Ausarbeitung

Ein Snake Spiel mit Darstellung über einen Raspberry Pi auf eine LED Matrix

Zoe Luca Günther

18. August 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

In diesem Projekt habe ich ein Snake Spiel entwickelt. Hierbei ist das besondere, dass man dieses Snake Spiel über die Lagesensoren des Handys steuern kann. Man verbindet sich mit einer Website, geht in die Warteschlange rein und sobald man an der Reihe ist, wird man zur Spielsteuerung weitergeleitet [1][2]. Am Anfang jedes Spiels wird ein kurzer Hinweis am Handy eingeblendet, welcher einem sagt, dass man das Handy vertikal und flach halten soll und die Bildschirmrotation deaktiviert werden sollte [3][4]. Dies ist wichtig, damit die Steuerung richtig funktioniert, da man um das Spiel zu steuern, lediglich sein Handy neigen muss. Um Das Spielerlebnis so gut wie möglich zu halten, muss das Handy flach und vertikal gehalten werden, da so die Steuerung am besten funktioniert, da diese sehr sensibel sein kann. Um das Spielgeschehen zu sehen, wird ein Raspberry Pi mit einer 64x64 LED Matrix verwendet, worauf der Score und das Spiel dargestellt werden. Um das Spiel zu pausieren, klickt man lediglich auf seinem Handy auf den Pause Button [5]. Dies geht jedoch nur, wenn man gerade an der Reihe ist. Um dies zu überprüfen, wird beim ersten betreten der Spiel Website ein Cookie gesetzt, worin die User ID steht. Jede ID ist unterschiedlich, um eine genaue Identifizierung zu ermöglichen. Sollte sich ein Spieler für mehr als 20 Sekunden im Pause Status befinden, oder mehr als 20 Sekunden keine Bewegung mehr machen, wird das Spiel beendet und der nächste Spieler aus der Warteschlange kann spielen. Nachdem man sich selber als Schlange gefressen hat, oder man zu lange im Spiel war, wird man auf eine Game Over Seite weitergeleitet, wo der eigene Score eingeblendet wird [6].

2 Vorgehensweise

Das Projekt wurde in verschiedene Unterpunkte eingeteilt, um die bestmögliche Implementierung sicherzustellen.

2.1 Grundgerüst

Das Grundgerüst besteht aus verschiedenen Klassen, welche für den Spielablauf essentiell sind. Folgende Klassen beinhaltet das Grundgerüst:

Klassenname	Beschreibung
Player.py	Zuständig für die Bewegung des Spielers, des Überprüfens auf Kollision, das Score Handling und das Fressen von Futter
Logger.py	Ausgeben von Nachrichten in der Konsole zu Debug Zwecken. Es kann ein Prefix der Klasse Prefix.py angegeben werden
Playground.py	Handling des Spielfeld, setzen von Blöcken[1] auf dem Spielfeld, setzen einer Random Futter Position, Prüfen ob das Spielfeld voll ist
Queue.py	Generelles Queue Handling (Spieler hinzufügen, entfernen aus Queue), Nächsten Spieler nehmen, UserId verifizieren[2]
SnakeGame.py	Game Status setzen, Game starten / stoppen / pausieren, Game loop, Game resetten, Game over Handling

Tabelle 1: Grundgerüst Klassen

Klassenname	Beschreibung
Direction.py	Richtungsangaben
GameStatus.py	Gamestatus mit Gamestatus Text
Message.py	Game over Nachrichten
PlaygroundTile.py	Spielfeld Blöcke (zum Beispiel Wand, Snake, Futter)
Prefix.py	Prefix zu Debug Zwecken

Tabelle 2: Enum Klassen

Zum testen der Funktionalitäten, wurde das Spiel vorerst mithilfe der Bibliothek *pygame* implementiert. Somit konnte das Spiel dargestellt werden, die Bewegungsabläufe getestet und der Spielablauf angepasst werden. Für den Gameloop wurde jedoch zu diesem Zeitpunkt die pygame eigenen Funktionen verwendet. Die Steuerung wurde vorerst über die Tastatur implementiert, da die spätere Steuerung über die Website mithilfe der API Schnittstelle funktioniert.

2.2 Website

Der Hauptteil der Website besteht aus dem Styling (CSS[3]) und den Funktionen (JS[4]), da im Hintergrund auf die API Schnittstelle zugegriffen werden muss und um Animationen für den Benutzer anzuzeigen, um das Benutzererlebnis an erster Stelle zu halten. Somit ist die Website speziell für die Mobilen Endgeräte konzipiert, da die Steuerung über dieses im weiteren Verlauf erfolgt. Was diese Spezialisierung ausmacht ist, dass die Schriftgröße besonders auf kleine Geräte angepasst ist und die Ladezeiten so gering wie möglich gehalten werden, da diese Seite meist aus dem normalen Mobilfunknetz

[1] Das Spielfeld besteht aus Blöcken, welche in einem 2d Array gespeichert sind
[2] Überprüfen ob die UserId die nötige länge hat
[3] Cascading Style Sheets, Gestaltungs- und Formatierungssprache
[4] JavaScript, für dynamisches HTML

aufgerufen wird. Außerdem wird ein responsive Design[5] verwendet, um die Website jederzeit an fast jedes Endgerät anzupassen. “Mobile first” spielt hierbei eine große Rolle, unter dem Aspekt der Steuerung, aber auch unter dem Aspekt, dass man davon ausgehen kann, dass die meisten Websites heutzutage mit dem Smartphone aufgerufen werden, da man dies zu jederzeit dabei haben kann.
Die Website besteht aus drei HTML Seiten:

Dateiname	Beschreibung
index.html	“Startseite” der Website. Joinen der Queue
controller.html	Steuerung der Snake
gameover.html	Game over Screen nach beenden des Spiels.

Tabelle 3: HTML Dateien

Um die Funktionalität der Website zu gewährleisten, wird wie oben erwähnt, JS verwendet. Hierfür wird jeder HTML Datei eine JS Datei zugewiesen:

Dateiname	Beschreibung
controller.js	Funktionalität und anzeige der Steuerung. Starten und pausieren des Spiels
gameover.js	Laden und anzeigen des Scores.
index.js	Anzeigen der aktuellen Spieler in der Warteschlange. Betreten und verlassen der Warteschlange. Anzeigen der Warteschlangen Animation.
constants.js	Konstante Variablen für die API Schnittstelle.

Tabelle 4: JS Dateien

Hierbei ist zu beachten, dass die Konstanten Variablen in jeder HTML Datei eingebunden sind, um jederzeit Zugriff auf die API Schnittstelle zu bekommen. Hinzu kommen Klassen, welche als Handler dienen:

[5]Responsive bedeutet in dem Fall zum Beispiel “auf jemanden eingehen” oder “reaktionsfähig bleiben”

Dateiname
controller_handler.js
cookie_handler.js
queue_handler.js
score_handler.js

Tabelle 5: Handler Klassen

Diese Klassen beinhalten alle nötigen Funktionen, um auf die API Schnittstelle zuzugreifen und die dadurch entstandenen Daten zurückzugeben, um das handling in den jeweiligen Klassen zu vereinfachen. Es ist zu beachten, dass Funktionen teilweise asynchron sind, was daran liegt, dass Daten, welche zur weiteren Verarbeitung benötigt werden (zum Beispiel zur Darstellung), erst zurückgegeben werden können, wenn diese vorliegen. Somit werden diese Funktionen in einem Promise gehandhabt. Die API Zugriffe werden über einen Standard fetch Aufruf gehandhabt.
Um das Benutzererlebnis zu verbessern, werden beim Game over Animationen angezeigt, welche den Score darstellen. Somit kann damit ein positives Erlebnis verbunden werden. Ein API Aufruf kann wie folgt aussehen:

```
async getScore() {
        let score = await fetch(API_ENDPOINTS["GET_SCORE"], {method: "
    GET"})
            .then(response => response.json())
            .then(data => {
                if(data["score"] != undefined) {
                    return data["score"];
                }
            })
            .catch(error => {
                console.log(error);
                return -2;
            });

        return score;
    }
```

Der Aufruf dieser Funktion erfolgt über ein Promise, wie bereits oben erwähnt:

```
Promise.resolve(scoreHandler.getScore())
        .then(score => {
            scoreElement.innerHTML = score + " Punkt" + (score == 1 ?
    "" : "e") + "!";
        });
```

2.3 API Schnittstelle

Das Web Framework *fastapi* ist ein modernes, hoch performanted Python Framework. Es ist einfach zu handhaben und bietet alle nötigen Funktionen, welche in diesem Projekt benötigt werden. Dies sind verschiedene CRUD[6] Operationen, das mitgeben eines HTML Statuscodes und das auslesen von Cookies, um die UserId abzufragen. Um die verschiedenen Klassen aus Unterabschnitt 2.1 zu verwenden, muss dem Systempfad

[6]create, read, update, delete

ein weiterer Pfad hinzugefügt werden, dem Übergeordneten Ordner, da sich die Backend API in einem eigenen Unterordner befindet:

```
import sys

sys.path.append('../snake')

"""
Ordnerstruktur:

snake
  backend
  static
  enums
"""
```

Jede Schnittstelle hat anfangs den gleichen Ablauf: Es wird überprüft, ob ein UserId Cookie vorhanden ist. Ist dies nicht der Fall, wird durch den Logger, welcher in Unterabschnitt 2.1 beschrieben ist, eine Mitteilung ausgegeben und der weitere Zugriff verwährt. Es wird ein JSON Objekt zurückgegeben, welches eine Nachricht und den Erfolgsstatus beinhaltet. Der Statuscode wird hierbei auf *401 UNAUTHORIZED* gesetzt:

```
if not userId:
  Logger.log(f"{userId}: UNAUTHORIZED", Prefix.API)
  response.status_code = status.HTTP_401_UNAUTHORIZED
  return {"message": "userId cookie not set", "success": False}
```

Sind die weiteren Voraussetzungen erfüllt, wie in etwa ob der Spieler, welcher eine Bewegung angefragt hat, momentan an der Reihe ist, wird die jeweilige Aktion ausgeführt und der Benutzer, welcher die Anfrage gestellt hat, bekommt eine Erfolgsmeldung:

```
return {"message": f"success message", "success": True}
```

Die JSON Objekte werden im Frontend verarbeitet.
Folgende Operationen werden durch die API geboten:

Pfad	REST (CRUD Operation)	Beschreibung
/move/direction	Put (update)	Bewegt die Snake in eine neue Richtung
/start	Put (create)	Startet das Spiel
/pause	Put (update)	Pausiert das Spiel
/queue/join	Put (create)	Tritt der Queue bei
/queue/leave	Put (update)	Verlässt die Queue
/queue/length	Get (read)	Fragt die aktuelle Queue größe ab. Wenn ein Spieler momentan spielt, zählt dieser als ein Spieler in der Queue extra
/current_user	Get (read)	Fragt ab, ob der Spieler, welcher die Anfrage gestellt hat, berechtigt ist zu Spielen (als nächstes dran ist)
/surrender	Delete (delete)	Gibt das Spiel direkt auf (für Debug Zwecke)
/gameover	Get (read)	Gibt zurück ob das Spiel vorbei ist und ob der Benutzer momentan Berechtigt ist zu spielen
/score	Get (read)	Gibt den Score zurück, sofern der Spieler bereits ein Spiel gespielt hat
/threads	Get (read)	Gibt alle laufenden Threads aus (für Debug Zwecke)

Tabelle 6: API Methoden

Damit die API gestarten werden kann, wird der ASGI[7] Web server *uvicorn* verwendet. Zu beachten ist hierbei, dass uvicorn über das Python Skript der API Schnittstelle gestartet wird, da es sonst zu Fehlern kommt, wenn das Spiel gestartet wird, da der Thread für die Darstellung auf dem Raspberry Pi versucht, die Startparameter zu übernehmen und dadurch eine Exception entsteht, da die Startparameter von uvicorn nicht in der Bibliothek *rpi-rgb-led-matrix* gefunden werden können, da diese dort nicht existieren. Die Startfunktion der API sieht schlussendlich wie folgt aus:

```
1 if __name__ == "__main__":
2     uvicorn.run("app:app", port=80, host="::", reload=True, debug=True
    , log_level="info", workers=1)
```

2.4 Darstellung

Für die Darstellung des Spielfeldes und des Scores wird die Bibliothek *rpi-rgb-led-matrix* verwendet. Diese Bibliothek ist auf https://github.com/hzeller/rpi-rgb-led-matrix zu finden. Sie wird direkt auf dem Raspberry Pi installiert und kann direkt in Python eingebunden werden. Dank dieser Bibliothek kann man einzelne Pixel darstellen, oder ganze Texte. Es werden verschiedene Schriftgrößen mitgeliefert. Um das Spielfeld darzustellen, wird die Spielfeldmatrix genommen und in Pixel übersetzt. Jeder Spielfeldblock wird auf der LED Matrix in der Größe 2x2 dargestellt, um die Sichtbarkeit zu erhöhen. Außerdem wurde der Hintergrund dunkler eingestellt von der Helligkeit her, um das Spielerlebnis zu fördern. Die Display Klasse, welche für die Darstellung zuständig ist,

[7] Asynchronous Server Gateway Interface

ist von der Klasse *SampleBase.py* abgeleitet und beinhaltet daher die Funktion *process*, welche den infinity loop und somit die Darstellung startet. Die Display Klasse beinhaltet daher eine Funktion *terminate*, um den loop zu unterbrechen. Die damit verbundenen Schwierigkeiten sind in Abschnitt 3 beschrieben.

3 Schwierigkeiten

Die größte Schwierigkeit war es, die API nicht zu unterbrechen, während das Spiel läuft. Es wurde *multithreading* verwendet, um verschiedene Threads parallel laufen zu lassen. Somit laufen im fertigen Projekt, der Thread für die Darstellung des Spiels auf dem Raspberry Pi, der Gameloop, die Überprüfung ob ein Spieler keine Inputs mehr macht und somit nicht mehr anwesend ist (AFK[8]) und die API Schnittstelle parallel. Es wurde versucht, einen neuen Prozess für die Darstellung auf dem Display zu verwenden, jedoch können live updates in einem separaten Prozess nicht mehr nachverfolgt werden, weshalb die Thread Variante verwendet worden ist. Die Threads wurden als *daemon* Threads gestartet, damit man sie zu jedem Zeitpunkt durch ein Interrupt unterbrechen kann. Die Implementierung sieht wie folgt aus, sobald die start Methode von der API Schnittstelle aufgerufen wird:

```
1  import threading
2
3  # Threads initialisieren
4  displayThread = threading.Thread(name="Display", target=display.
       process)
5  displayThread.daemon = True
6
7  gameLoopThread = threading.Thread(name="Gameloop", target=self.loop)
8  gameLoopThread.daemon = True
9
10 gameLoopAfkThread = threading.Thread(name="GameloopAfk", target=self.
       loopAfkCheck)
11 gameLoopAfkThread.daemon = True
12
13 # Threads starten
14 gameLoopThread.start()
15 displayThread.start()
16 gameLoopAfkThread.start()
17
18 # Threads beenden
19 gameLoopThread.join()
20 self.loopAfkCheckRunning = False
21 gameLoopAfkThread.join()
22 display.terminate()
23 displayThread.join()
```

Entscheidend ist hierbei die Reihenfolge, in der die Threads gestartet und gestoppt werden. Zu beachten ist außerdem, dass der Display Thread eine Funktion benötigt, um terminiert zu werden, ansonsten kann er nicht gestoppt werden. die Loops greifen jeweils auf den Gamestatus zu, um den in jedem Thread integrierten *infinity loop* zu unterbrechen, sollte sich der Gamestatus ändern und der Thread nicht mehr benötigt werden. Somit erklärt sich auch die Reihenfolge, in welcher die Thread beendet werden.

[8] away from keyboard, abwesend

Sobald das Spiel vorbei ist und der Gameloop beendet wird, kann die afk Überprüfung ebenfalls beendet werden. Da das Spiel vorbei ist wird außerdem zum Schluss der Displaythread beendet, da es kein Spiel mehr zum darstellen gibt. Somit gehen alle Aktionen der restlichen Threads vom eigentlichen Gameloop aus.

4 Zusammenfassung und Ausblick

Mithilfe von verschiedenen Bibliotheken ist es möglich, ein Snake Spiel zu entwickeln, welches mithilfe eines Mobilen Endgerätes gesteuert werden kann und auf einer LED Matrix mithilfe eines Raspberry Pi dargestellt werden kann. Dieses Ziel wurde innerhalb dieses Projektes erreicht. Es wurden die Bibliotheken *pygame* und *rpi-rgb-led-matrix*, das Framework *fastapi* und der web server *uvicorn* verwendet. Die Snake Website verfügt über ein Warteschlangensystem, welches über UserIds funktioniert, ein Controller, sowie ein Game Over screen. Im Projekt Pfad ist außerdem eine Datei *Splashscreen.py* zu finden, welche einen Splashscreen auf der LED Matrix anzeigt. Dieser Splashscreen wurde jedoch nicht implementiert, da dies viele weitere Probleme bereitete. Im weiteren Verlauf gilt dies vollständig zu implementieren. Es kann im weiteren Verlauf außerdem eine Highscore Liste implementiert werden, wo man einen Namen am Ende eines Spiels auf dem Mobilen Endgerät eingeben kann, und dieser dann bei erreichen eines Highscores angezeigt wird.

5 Abbildungen

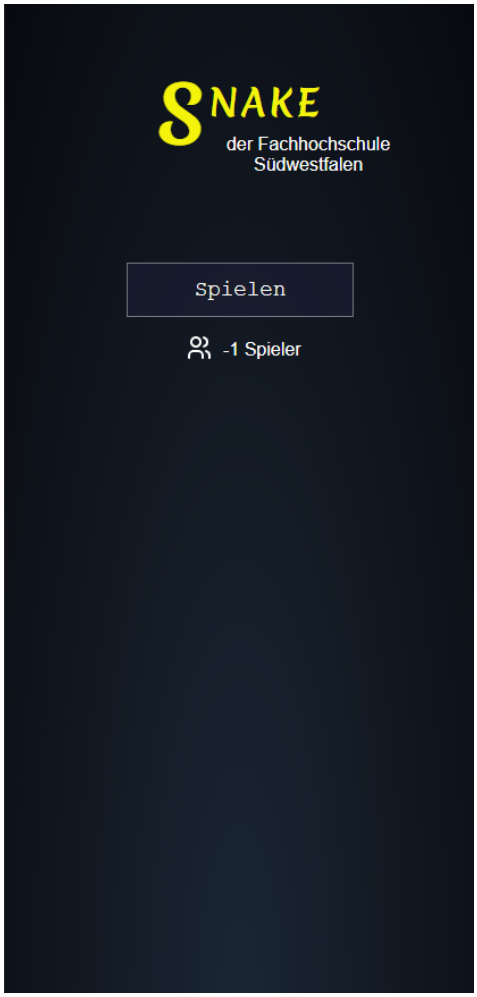

Abbildung 1: Snake Website Hauptmenü

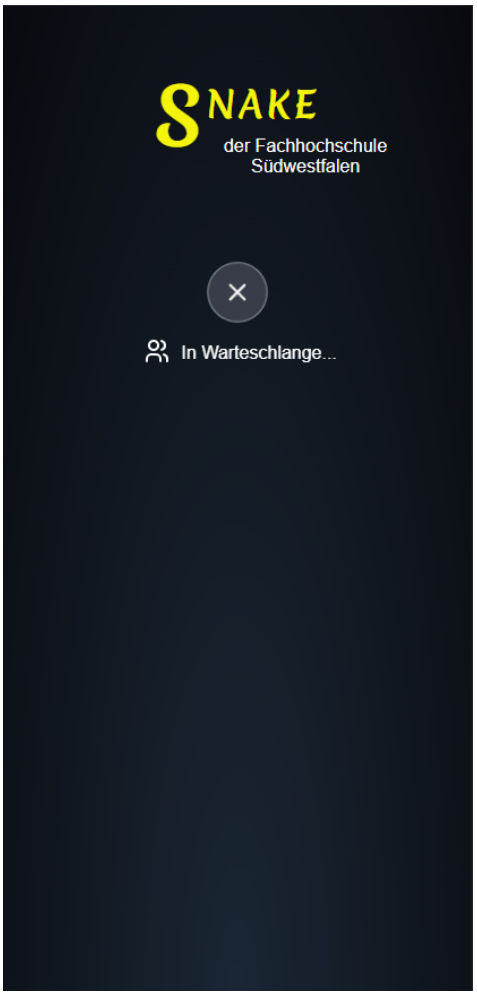

Abbildung 2: Snake Website Hauptmenü in Warteschlange

Abbildung 3: Snake Website Controller Hinweis 1

Abbildung 4: Snake Website Controller Hinweis 2

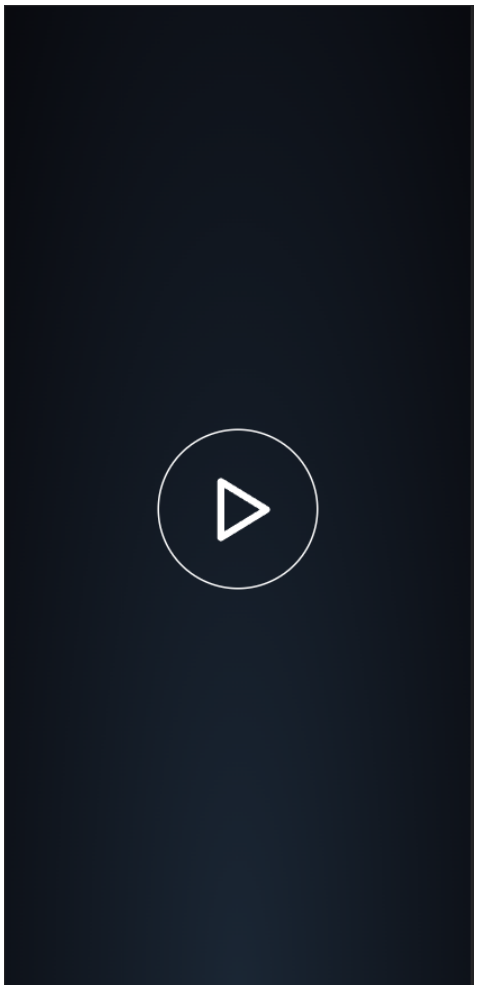

Abbildung 5: Snake Website Controller

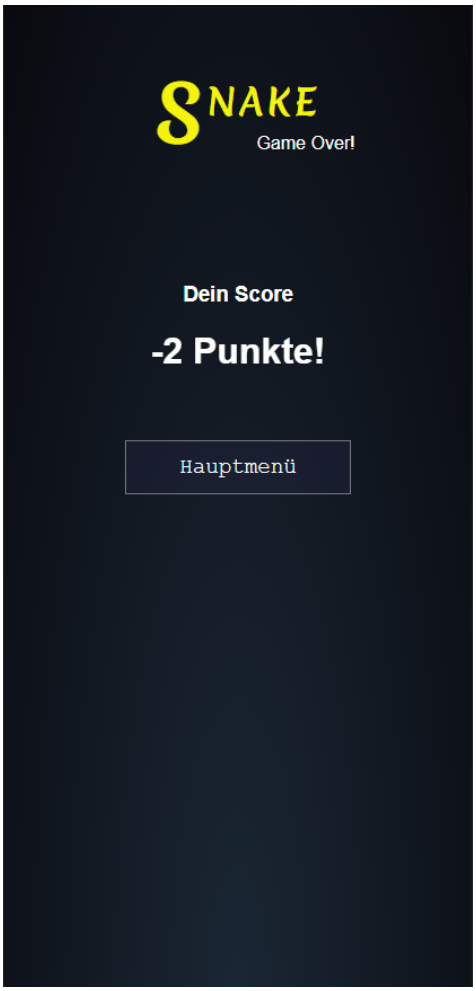

Abbildung 6: Snake Website Game Over Score

6 Quellenverzeichnis

- https://www.konversionskraft.de/trends/die-3-saeulen-des-responsive-webdesign.html
  Stand: 16.08.2022
- https://tetris.informatik.fh-swf.de/snake/index.html
  Stand: 16.08.2022
- https://www.uvicorn.org
  Stand: 18.08.2022