ihn einfach wieder hinzu.

Einleitung

Diese Dokumentation beschreibt die Entwicklung eines interaktiven Spiels mit Processing und der Integration von Sound über Max 8.

Projektüberblick

Spielidee:

Bullet-Hell Shooter: Features und Mechaniken

1. **Spielermechanik:**
    - Der Spieler steuert ein Rechteck mit den Pfeiltasten (oder W/A/S/D).
    - Der Player schießt mit der Leertaste Projektile.
2. **Gegner und Projektile:**
    - Gegner spawnen in der oberen Bildschirmhälfte und feuern kontinuierlich Projektile.
    - Es gibt verschiedene Gegnertypen:
        - Normaler Gegner: Langsame, einfache Schüsse.
        - Boss: Schießt viele Projektile in alle Richtungen.
3. **Schwierigkeit:**
    - Gegner und ihre Projektile werden schneller und komplexer mit der Zeit.
    - Die Spawnrate der Gegner erhöht sich, und Bossgegner erscheinen alle paar Sekunden.
4. **Treffer und Leben:**
    - Der Spieler hat 3 Leben. Ein Treffer durch ein gegnerisches Projektil kostet 1 Leben.
    - Gegner müssen mit Treffern besiegt werden.
5. **Punkte und Highscore:**
    - Für jeden getroffenen Gegner erhält der Spieler Punkte.
    - Nach dem Tod wird der Highscore angezeigt.

Technologien:

- Processing (Grafik & Logik)
- Max 8 (Sound)
- OSC (Kommunikation zwischen beiden)

Entwicklungsumgebung

Installation: Processing, Max 8 und OSC-Bibliothek installieren.

Verbindung: OSC für die Kommunikation einrichten.

Kommunikation mit OSC

OSC (Open Sound Control) ermöglicht die Echtzeitkommunikation zwischen Processing und Max 8. Dabei werden Nachrichten über das Netzwerk gesendet. In Processing wird eine OSC-Bibliothek verwendet, um Nachrichten an Max 8 zu senden und zu empfangen.

Grundlegender OSC-Workflow:

1. Ein OSC-Objekt in Processing erstellen und eine Netzwerkverbindung initialisieren.
2. Nachrichten an eine bestimmte IP-Adresse und einen Port senden.
3. Max 8 empfängt diese Nachrichten und interpretiert sie zur Steuerung von Sounds.
4. Max 8 kann ebenfalls OSC-Nachrichten an Processing zurücksenden.

Beispielcode für Processing:

```
import oscP5.*;
import netP5.*;

OscP5 oscP5;
NetAddress max;

void setup() {
  size(400, 400);
  oscP5 = new OscP5(this, 12000); // Port für eingehende Nachrichten
  max = new NetAddress("127.0.0.1", 7400); // Zieladresse von Max 8
}

void draw() {
  // Beispiel: Eine OSC-Nachricht senden
  OscMessage msg = new OscMessage("/triggerSound");
  msg.add(1); // Parameter
  oscP5.send(msg, max);
}
```

Spielmechanik

Grundstruktur:

Die **Grundstruktur des Spiels** basiert auf der Klasse `Game`, die als zentrale Steuerungseinheit für das gesamte Spiel fungiert. Sie verwaltet die Spielzustände, Objekte, Level-Logik und die Benutzerinteraktionen. Hier ist eine Übersicht der wichtigsten Bestandteile:

1. Grundlegender Aufbau des Spiels

Das Spiel folgt einer klassischen **State-Machine-Struktur**, bei der verschiedene Bildschirmzustände (`screen`) definiert sind:

- `0` = Startbildschirm
- `3` = Spielbildschirm (eigentliche Spielmechanik)
- `4` = Game-Over-Bildschirm
- `5` = Sieg-Bildschirm
- `6` = Intro-Bildschirm

Der aktuelle Zustand wird in der `screen`-Variable gespeichert und entsprechend in der `draw()`-Methode gezeichnet.

2. Verwaltung der Spielobjekte

Das Spiel besteht aus mehreren wichtigen Objekten, die in Listen organisiert sind:

- **Spieler (`Player`)** – das Hauptobjekt, das gesteuert wird
- **Gegner (`Enemy`)** – Gegner, die den Spieler angreifen
- **Spieler-Projektile (`playerBullets`)** – Kugeln, die vom Spieler abgefeuert werden
- **Gegner-Projektile (`enemyBullets`)** – Kugeln, die von Gegnern abgefeuert werden
- **Power-Ups (`powerUps`)** – Objekte, die Boni verleihen

Die Objekte werden im Konstruktor der `Game`-Klasse initialisiert und in der `playGame()`-Methode aktualisiert und gezeichnet.

3. Spielmechanik & Level-Logik

Das Spiel basiert auf einem **Level-System**, wobei `currentLevel` die aktuelle Levelnummer speichert.

- Die `loadLevel(int levelNumber)`-Methode lädt das gewünschte Level, indem ein neues `Level`-Objekt erstellt und initialisiert wird.
- `timeRemaining` steuert die verbleibende Zeit für das Level.
- Jedes Level hat einen Eintrag in der `levelTimers`-HashMap (z. B. Level 1 = 30 Sekunden Spielzeit).

Sobald die Zeit abläuft oder alle Gegner besiegt sind, wird die `levelCompleted()`-Methode aufgerufen, um zum nächsten Level zu wechseln oder das Spiel als gewonnen zu markieren.

4. Benutzersteuerung & Eingaben

Die Methode `keyPressed()` verarbeitet Tasteneingaben und ermöglicht:

- **Spielstart (`ENTER`)** – Startet das Spiel und lädt das erste Level.
- **Schießen (`LEERTASTE`)** – Der Spieler kann Projektile abfeuern.
- **Neustart bei Game Over oder Sieg (`ENTER`)** – Das Spiel wird zurückgesetzt.

Diese Methode steuert auch Übergänge zwischen Bildschirmen mit `triggerTransition(int newScreen)`.

5. Bildschirmverwaltung & Übergänge

Das Spiel verfügt über mehrere Bildschirme (`StartScreen`, `GameOverScreen`, `WinScreen`, etc.), die in der `renderScreenContent(int currentScreen)`-Methode angezeigt werden.

- Die Methode `renderTransition()` sorgt für sanfte Übergänge zwischen den Bildschirmen.
- `triggerTransition(int newScreen)` startet den Wechsel zu einem neuen Bildschirm (z. B. nach Game Over).

6. Sound- und Highscore-Management

- **Soundsteuerung:**
  Das Spiel spielt verschiedene Sounds für Hintergrundmusik, Schüsse oder Game-Over-Ereignisse mit `sound.sendSound("...")`.

- **Highscore-Verwaltung:**
  Der Highscore wird in einer Datei gespeichert (`highscore.txt`) und mit `loadHighScore()` geladen bzw. mit `saveHighScore()` aktualisiert.

7. Game-Loop (draw-Methode)

Die zentrale `draw()`-Methode wird in jedem Frame ausgeführt:

1. Prüft, ob ein Bildschirmwechsel stattfindet (`renderTransition()`).
2. Zeichnet den aktuellen Bildschirm (`renderScreenContent(screen)`).
3. Falls `screen == 3` (Spielbildschirm):
   - Hintergrundbild setzen
   - Level-Timer aktualisieren (`timeRemaining`)
   - Spielobjekte aktualisieren (`player.update()`, `enemies.update()`, `bullets.update()`)
   - Gegner und Power-Ups verwalten
   - Kollisionen überprüfen (z. B. `bulletHitsEnemy()`)

Zusammenfassung

Die `Game`**-Klasse** dient als **zentrale Steuerungseinheit**, die:

- Den **Spielzustand** verwaltet (`screen`, `gameOver`, `gameStarted`)
- Alle **Spielobjekte** speichert und aktualisiert (`Player`, `Enemy`, `Bullet`, `PowerUp`)
- Die **Level-Logik** steuert (`loadLevel()`, `levelCompleted()`)
- Benutzerinteraktionen verarbeitet (`keyPressed()`)
- **Bildschirme und Übergänge** verwaltet (`renderScreenContent()`, `triggerTransition()`)
- **Sound und Highscore** speichert

Interaktion mit Max 8

Das Spiel nutzt **Open Sound Control (OSC)** zur Kommunikation mit **Max/MSP**, um Soundeffekte und Musik dynamisch zu steuern. Die zentrale Schnittstelle ist die `SoundManager`**-Klasse**, die OSC-Nachrichten sendet und empfängt.

1. Kommunikation mit Max/MSP

Max/MSP verarbeitet Sound und reagiert auf Nachrichten, die das Spiel über OSC sendet. Die Kommunikation erfolgt über folgende Mechanismen:

Senden von OSC-Nachrichten (Processing → Max/MSP)

- Das `SoundManager`-Objekt nutzt `sendSound(String command)`, um OSC-Nachrichten an Max/MSP zu senden.
- Die Nachricht enthält einen **Befehl (command)**, der in Max/MSP eine Aktion auslöst, z. B.:
  - `"ambient"` – Startet die Hintergrundmusik
  - `"button"` – Spielt einen Button-Klick-Sound
  - `"hit"` – Löst den Sound eines Schusses aus
  - `"stop"` – Stoppt alle Sounds

🔹 **Technisch umgesetzt durch:**

```
void sendSound(String command) {
  OscMessage nachricht = new OscMessage("max");
  nachricht.add(command);
  receiver.send(nachricht, sender);
}
```

🔹 **Beispiel für einen Soundbefehl in `setup()` zum Start der Hintergrundmusik:**

```
sound.sendSound("ambient");
```

Empfangen von OSC-Nachrichten (Max/MSP → Processing)

- **Momentan wird kein direkter Empfang implementiert**, aber die `SoundManager`-Klasse kann OSC-Nachrichten von Max/MSP empfangen.
- Falls erforderlich, könnte `OscP5` in der `SoundManager`-Klasse eine `oscEvent(OscMessage msg)`-Methode haben, um Nachrichten von Max/MSP zu verarbeiten.

2. Wo werden Sounds im Spiel abgespielt?

Setup-Phase (`setup()`)

- Beim Start des Spiels wird **die externe Max/MSP-Patch-Datei** geladen:

  ```
  launch(sketchPath("./data./Sound.maxpat"), "-g");
  ```

- Danach wird der `SoundManager` initialisiert:

  ```
  sound = new SoundManager("127.0.0.1", 11000, 12000);
  ```

- Direkt nach der Initialisierung beginnt die Hintergrundmusik:

  ```
  sound.sendSound("ambient");
  ```

Interaktion mit dem Spieler

1️⃣ Mausklick-Sounds (`mousePressed()`)

- Wird ein Button angeklickt, sendet das Spiel einen `"button"`-Sound an Max/MSP:

```
if (mouseButton == LEFT) {
  sound.sendSound("button");
}
```

2️⃣ Schießen-Sound (`keyPressed()`)

- Drückt der Spieler die **Leertaste (`' '`)**, wird der `"hit"`-Sound abgespielt:

```
if (key == ' ') {
  sound.sendSound("hit");
}
```

3️⃣ Spiel beenden (`exit()`)

- Beim Beenden des Spiels wird sichergestellt, dass die Sounds gestoppt werden:

```
sound.stopSound();
```

3. Verbindung zur `Game`-Klasse

- In `Game` werden bestimmte Ereignisse durch Soundeffekte ergänzt:
  - **Spielstart →** `"ambient"`
  - **Schießen →** `"hit"`
  - **Button-Klick →** `"button"`
  - **Spielende →** `"stop"`
- Diese Sounds machen das Spiel **interaktiver und immersiver**, indem sie wichtige Aktionen akustisch verstärken.

Zusammenfassung

- **Processing → Max/MSP:**
  - Über `SoundManager.sendSound(command)` sendet das Spiel Sound-Befehle an Max/MSP.
- **Verwendete Sounds:**
  - `"ambient"` → Hintergrundmusik
  - `"button"` → Klickgeräusch
  - `"hit"` → Schussgeräusch
  - `"stop"` → Stoppt alle Sounds
- **Eingebaut in:**
  - `setup()` (Musikstart)
  - `mousePressed()` (Button-Klick-Sound)
  - `keyPressed()` (Schießen-Sound)
  - `exit()` (Stopp aller Sounds)

Damit wird Max/MSP als **externer Soundprozessor** genutzt, während Processing die **Spielmechanik und Interaktion** steuert.

Spielanleitung

Spielziel:
Steuere den Spieler, weiche Hindernissen aus, sammle Power-Ups und schieße Gegner ab, um den höchstmöglichen Punktestand zu erreichen.

Steuerung:

- Links: 'A' oder Pfeiltaste ← Links
- Rechts: 'D' oder Pfeiltaste → Rechts
- Hoch: 'W' oder Pfeiltaste ↑ Hoch
- Runter: 'S' oder Pfeiltaste ↓ Runter
- Leertaste (SPACE) – Schieße mit deiner Waffe auf Gegner.
- Enter (↵) – Starte das Spiel oder bestätige den Game-Over-Bildschirm.

Spielmodi

- Kampagnenmodus: Durchlaufe nacheinander 9 Level mit steigender Schwierigkeit.
- Einzelspielermodus: Spiele ein einzelnes Level ohne Kampagnenfortschritt.

Spielmechanik:

1. **Bewegung:**
    - Der Spieler bewegt sich mit den Pfeiltasten oder WASD.
    - Die Bewegung ist auf den Bildschirmbereich begrenzt.
2. **Schießen:**
    - Der Spieler kann Projektile abschießen.
    - Falls das Multi-Schuss-Power-Up aktiv ist, werden drei Projektile abgefeuert.
3. **Power-Ups:**
    - **Schild-Power-Up:**
        - Aktiviert einen Schutzschild (blaue Aura).
        - Läuft nach einer bestimmten Zeit (300 Frames) ab.
    - **Multi-Schuss-Power-Up:**
        - Erlaubt dem Spieler, drei Projektile gleichzeitig abzufeuern.
        - Läuft nach einer bestimmten Zeit (300 Frames) ab.
4. **Lebensanzeige:**
    - Der Spieler startet mit 3 Leben.
    - Bei Kollision mit Gegnern verliert der Spieler ein Leben.
    - Verliert der Spieler alle Leben, endet das Spiel.
5. **Punktesystem:**

- Jeder abgeschossene Gegner gibt Punkte.
- Der Punktestand wird im Spiel aktualisiert.

6. **Anzeige von Power-Up-Timern:**
   - Schild- und Multi-Schuss-Power-Ups haben Timer-Balken.
   - Diese zeigen die verbleibende Dauer der aktiven Power-Ups an.