

Aufgabe 1 – Projektvorschlag

Projektname

Memory Game – Android Mini-Game

Marvin Chow

Romina Mirmehdi

Projektbeschreibung

Im Rahmen dieses Projekts wird eine mobile Android-Anwendung mithilfe von **Android Studio** entwickelt. Bei der Anwendung handelt es sich um ein **Memory-Mini-Game**, bei dem der Nutzer verdeckte Karten antippt, um zusammengehörige Kartenpaare zu finden.

Um eine kreative und abwechslungsreiche Gestaltung in das Memory-Spiel einzubringen, wurde bewusst mit **unterschiedlichen Kartenmotiven** gearbeitet. Die Motive sind thematisch an die jeweiligen **Schwierigkeitsstufen** angepasst und unterstützen dadurch nicht nur die visuelle Abwechslung, sondern auch den steigenden Anspruch des Spiels.

Für die **leichte Schwierigkeitsstufe** wurden Kartenmotive mit **Coding-Sprüchen** gewählt, die thematisch zum Studiengang passen. Dadurch entsteht ein direkter Bezug zum Studium. Gleichzeitig sind die Motive gut erkennbar und erleichtern den Einstieg in das Spiel.

In der **mittleren Schwierigkeitsstufe** kommen **Obst-Motive** zum Einsatz. Diese sind visuell etwas vielfältiger und erfordern bereits mehr Aufmerksamkeit beim Merken der Kartenpaare, ohne den Spieler zu überfordern.

Die **schwere Schwierigkeitsstufe** basiert auf dem Thema „**Monster AG**“. Durch die detaillierteren und zahlreicher Motive steigt der Schwierigkeitsgrad deutlich an. Diese Stufe richtet sich an Spieler, die eine größere Herausforderung suchen und ihre Gedächtnisleistung weiter testen möchten.

Durch die thematische Zuordnung der Kartenmotive zu den einzelnen Schwierigkeitsstufen wird das Spiel nicht nur anspruchsvoller, sondern auch abwechslungsreicher und persönlicher gestaltet. Gleichzeitig zeigt dieses Konzept, wie gestalterische Entscheidungen sinnvoll mit der Spiellogik verbunden werden können.

Zu Beginn des Spiels kann der Nutzer eine **Schwierigkeitsstufe auswählen**. Es stehen die Stufen **einfach**, **mittel** und **schwer** zur Verfügung. Je höher die gewählte Schwierigkeitsstufe, desto mehr Kartenpaare werden auf dem Spielfeld angezeigt. Dadurch erhöht sich der Anspruch an Konzentration und Gedächtnisleistung.

Während des Spiels werden dem Nutzer wichtige **Spielinformationen übersichtlich im oberen Bereich des Bildschirms** angezeigt. Dazu zählen:

- die Anzahl der bisher benötigten **Versuche**,
- die Anzahl der gefundenen **Kartenpaare**,
- sowie die **vergangene Spielzeit** in Sekunden.

Durch Antippen einer Karte wird deren Vorderseite sichtbar. Sobald zwei Karten aufgedeckt sind, überprüft das System automatisch, ob diese ein passendes Paar bilden. Stimmen die Karten überein, bleiben sie dauerhaft sichtbar. Stimmen sie nicht überein, werden sie nach einer kurzen Verzögerung wieder verdeckt. Dieser Ablauf wiederholt sich, bis alle Kartenpaare gefunden wurden.

Nach erfolgreichem Abschluss des Spiels erhält der Nutzer eine deutliche Rückmeldung in Form einer „**Gewonnen**“-Meldung. Zusätzlich besteht die Möglichkeit, das Spiel über einen **Neustart-Button** erneut zu starten, um eine andere Schwierigkeitsstufe zu wählen oder die eigene Leistung zu verbessern.

Die Anwendung ist bewusst **übersichtlich und intuitiv** gestaltet, sodass sie problemlos auf mobilen Endgeräten bedient werden kann. Gleichzeitig werden zentrale Konzepte der mobilen Anwendungsentwicklung umgesetzt, darunter die Gestaltung von Benutzeroberflächen mit XML, die Verarbeitung von Benutzereingaben, die Umsetzung von Spiellogik sowie die Anzeige dynamischer Spielinformationen.

Der Funktionsumfang der Anwendung ist so gewählt, dass das Projekt **innerhalb eines Zeitraums von zwei Wochen realistisch umsetzbar** ist und sich gut für die praktische Anwendung sowie die Dokumentation der im Modul vermittelten Inhalte eignet.

Zielgruppe

- Studierende, die kurze und unterhaltsame Spiele auf dem Smartphone spielen möchten
- Nutzer von Casual Games, die ihr **Gedächtnis und ihre Konzentration** trainieren wollen
- Wegen den Bezug zu Coding-Sprüchen: Studierende im Bereich **Informatik / Medieninformatik**
- Gelegenheitsspieler, die ein **einfach bedienbares Spiel** für zwischendurch suchen
- Nutzer aller Altersgruppen, die Interesse an **Memory- und Denkspielen** haben

Grundfunktionen (Must-Have)

- Die App muss dem Nutzer ermöglichen, **eine Schwierigkeitsstufe** (leicht, mittel, schwer) auszuwählen.
- Die App muss je nach Schwierigkeitsstufe **eine unterschiedliche Anzahl an Kartenpaaren** anzeigen.
- Der Nutzer muss **Karten antippen können**, um sie aufzudecken.
- Die App muss **zwei aufgedeckte Karten automatisch vergleichen**.
- Passende Kartenpaare müssen **dauerhaft sichtbar bleiben**.
- Nicht passende Karten müssen **nach kurzer Verzögerung wieder verdeckt werden**.
- Die App muss erkennen, **wann alle Kartenpaare gefunden wurden**.
- Nach Spielende muss eine „**Gewonnen**“-Meldung angezeigt werden.

Erweiterte Funktionen (Should-Have)

- Die App soll die **Anzahl der Versuche** mitzählen und anzeigen.
- Die App soll die **Anzahl der gefundenen Kartenpaare** anzeigen.
- Die App soll die **vergangene Spielzeit** anzeigen.
- Die App soll dem Nutzer ermöglichen, das Spiel **neu zu starten**.
- Die Benutzeroberfläche soll **übersichtlich und intuitiv** gestaltet sein.

Optionale Funktionen (Could-Have)

- Die App kann verschiedene **Spielfeldergrößen** anbieten.
- Die App kann eine **Bestleistung (Score)** speichern.
- Die App kann visuelle Effekte (z. B. Animationen beim Aufdecken) enthalten.

Priorisierung der Anforderungen

- **Must:**
Grundlegende Spiellogik (Karten aufdecken und vergleichen) sowie das Erkennen des Spielendes
- **Should:**
Funktionen zur Verbesserung des Spielkomforts und der Übersicht, wie z. B. Anzeige von Versuchen, Spielzeit und gefundenen Paaren

- **Could:**
Zusätzliche Funktionen zur Motivation und Erweiterung des Spiels, z. B. Highscore, Animationen oder weitere Schwierigkeitsoptionen

Technischer Rahmen

- **Entwicklungsumgebung:** Android Studio
- **Programmiersprache:** Kotlin
- **Benutzeroberfläche:** Jetpack Compose (Kotlin)
- **Architektur:** Lokale Spiellogik ohne Server- oder Datenbankanbindung

Aufgabe2 – Dokumentation

Projektübersicht

Das Projekt ist ein Memory-Spiel in Android (Jetpack Compose). Es gibt:

- Modelle (Datenklassen und Enums)
- Logik für Spielstart und Kartenmischen
- UI-Screens und UI-Komponenten
- Animationen für Karten und Gewinn
- Theme/Farbwelt

1) Einstiegspunkt

- Datei: app/src/main/java/com/example/memory/MainActivity.kt
- Zweck: Startet die App und lädt die Oberfläche.
 - Ablauf: setContent() -> MemoryTheme -> Scaffold -> MemoryGameScreen.
 - Ergebnis: Die gesamte UI wird über Compose gerendert.

2) Modelle und Konfiguration

- Datei: app/src/main/java/com/example/memory/model/GameModels.kt
- MemoryCard: Datenklasse für eine Karte.
 - Felder: id, imageResId, isFaceUp, isMatched.
 - Bedeutung: Speichert Zustand jeder Karte (offen/zu, gematcht).
 - Difficulty (Enum): Schwierigkeitsstufen.
 - EASY/MEDIUM/HARD mit Bildlisten und Hintergrund.
 - pairCount gibt die Anzahl der Paare an.
 - GridConfig + gridConfigFor():
 - Berechnet Spaltenanzahl und Kartengröße passend zur Schwierigkeit.
 - DifficultyChoice:
 - Struktur für die Startseite (Titel/Untertitel/Schwierigkeit), um UI übersichtlich zu halten.

3) Spiellogik

- Datei: app/src/main/java/com/example/memory/game/GameLogic.kt
- createDeck(difficulty):
 - Wählt zufällige Bilder, dupliziert sie zu Paaren, mischt sie und erstellt MemoryCard-Objekte.
 - Ergebnis: Ein fertiges, gemischtes Kartendeck.

4) Hauptscreen und Spielablauf

- Datei: app/src/main/java/com/example/memory/ui/screen/MemoryGameScreen.kt
- Zustände (State):
 - difficulty, cards, attempts, matchedPairs, isChecking, startTime, elapsedSeconds.
 - wobbleTokens: zählt Fehlversuche pro Karte (für Wobble-Animation).
 - resetGame():
 - Zentrale Reset-Funktion, setzt Karten, Zähler und Timer zurück.
 - Timer:
 - LaunchedEffect aktualisiert elapsedSeconds jede Sekunde.
 - chooseDifficulty():
 - Setzt Schwierigkeit und startet neues Spiel.

- restart():
 - Startet Spiel mit aktueller Schwierigkeit neu.
- flipCard(index):
 - Öffnet eine Karte, prüft zwei offene Karten, markiert Match oder dreht zurück.
 - Bei Fehlversuch wird wobbleTokens erhöht (Animation).
- UI-Aufbau:
 - Titel "Memory" oben.
 - Wenn keine Schwierigkeit gewählt: DifficultySelection.
 - Wenn Spiel aktiv: Statusleiste + Karten-Grid.
 - WinOverlay als Overlay bei Sieg.

5) Startseite / UX

Datei: app/src/main/java/com/example/memory/ui/components/DifficultySelection.kt

- Hero-Card mit kurzem Erklärungstext:
 - Text: "Trainiere dein Gedächtnis in 60 Sekunden..."
 - Drei Auswahlkarten (Leicht/Mittel/Schwer).
 - Große Touch-Flächen für gute Bedienbarkeit.
 - Professionelle Farbwelt, nicht kindisch.

6) Karten-UI + Animationen

Datei: app/src/main/java/com/example/memory/ui/components/MemoryCardView.kt

- Flip-Animation:
 - rotationY von 0° auf 180° (animateFloatAsState).
- Match-Pop:
 - Wenn isMatched true, skaliert Karte kurz (1.0 -> 1.06 -> 1.0).
- Fehlversuch-Wobble:
 - Kurzes horizontales Wackeln (Animatable + keyframes).
- Kartenrückseite:
 - Leichter Farbverlauf + optionales Hintergrundbild.
- Rahmen und Schatten:
 - Card mit border und elevation für haptisches Gefühl.

7) Gewinn-Overlay + Glow

Datei: app/src/main/java/com/example/memory/ui/components/WinOverlay.kt

- Overlay legt dunklen Layer über das Spielfeld.
- Gewinndialog erscheint mit Fade + Scale.
- Glow-Effekt:
 - Animiertes, leichtes Leuchten hinter der Gewinnkarte.
- CTA-Buttons:
 - "Noch eine Runde" und "Schwierigkeit ändern".

8) Theme & Farben

Dateien:

- app/src/main/java/com/example/memory/ui/theme/Color.kt
- app/src/main/java/com/example/memory/ui/theme/Theme.kt
- app/src/main/java/com/example/memory/ui/theme/Type.kt

Farbkonzept:

- Hintergrund: professionelles, leicht gebrochenes Weiß (Paper).
- Primärfarbe: Accent (für Buttons und Highlights).
- Sekundärfarbe: Accent2 (für Statuswerte).
- Ziel: erwachsen, ruhig, klarer Kontrast.

9) Vertiefung: wichtige Logikstellen und warum sie so programmiert sind

A) MemoryGameScreen – Zustandsschutz, Timer und Paarprüfung

Datei: app/src/main/java/com/example/memory/ui/screen/MemoryGameScreen.kt

Codeausschnitt 1 – konsistenter Reset:

```
fun resetGame(selected: Difficulty) {  
    cards = createDeck(selected)  
    attempts = 0  
    matchedPairs = 0  
    isChecking = false  
    startTime = System.currentTimeMillis()  
    elapsedSeconds = 0  
    wobbleTokens = emptyMap()  
}  
...  
...
```

Erklärung:

- Der Reset bündelt alle Zustandsänderungen an einer Stelle (Single Source of Truth).
- Dadurch werden Reset-Fehler vermieden, z. B. wenn Timer oder Zähler vergessen würden.

Codeausschnitt 2 – Timer als Side-Effect:

```
...  
  
LaunchedEffect(startTime, isGameOver) {  
    if (!isGameOver) {  
        while (true) {  
            delay(1000)  
            elapsedSeconds = (System.currentTimeMillis() - startTime) / 1000  
            if (matchedPairs == totalPairs) break  
        }  
    }  
}
```

Erklärung:

- Der Timer ist ein Side-Effect (Coroutine), der automatisch an den UI-State gebunden ist.
- Durch die Schlüssel (startTime, isGameOver) startet/stoppt er korrekt bei Neustart und Spielende.

Codeausschnitt 3 – Paarprüfung mit Sperre:

```

```
if (isChecking) return

```
if (faceUp.size == 2) {
    attempts++
    isChecking = true
    scope.launch {
        delay(800)
        if (first.value.imageResId == second.value.imageResId) {
            ... isMatched = true ...
        } else {
            wobbleTokens = wobbleTokens.toMutableMap().also { ... }
            ... isFaceUp = false ...
        }
        isChecking = false
    }
}
```

```

Erklärung:

- isChecking verhindert, dass der Spieler während der Prüfung weitere Karten öffnet.
- delay(800) sorgt für eine kurze Sichtzeit der zweiten Karte (UX-Feedback).
- wobbleTokens ist ein Trigger für die Wobble-Animation und hält die Logik der Animation vom UI getrennt.

B) MemoryCardView – Flip, Pop und Wobble technisch erklärt

Datei: app/src/main/java/com/example/memory/ui/components/MemoryCardView.kt

Codeausschnitt 1 – 3D-Flip:

...

```

val rotation by animateFloatAsState(
 targetValue = if (card.isFaceUp || card.isMatched) 180f else 0f,
 animationSpec = tween(durationMillis = 350),
 label = "cardFlip"
)

```

```
)
val showFront = rotation > 90f
...
...
```

Erklärung:

- animateFloatAsState koppelt die Rotation direkt an den Karten-State.
- showFront entscheidet, wann die Vorderseite sichtbar ist (nach 90°).

Codeausschnitt 2 – Pop bei Treffer:

```
...
...
val popScale = remember { Animatable(1f) }
...
popScale.animateTo(1.06f, animationSpec = tween(120))
popScale.animateTo(1f, animationSpec = tween(160))
...
...
```

Erklärung:

- Animatable erlaubt kontrollierte Sequenzen (hoch, dann zurück).
- Das ist eine bewusste UX-Bestätigung für ein korrektes Paar.

Codeausschnitt 3 – Wobble bei Fehlversuch:

```
...
...
wobbleOffset.animateTo(
 targetValue = 0f,
 animationSpec = keyframes {
 durationMillis = 320
 0f at 0
 -amplitudePx at 60
 amplitudePx at 120
 ...
 }
)
...
```

Erklärung:

- Keyframes geben einen natürlichen "Wackel"-Rhythmus vor.
- amplitudePx basiert auf dp-Werten, damit es auf allen Geräten gleich wirkt.

Codeausschnitt 4 – Kombination in graphicsLayer:

```
...
.graphicsLayer {
 rotationY = rotation
 cameraDistance = 12 * density
 translationX = wobbleOffset.value
 scalex = popScale.value
 scaley = popScale.value
}
...
```

Erklärung:

- Alle Effekte werden in einer Transformationsschicht gebündelt.
- Dadurch bleiben Layout und Logik getrennt, während die Animationen sauber zusammenspielen.

C) WinOverlay – Overlay, Glow und Übergang

Datei: app/src/main/java/com/example/memory/ui/components/WinOverlay.kt

Codeausschnitt 1 – Glow-Animation:

```
...
val winglow = remember { Animatable(0f) }

...
winglow.animateTo(1f, animationSpec = tween(250))
winglow.animateTo(0f, animationSpec = tween(700))
...
```

Erklärung:

- Der Glow läuft nur bei Gewinn (LaunchedEffect über isGameOver).
- Zwei Stufen (hoch/runter) wirken wie ein kurzer Erfolgs-Impuls.

Codeausschnitt 2 – Sichtbarkeit und Übergang:

```
...
AnimatedVisibility(
 visible = isGameOver,
 enter = fadeIn(...) + scaleIn(...),
 exit = fadeOut(...) + scaleOut(...)
)
...
```

Erklärung:

- AnimatedVisibility verbindet Logik (isGameOver) direkt mit UI-Transition.
- Fade + Scale macht den Gewinn-Dialog klar wahrnehmbar, aber nicht aufdringlich.

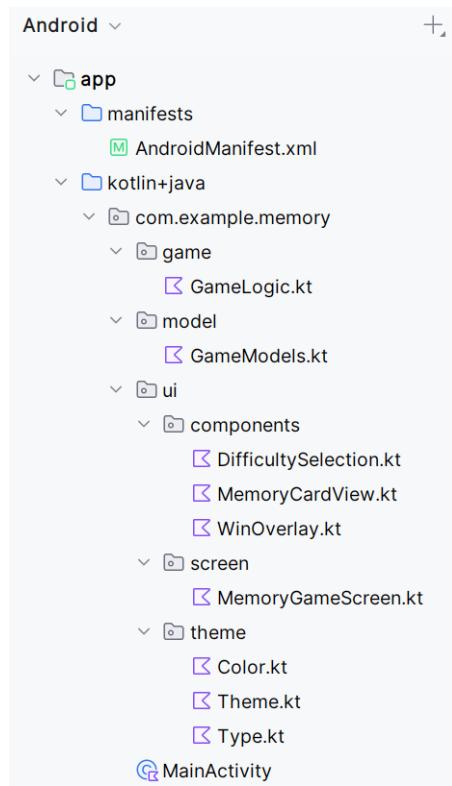
Codeausschnitt 3 – Glow-Layer hinter der Karte:

```
...
.graphicsLayer {
 alpha = winGlow.value * 0.35f
 scalex = 1.03f
 scaleY = 1.03f
}
...
```

Erklärung:

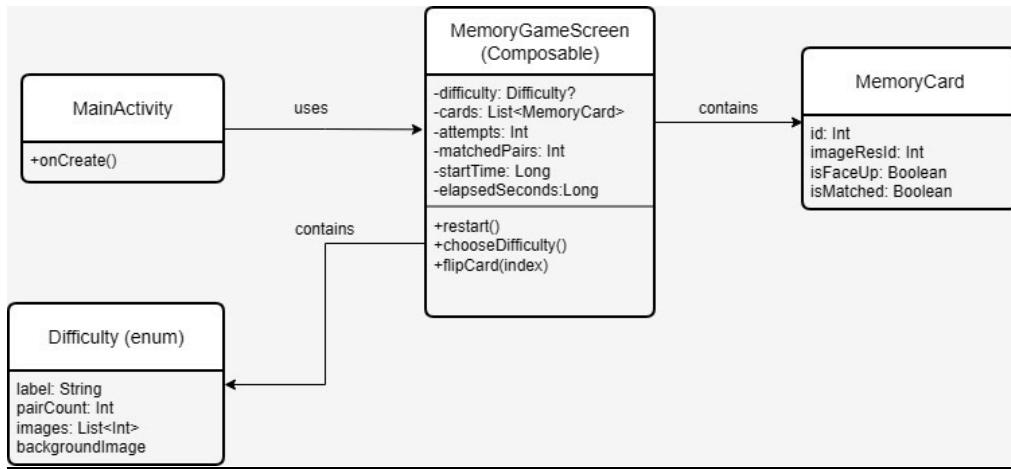
- Der Glow ist nur eine zusätzliche, leicht vergrößerte Fläche hinter der Card.
- So bleibt die Karte selbst klar lesbar und der Effekt wirkt professionell.

## Ordnerstruktur



## Klassendiagramm

[https://drive.google.com/file/d/1NpSan0vi\\_dtCtwNinVEbQOtvdm2K5Xbh/view](https://drive.google.com/file/d/1NpSan0vi_dtCtwNinVEbQOtvdm2K5Xbh/view)



## Beschreibung des Bedienungsablaufs

1. App starten → Titel Memory erscheint
2. Schwierigkeit wählen (Leicht / Mittel / Schwer)
3. Spielbeginn
4. Karten liegen verdeckt im Grid
5. Karte antippen
6. Zwei Karten werden aufgedeckt
7. Bei Gleichheit → Paar bleibt offen
8. Bei Unterschied → Karten drehen sich zurück
9. Spielstatus
10. Anzeige von Versuchen, gefundenen Paaren und Zeit
11. Spielende
12. Alle Paare gefunden → Meldung Gewonnen!
13. Neustart oder neue Schwierigkeit wählen

Bilder

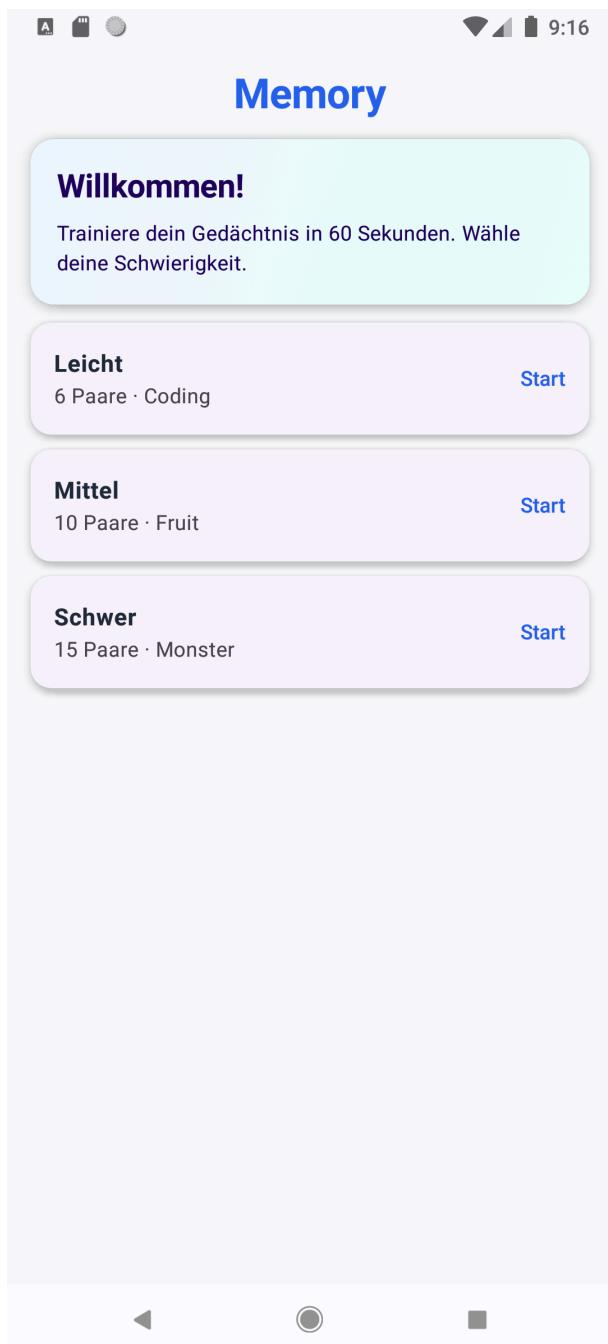
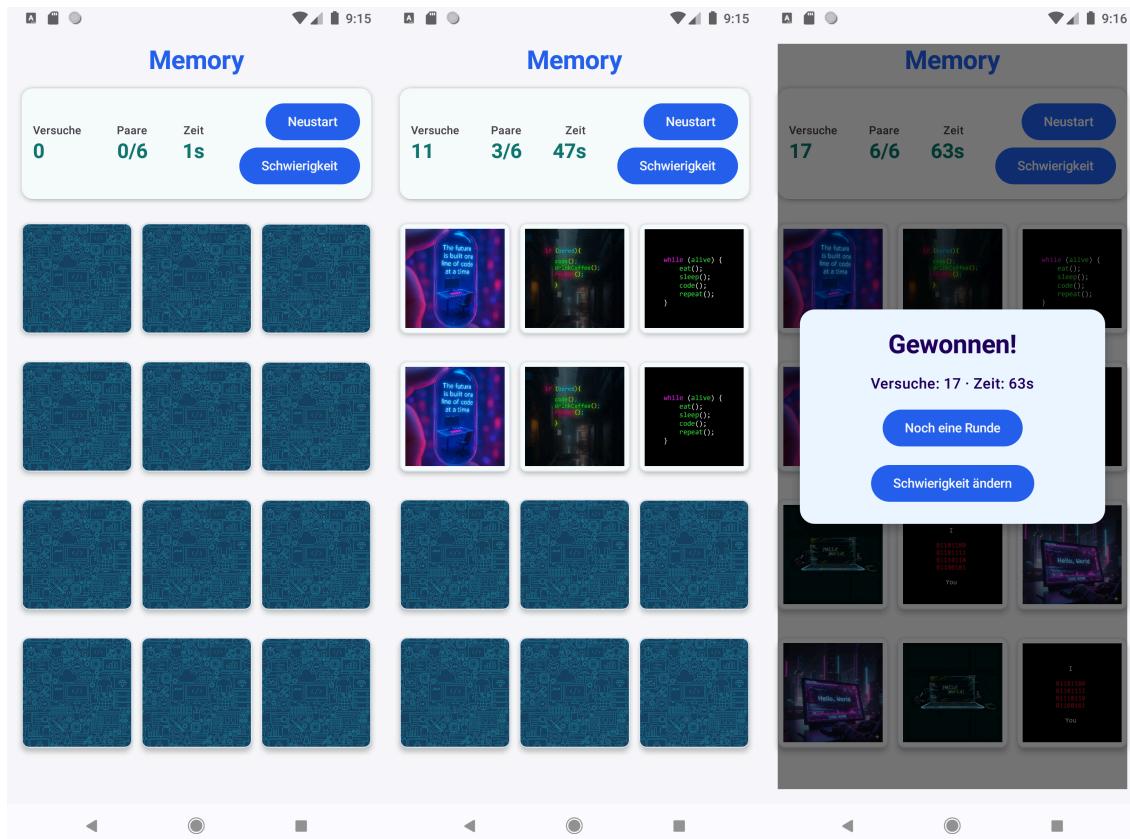
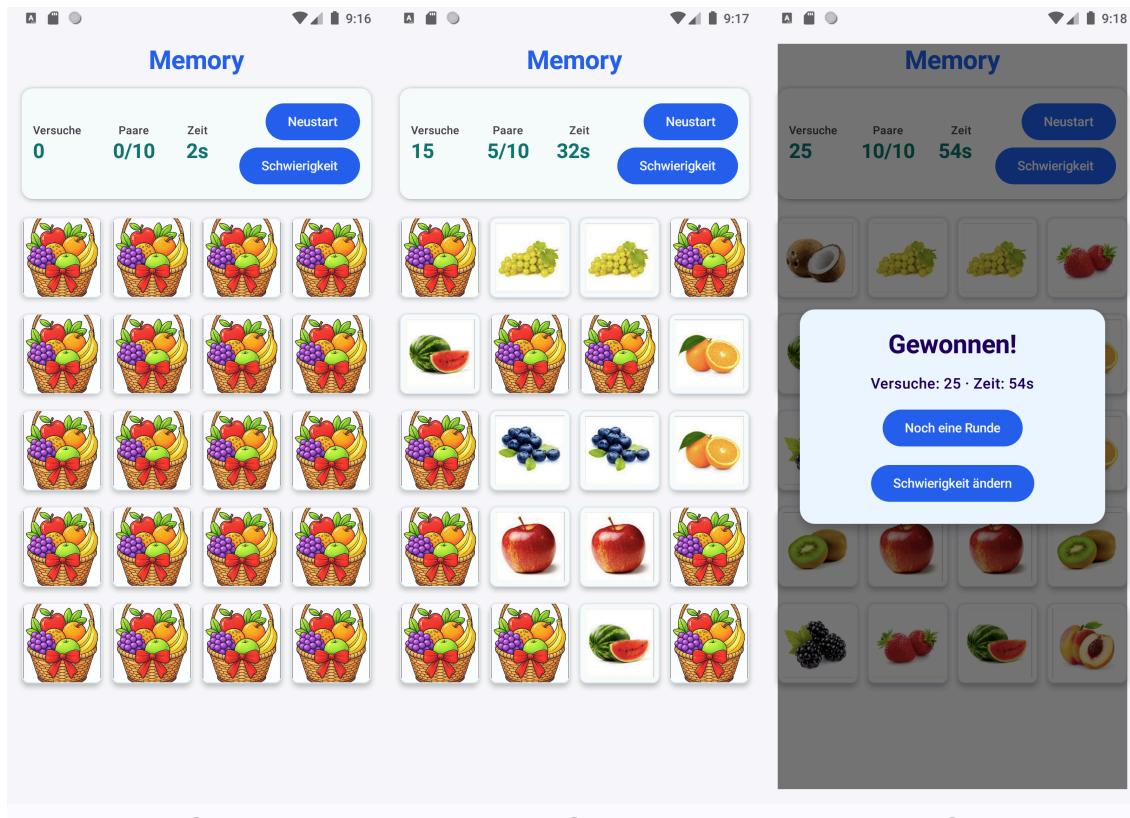


Bild1: Startseite



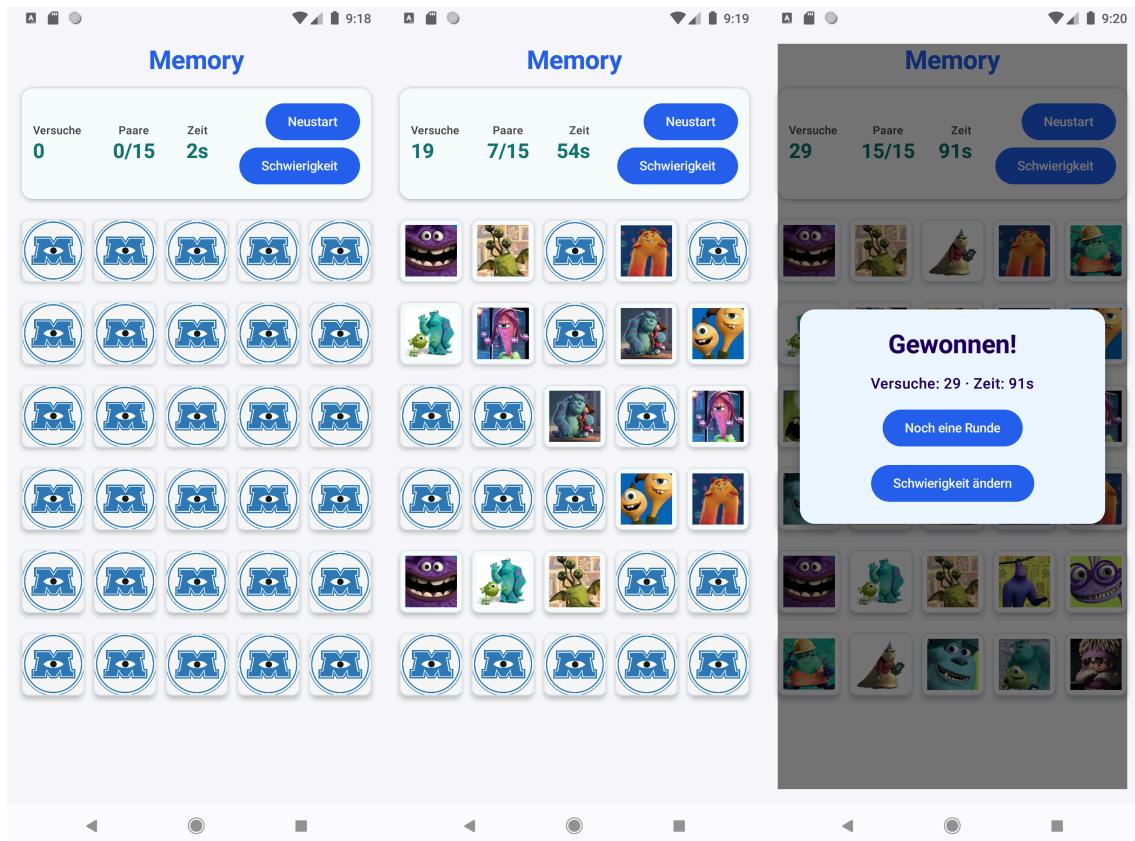
Drei Beispielbilder zur Veranschaulichung des Schwierigkeitsgrades "leicht":

1. Bild eins: Vollständig abgedeckt, alle Karten verdeckt.
2. Bild zwei: drei Paare aufgedeckt, der Rest verdeckt.
3. Letztes Bild: Alle Karten aufgedeckt, Sieg.



Drei Beispielbilder zur Veranschaulichung des Schwierigkeitsgrades "Mittel":

1. Bild eins: Das gesamte Spielfeld ist vollständig abgedeckt, alle Karten sind verdeckt.
2. Im zweiten Bild sind die Hälfte der Kartenpaare aufgedeckt, während der verbleibende Teil des Spielfeldes verdeckt bleibt.
3. Letztes Bild: Alle Karten sind aufgedeckt, was den Sieg anzeigen.



Fünf Beispielbilder zur Veranschaulichung des Schwierigkeitsgrades "Scha":

1. Bild eins: Das gesamte Spielfeld ist vollständig abgedeckt, alle Karten sind verdeckt.
2. Bild zwei: Ein Paar aufgedeckt, der Rest des Spielfeldes ist verdeckt.
3. Letztes Bild: Alle Karten sind aufgedeckt, was den Sieg anzeigen.

## Auflistung, wer in der Gruppe was gemacht hat:

### Romina

- Ausarbeitung des **Projektkonzepts** (Memory-Mini-Game)
- Erstellung der **Projektbeschreibung** und Zielgruppenbeschreibung
- Formulierung der **funktionalen Anforderungen** und deren Priorisierung
- Dokumentation des **Spielablaufs** und der Schwierigkeitsstufen
- Erstellung und Pflege der **PDF-Dokumentation**
- Strukturierung und sprachliche Überarbeitung des gesamten Protokolls

### Marvin

- Technische Umsetzung des Projekts in **Android Studio**
- Erstellung der **Benutzeroberfläche** mit XML-Layouts
- Implementierung der **Schwierigkeitsstufen** (leicht, mittel, schwer)
- Einbindung der **Kartenmotive** (Coding-Sprüche, Obst, Monster AG)
- Testen der Anwendung und Fehlerbehebung

## Aufteilung der Programmierarbeiten

### Romina (Code-Anteil)

- Implementierung der **Spielstatistiken** (Versuche, gefundene Paare, Spielzeit)
- Logik für die **Anzeige der Spielinformationen** im oberen Bildschirmbereich
- Umsetzung der **Gewonnen-Meldung** und Neustart-Funktion
- Unterstützung bei der Spiellogik sowie Kommentierung des eigenen Codes

### Marvin (Code-Anteil)

- Implementierung der **grundlegenden Spiellogik** (Karten aufdecken, Vergleich der Karten)
- Verwaltung des **Spielzustands** (aufgedeckt, verdeckt, gefunden)
- Dynamische Erstellung des Spielfelds je nach Schwierigkeitsstufe
- Organisation der Kartenstruktur und Spielinitialisierung

## Dokumentation des erreichten Wissensstandes und der Lernerfolge

| Bereich              | Wissensstand vor dem Projekt                        | Wissensstand nach dem Projekt / Lernerfolge                                       |
|----------------------|-----------------------------------------------------|-----------------------------------------------------------------------------------|
| Android Studio       | Grundlegende Kenntnisse, einfache Projekte          | Sicherer Umgang mit Android Studio und Projektstruktur                            |
| Projektplanung       | Wenig Erfahrung in der Planung einer kompletten App | Strukturierte Planung mit Projektbeschreibung, Anforderungen und Priorisierung    |
| UI-Gestaltung        | Einfache Layouts bekannt                            | Umsetzung übersichtlicher Benutzeroberflächen mit XML-Layouts                     |
| Ereignisverarbeitung | Grundlegende Klick-Events                           | Sicherer Umgang mit Benutzerinteraktionen (Karten antippen, Buttons, Spielablauf) |
| Spiellogik           | Kaum Erfahrung mit Spielmechaniken                  | Umsetzung einer vollständigen Memory-Spiellogik (Vergleich, Spielende, Neustart)  |
| Zustandsverwaltung   | Begrenztes Verständnis                              | Verwaltung von Spielzuständen (aufgedeckt, gefunden, Spiel beendet)               |
| Dynamische Inhalte   | Wenig Erfahrung                                     | Anzeige dynamischer Spielinformationen (Versuche, Zeit, Paare)                    |

| Bereich              | Wissensstand vor dem Projekt         | Wissensstand nach dem Projekt / Lernerfolge                               |
|----------------------|--------------------------------------|---------------------------------------------------------------------------|
| Schwierigkeitsstufen | Keine praktische Umsetzung           | Umsetzung mehrerer Schwierigkeitsstufen mit unterschiedlicher Kartenzahl  |
| Teamarbeit           | Grundlegende Gruppenarbeit           | Klare Aufgabenverteilung, Zusammenarbeit am Code und an der Dokumentation |
| Dokumentation        | Unsicher im strukturierten Schreiben | Strukturierte und verständliche Projektdokumentation für eine Abgabe      |

### Ausblick

Aus der Lehrveranstaltung nehme ich mehrere Ideen und Ansätze für zukünftige Projekte mit. Besonders wichtig ist für mich die Erkenntnis, Anwendungen von Beginn an strukturiert zu planen und schrittweise umzusetzen. Die im Projekt erlernten Konzepte können gut als Grundlage für weiterführende Anwendungen genutzt werden, zum Beispiel durch die Erweiterung um zusätzliche Funktionen, komplexere Logiken oder eine dauerhafte Datenspeicherung.

Darüber hinaus besteht Interesse, sich künftig intensiver mit fortgeschrittenen Themen der mobilen Entwicklung zu beschäftigen, wie etwa der Nutzung von Datenbanken, Animationen, moderner App-Architektur oder der Entwicklung größerer, modular aufgebauter Anwendungen. Die in der Lehrveranstaltung gesammelten Erfahrungen stellen hierfür eine solide Basis dar und können auch in zukünftigen Studienprojekten sinnvoll eingesetzt werden.

### Eigenanteil und Quellcodeangaben

Der Quellcode für dieses Projekt wurde vollständig von uns selbst programmiert mit Hilfe von KI und Recherche. Wir sind mit einer eigenen Projektidee gestartet und haben bewusst auf vorgefertigte Beispielprojekte verzichtet. Ziel war es, unsere vorhandenen Kenntnisse zu festigen, aufzufrischen und durch praktische Umsetzung zu erweitern.

Zur Unterstützung während der Entwicklung wurde KI als Hilfsmittel genutzt, insbesondere zur Klärung von Verständnisfragen und zur Überprüfung von Lösungsansätzen. Der Code wurde dabei nicht unreflektiert übernommen, sondern verstanden, angepasst und eigenständig in das Projekt integriert.

Alle zentralen Bestandteile der Anwendung, einschließlich Spiellogik, Benutzeroberfläche und Spielstatistiken, wurden von uns selbst implementiert. Da keine externen Beispielcodes übernommen wurden, sind keine zusätzlichen Quellenangaben erforderlich.