Murmellabyrinth

Hochschule-Anhalt
Lohmannstraße 23
06366 Köthen / Anhalt

Ricardo Hoppe
Modul: Mobile Anwendungen
Betreuender Professor: Professor Cebulla

Inhaltsverzeichnis

1 Einleitung

Das Projekt *Murmellabyrinth* ist ein mobiles Geschicklichkeitsspiel, inspiriert von klassischen, handgefertigten Murmellabyrinthen aus Holz. Der Spieler steuert eine Kugel durch Neigen des Smartphones und muss dabei Hindernisse wie Löcher vermeiden. Ziel ist es, die Kugel vom Startpunkt zum Ziel zu führen, während ein Timer zusätzlichen Druck erzeugt. Durch den Einsatz von Sensorik und Bluetooth-Synchronisation bietet das Spiel eine moderne und interaktive Nutzererfahrung.

2 Zielsetzung

Das Hauptziel dieses Projekts war die Entwicklung eines mobilen Spiels, das realistische physikalische Bewegungen durch die Integration eines Beschleunigungssensors nachbildet. Zusätzlich sollte eine Bestenliste implementiert werden, die über Bluetooth zwischen Geräten synchronisiert werden kann.

3 Methodik

Das Projekt wurde nach einem iterativen Entwicklungsansatz realisiert, der ausfolgenden Schritten bestand.

3.1 Konzeption & Planung

Zu Beginn des Projekts wurde ein detailliertes Konzept erstellt, das die Kernfunktionen und die verwendeten Technologien festlegt. Die Entwicklung der Anwendung erfolgte mit Android Studio in der Programmiersprache Java, wobei verschiedene Hauptkomponenten in die App integriert wurden.

3.1.1 Sensorik

Die Steuerung der Kugel im Spiel erfolgt durch den Beschleunigungssensor des Smartphones. Dieser Sensor misst die Bewegung und Neigung des Geräts, wodurch der Spieler die Kugel durch das Labyrinth bewegen kann. Die präzise Erfassung der Beschleunigungswerte ist dabei entscheidend für die realistische Steuerung der Kugel.

3.1.2 Maze-Generator

Ein dynamischer Algorithmus wurde entwickelt, um zufällige Labyrinthe zu generieren. Diese beinhalten Hindernisse, die vom Spieler umgangen werden müssen, je einen Start- und Zielpunkt. Der Algorithmus sorgt dafür, dass jedes generierte Labyrinth einzigartig ist und den Spieler herausfordert.

3.1.3 SharedPreferences

Um den Fortschritt des Spiels zu speichern, insbesondere die Highscores und Spielerdaten, wurde die Funktion der **SharedPreferences** genutzt. Diese speichert die Daten lokal auf dem Gerät, sodass der Spieler seine Fortschritte auch nach dem Beenden der App wieder aufrufen kann.

3.1.4 Bluetooth

Für die drahtlose Synchronisation der Bestenliste zwischen verschiedenen Geräten wurde die Bluetooth-Technologie integriert. Dies ermöglicht es den Spielern, ihre Highscores mit anderen zu teilen und sich untereinander zu vergleichen.

3.2 Implementierungsphase

Im Anschluss an die Konzeptentwicklung wurde mit der Implementierungsphase begonnen. Hier wurden die geplanten Funktionen nach und nach umgesetzt. Jede Funktion wurde zunächst einzeln entwickelt und anschließend in das bestehende System integriert, wobei stets auf die Kompatibilität und Funktionsweise der anderen Komponenten geachtet wurde.

3.3 Testphase

Schließlich folgte die Phase des Testens und der Optimierung. Hierbei wurden alle Funktionen auf ihre Funktionsfähigkeit geprüft, Fehler identifiziert und die Spielmechanik optimiert.

4 Architektur

Die Architektur des Murmellabyrinths basiert auf einer modularen Struktur, die verschiedene Klassen zur Steuerung der Spiellogik, der Sensorik, der Benutzeroberfläche und der Bluetooth-Funktionalität umfasst. Das Projekt wurde in **Java** entwickelt und verwendet das **Android SDK** zur Umsetzung der App-Funktionalitäten. Es wurde die Entwicklungsumgebung Android Studio benutzt.

4.1 MainActivity.java

Die *MainActivity* bildet den Einstiegspunkt der Anwendung und steuert das Hauptmenü des Spiels. In dieser Klasse kann der Spieler seinen Namen eingeben, ein neues Spiel starten oder auf die Highscore-Liste zugreifen. Zu den wesentlichen Aufgaben der *MainActivity* gehört die **Initialisierung der Benutzeroberflächen-Komponenten (UI)**, um dem Spieler eine benutzerfreundliche Interaktion zu ermöglichen. Weiterhin verwaltet diese Klasse die Speicherung der Highscores mithilfe von **SharedPreferences**, sodass die besten Ergebnisse auch nach dem Schließen der App verfügbar bleiben.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        EditText playerNameInput = findViewById(R.id.player_name_input);
        Button startButton = findViewById(R.id.start_button);
        Button clearButton = findViewById(R.id.clear_button);
        Button bluetoothButton = findViewById(R.id.bluetooth_button);
        highscoreList = findViewById(R.id.highscore_list);

        // Lade gespeicherte Spieler
        players = loadPlayers();

        // Highscores anzeigen
        displayHighscores();
      …
}
```

Code Snippet 1
https://github.com/Cardostyle/Ballgame/blob/master/app/src/main/java/com/example/ballgame/MainActivity.java

Darüber hinaus ist die *MainActivity* für den Wechsel zwischen den verschiedenen Activities verantwortlich, um dem Spieler eine flexible und nahtlose Navigation durch das Spiel zu bieten.

4.2 GameActivity.java

Die Klasse *GameActivity* stellt die Hauptspielumgebung bereit und kümmert sich um die Interaktionen des Spielers mit dem Labyrinth. In dieser Klasse wird das *GameView* eingebunden, das für die Darstellung des Spielfelds verantwortlich ist. Sie verwaltet außerdem den **Spielstand** und die **Highscore-Speicherung**. Die Steuerung erfolgt über den **Beschleunigungssensor**, dessen Werte in Echtzeit

verarbeitet werden:

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (!isPaused) {
        gameView.updateBallPosition(event.values[0], event.values[1]);
    }
}
```

Code Snippet 2
https://github.com/Cardostyle/Ballgame/blob/master/app/src/main/java/com/example/ballgame/GameActivity.java

Ein weiteres wichtiges Element dieser Klasse ist die Integration eines Pause-Menüs, das es dem Spieler ermöglicht, das Spiel jederzeit anzuhalten, vorzeitig zu beenden, neu zu starten oder zu einem späteren Zeitpunkt fortzusetzen.

4.3 GameView.java

Die *GameView* ist die Klasse, die für die visuelle Darstellung des Labyrinths sowie für die physikalische Bewegung der Kugel verantwortlich ist. Sie enthält alle notwendigen Zeichenmethoden, um das Spielfeld korrekt darzustellen.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    for (int y = 0; y < maze.length; y++) {
        for (int x = 0; x < maze[y].length; x++) {
            Rect destRect = new Rect(x * tileSize, y * tileSize, (x + 1) *
tileSize, (y + 1) * tileSize);
            switch (maze[y][x]) {
                case MazeGenerator.WALL:
                    canvas.drawBitmap(wallBitmap, null, destRect, paint);
                    break;
                case MazeGenerator.PATH:
                    canvas.drawBitmap(pathBitmap, null, destRect, paint);
                    break;
                case MazeGenerator.HOLE:
                    canvas.drawBitmap(holeBitmap, null, destRect, paint);
                    break;
                case MazeGenerator.GOAL:
                    canvas.drawBitmap(goalBitmap, null, destRect, paint);
                    break;
            }
        }
    }

    Rect ballRect = new Rect((int) (ballX - ballRadius), (int) (ballY -
ballRadius), (int) (ballX + ballRadius), (int) (ballY + ballRadius));
    canvas.drawBitmap(ballBitmap, null, ballRect, paint);
}
```

Code Snippet 3
https://github.com/Cardostyle/Ballgame/blob/master/app/src/main/java/com/example/ballgame/GameView.java

Darüber hinaus ist die *GameView* für die Logik zuständig, die sicherstellt, dass die Kugel korrekt mit den Hindernissen und Wänden im Labyrinth kollidiert. Dies erfordert eine präzise Kollisionsprüfung. Zudem wird die Position der Kugel

regelmäßig basierend auf den gesammelten Sensor-Daten aktualisiert, sodass die Bewegung des Spielers realistisch und in Echtzeit erfolgt.

```
public void updateBallPosition(float dx, float dy) {
    float newX = ballX - dx * 5;
    float newY = ballY + dy * 5;

    if (canMoveTo(newX, newY)) {
        ballX = newX;
        ballY = newY;

        if (isAtGoal()) {
            mazeGenerator.generateNewMaze();
            maze = mazeGenerator.getMaze();
            resetBallPosition();
            gameActivity.scoreIncrease();
        }

        if (isAtHole()) {
            mazeGenerator.generateNewMaze();
            maze = mazeGenerator.getMaze();
            resetBallPosition();
        }
    }

    invalidate();
}
```

Code Snippet 4
https://github.com/Cardostyle/Ballgame/blob/master/app/src/main/java/com/example/ballgame/GameView.java

4.4 MazeGenerator.java

Die Klasse *MazeGenerator* implementiert einen Algorithmus, der die prozedurale Generierung von Labyrinthen ermöglicht. Sie erstellt zufällige Spielfelder, die Wände, Hindernisse und Löcher enthalten, um die Herausforderung für den Spieler zu erhöhen. Der Start- und Zielpunkt des Labyrinths werden ebenfalls von dieser Klasse gesetzt.

```
public void generateNewMaze() {
    for (int y = 0; y < height + 1; y++) {
        for (int x = 0; x < width + 1; x++) {
            maze[y][x] = WALL;
        }
    }

    generatePath(1, 1);
    addHoles(holeCount);
    setSpawnPoint();
    setGoalPoint();
}
```

Code Snippet 5
https://github.com/Cardostyle/Ballgame/blob/master/app/src/main/java/com/example/ballgame/MazeGenerator.java

Um sicherzustellen, dass das generierte Labyrinth lösbar ist, berechnet der Algorithmus den kürzesten Pfad vom Startpunkt zum Zielpunkt. Dadurch wird gewährleistet, dass der Spieler immer eine Möglichkeit hat, das Labyrinth zu durchqueren, ohne auf unlösbare Hindernisse zu stoßen.

4.5 Player.java

Die *Player*-Klasse verwaltet die Spiellogik und die Zustände des Spielers. Sie speichert den Namen und den aktuellen Highscore des Spielers, sodass dieser jederzeit abgerufen und mit anderen Spielern verglichen werden kann.

```
public class Player implements Parcelable {
    private String name;
    private int highscore;
    …
}
```

Code Snippet 6
https://github.com/Cardostyle/Ballgame/blob/master/app/src/main/java/com/example/ballgame/Player.java

4.6 BluetoothActivity.java

Die *BluetoothActivity*-Klasse ermöglicht es dem Spieler, Highscores über Bluetooth mit anderen Geräten zu synchronisieren. Sie stellt eine Bluetooth-Verbindung zwischen Geräten her, sodass die Highscore-Daten zwischen ihnen ausgetauscht werden können. Dies erfolgt sowohl durch das Senden als auch das Empfangen der Highscore-Daten.

```
private void sendHighscores() {
    try {
        if (bluetoothSocket == null) throw new IOException("Kein BluetoothSocket
verfügbar");
        OutputStream outputStream = bluetoothSocket.getOutputStream();

        for (Player player : players) {
            outputStream.write((player.toString() + "\n").getBytes());
        }

        runOnUiThread(() -> Toast.makeText(this, "Bestenliste gesendet",
Toast.LENGTH_SHORT).show());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Code Snippet 7
https://github.com/Cardostyle/Ballgame/blob/master/app/src/main/java/com/example/ballgame/BluetoothActivity.java

```
private void receiveHighscores() {
     try {
         if (bluetoothSocket == null) throw new IOException("Kein BluetoothSocket
verfügbar");
         InputStream inputStream = bluetoothSocket.getInputStream();
         List<Player> receivedPlayers = new ArrayList<>();
         byte[] buffer = new byte[1024];
         int bytesRead;

         while ((bytesRead = inputStream.read(buffer)) != -1) {
             String data = new String(buffer, 0, bytesRead);
             for (String playerData : data.split("\n")) {
                 try {
                    receivedPlayers.add(Player.fromString(playerData));
                 } catch (IllegalArgumentException e) {
                     Log.d("BluetoothActivity", "Empfangene Daten: " + data);
                     runOnUiThread(() -> Toast.makeText(this, "Ungültige Daten
empfangen: " + playerData, Toast.LENGTH_SHORT).show());
                 }
             }
         }

         mergeHighscores(receivedPlayers);
         runOnUiThread(() -> Toast.makeText(this, "Bestenliste empfangen",
Toast.LENGTH_SHORT).show());
     } catch (IOException e) {
         e.printStackTrace();
         runOnUiThread(() -> Toast.makeText(this, "Empfangen fehlgeschlagen",
Toast.LENGTH_SHORT).show());
     }
 }
```

Code Snippet 8
https://github.com/Cardostyle/Ballgame/blob/master/app/src/main/java/com/example/ballgame/BluetoothActivity.java

Die Klasse sorgt außerdem dafür, dass die Bestenliste zwischen den Geräten synchronisiert wird, sodass alle Spieler ihre aktuellen Platzierungen miteinander vergleichen können. Dies fördert die Interaktivität und den Wettbewerb zwischen den Nutzern und erhöht die Langzeitmotivation weiterzuspielen.

5 Implementierung

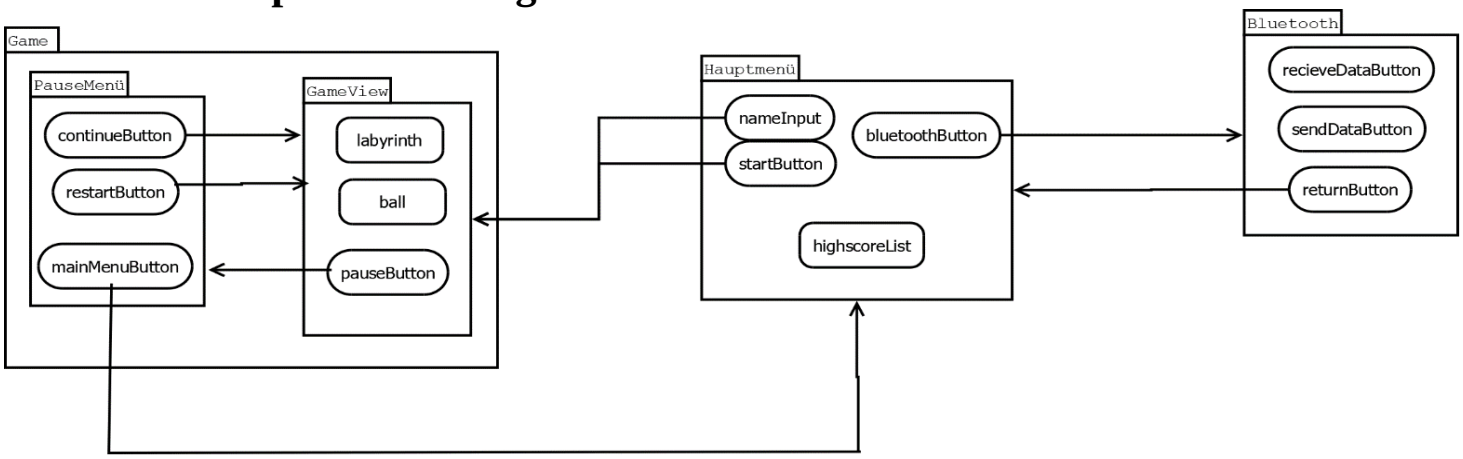

Abbildung 1Architekturdiagramm

5.1 Hauptmenü

Das Hauptmenü der App bietet dem Spieler verschiedene Optionen, die eine einfache Navigation und Steuerung des Spiels ermöglichen. Eine der wichtigsten Funktionen ist die Möglichkeit, ein neues Spiel zu starten. Wenn der Spieler diese Option wählt, wird ein zufälliges Labyrinth mit neuen Parametern generiert, sodass jedes Spiel einzigartig ist. Darüber hinaus gibt es die Möglichkeit, ein bestehendes Spiel zu laden. In diesem Fall werden die gespeicherten Highscores und andere relevante Daten abgerufen, sodass der Spieler genau dort weitermachen kann, wo er zuvor aufgehört hat. Eine weitere Funktion des Hauptmenüs ist die Anzeige der Highscore-Liste. Hier können die besten Spielergebnisse eingesehen werden, sowohl lokal gespeicherte als auch synchronisierte Highscores, die über Bluetooth zwischen Geräten übertragen wurden. Schließlich gibt es eine Option zur Bluetooth-Synchronisation. Diese ermöglicht es dem Spieler, eine Verbindung zu anderen Spielern herzustellen, um Highscores auszutauschen oder sogar in Mehrspieler-Settings zu spielen.

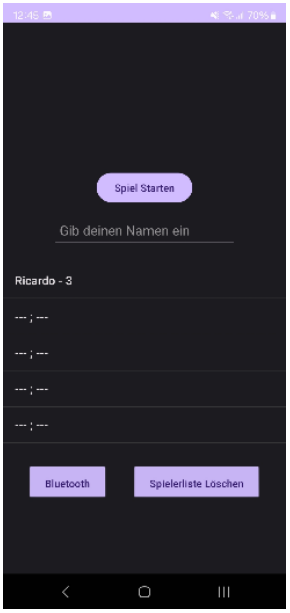

Abbildung 2 Hauptmenü

5.2 Ballsteuerung

Die Ballsteuerung im Spiel erfolgt über die Neigung des Geräts. Dabei kommt der Beschleunigungssensor (Accelerometer) des Smartphones zum Einsatz, um die Bewegung der Kugel zu steuern. Zunächst werden die Beschleunigungswerte auf der X- und Y-Achse erfasst. Diese Werte werden dann in Bewegungsbefehle umgewandelt, die die Kugel im Labyrinth steuern. Während des Spiels wird kontinuierlich überprüft, ob die Kugel mit Hindernissen oder den Rändern des Spielfeldes kollidiert. Falls eine Kollision festgestellt wird, erfolgt eine entsprechende Anpassung der Position der Kugel. Berührt die Kugel eine Wand, wird ihre Geschwindigkeit auf 0 gesetzt, um die ruhige Hand des Spielers zu bevorzugen und hektische Bewegungen zu vermeiden. Die Anpassung der Kugelposition wird dabei in der *GameView* durchgeführt, die für die Darstellung des Spiels zuständig ist.

5.3 Dynamische Labyrinth Generierung

Das Spiel bietet eine dynamische Labyrinth-Generierung, bei der jedes neue Labyrinth zufällig erstellt wird. Jedes Labyrinth besteht aus einem Startpunkt und einem Zielpunkt, zwischen denen der Spieler die Kugel steuern muss. Darüber hinaus werden Wände und Löcher zufällig im Labyrinth platziert, was jede Spielrunde einzigartig macht. Ein wesentlicher Bestandteil des Algorithmus ist die Berechnung des kürzesten Pfades. Dies stellt sicher, dass das Labyrinth immer lösbar ist, sodass der Spieler immer eine Möglichkeit hat, das Ziel zu erreichen. Die Schwierigkeit des Labyrinths passt sich zudem dynamisch dem Fortschritt des Spielers an. Dies bedeutet, dass die Labyrinthe im Laufe des Spiels zunehmend schwieriger werden, um eine kontinuierliche Herausforderung zu gewährleisten. Der Algorithmus sorgt dafür, dass das Labyrinth in jedem Spieldurchlauf anders aussieht, wodurch das Spiel abwechslungsreich und spannend bleibt.

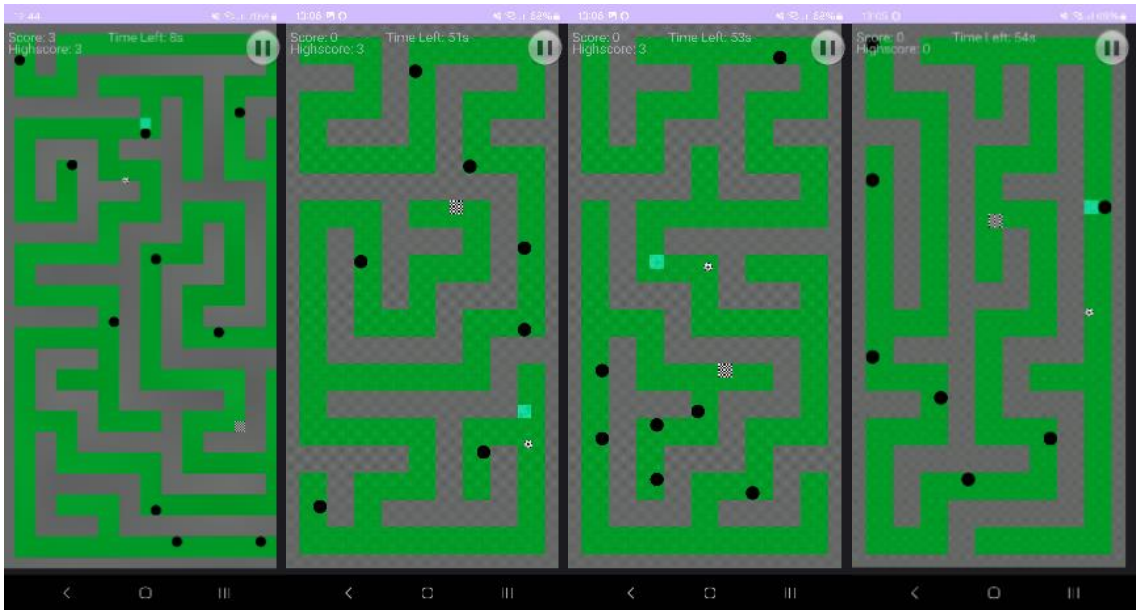

Abbildung 3 Labyrinthe

5.4 Bluetooth

Die Bluetooth-Implementierung des Spiels ermöglicht eine Highscore-Synchronisation zwischen mehreren Geräten. Der Ablauf dieser Synchronisation ist gut strukturiert und beinhaltet mehrere Schritte. Zunächst erfolgt eine Suche nach Bluetooth-fähigen Geräten in der Nähe. Sobald ein Gerät gefunden wurde, wird eine sichere Verbindung zwischen den Geräten hergestellt. Nachdem die Verbindung aufgebaut wurde, werden die Highscore-Daten ausgetauscht. Dies ermöglicht es den Spielern, ihre besten Ergebnisse mit anderen zu teilen. Der höchste Wert wird dabei in der Bestenliste gespeichert, sodass die Spieler ihre Leistungen mit anderen vergleichen können. Diese Implementierung nutzt das Android Bluetooth API, um die Datenpakete zwischen den Geräten auszutauschen und eine reibungslose Synchronisation der Highscores zu gewährleisten.

Abbildung 4 Bluetooth

6 Szenarien

6.1 Spiel spielen

6.1.1 Neues Spiel

Im Hauptmenü gibt der Spieler seinen Namen ein, wenn dieser noch nicht vorhanden ist. Das Spiel erstellt daraufhin einen neuen Spieler mit einem Highscore von 0. Der Spieler steuert dann die Kugel durch das Labyrinth. Das Ziel des Spiels ist es, so viele Labyrinthe wie möglich in 60 Sekunden zu lösen. Der Punktestand sowie der Highscore steigen mit jedem erfolgreich abgeschlossenen Labyrinth. Nach Abschluss des Spiels wird der Highscore gespeichert.

6.1.2 Spiel laden

Im Hauptmenü gibt der Spieler seinen Namen ein, der bereits im System gespeichert ist. Das Spiel lädt daraufhin den Highscore des Spielers aus dem **SharedPreferences** und zeigt diesen im Spiel an. Der Spieler steuert die Kugel durch das Labyrinth, und das Ziel bleibt, so viele Labyrinthe wie möglich in 60 Sekunden zu lösen. Der Punktestand steigt mit jedem erfolgreich abgeschlossenen Labyrinth. Wenn der Spieler einen neuen Highscore erreicht, wird dieser nach Abschluss des Spiels gespeichert.

6.1.3 Spiel abbrechen

Der Spieler gibt im Hauptmenü seinen Namen ein und drückt auf „Spiel starten“. Er steuert die Kugel durch das Labyrinth, mit dem Ziel, so viele Labyrinthe wie möglich in 60 Sekunden zu lösen. Der Punktestand steigt mit jedem erfolgreich abgeschlossenen Labyrinth. Wenn der Spieler das Spiel pausiert, kann er entweder auf „Neustart“ oder „Hauptmenü“ drücken. Der Highscore wird nach Abbruch des Spiels gespeichert.

6.2 Mehrspieler-Highscore-Synchronisation

6.2.1 Highscores von anderen laden

Um Highscores von anderen Spielern zu laden, muss der Spieler der App die Berechtigung der Bluetooth-Nutzung erteilen und sicherstellen, dass das Bluetooth auf seinem Gerät aktiviert ist. Es wird vorausgesetzt, dass die beiden Geräte bereits vorher miteinander über Bluetooth verbunden wurden. Im Hauptmenü drückt der Spieler auf die Bluetooth-Option und aktiviert die Sichtbarkeit seines Geräts. Nun wartet der Spieler auf einen anderen Spieler, dessen Gerät ebenfalls bereit ist. Sobald beide Geräte verbunden sind, werden die Daten synchronisiert und automatisch gespeichert.

6.2.2 Highscores an andere senden

Auch hier muss der Spieler der App die Berechtigung der Bluetooth-Nutzung erteilen und sicherstellen, dass das Bluetooth auf seinem Gerät aktiviert ist. Auch hier wird vorausgesetzt, dass die beiden Geräte bereits vorher miteinander über Bluetooth verbunden wurden. Der Spieler drückt im Hauptmenü auf die Bluetooth-Option und wählt das Gerät aus, das die Daten empfangen soll. Danach drückt der Spieler auf „Senden“, um die Highscore-Daten an das andere Gerät zu übertragen.

7 Zusammenfassung

Das Projekt Murmellabyrinth zeigt die gelungene Umsetzung eines mobilen Geschicklichkeitsspiels mit moderner Sensorik und drahtloser Synchronisation. Die Implementierung von dynamischen Labyrinthen, physikbasierter Steuerung und einer Bluetooth-Bestenliste stellt die Kernaspekte dar. Durch iterative Entwicklung und Testing wurden realistische Bewegungen und eine intuitive Spielerfahrung ermöglicht.

8 Projektverzeichnis

R. Hoppe, Sensorbasiertes Murmellabyrinth, GitHub Repository, 2025. [Online]. https://github.com/Cardostyle/Ballgame