종합 게임 앱 프로젝트 계획서

🎯 프로젝트 개요

목표: 하나의 계정으로 여러 미니게임을 즐기고 기록을 경쟁하는 종합 게임 플랫폼

첫 번째 게임: 총알 피하기 게임 (Bullet Dodge)

📋 전체 프로젝트 단계

Phase 0: 프로젝트 초기 설계 (1-2일)

```
목표: 아키텍처 설계 및 기술 스택 결정
```

0.1 아키텍처 설계

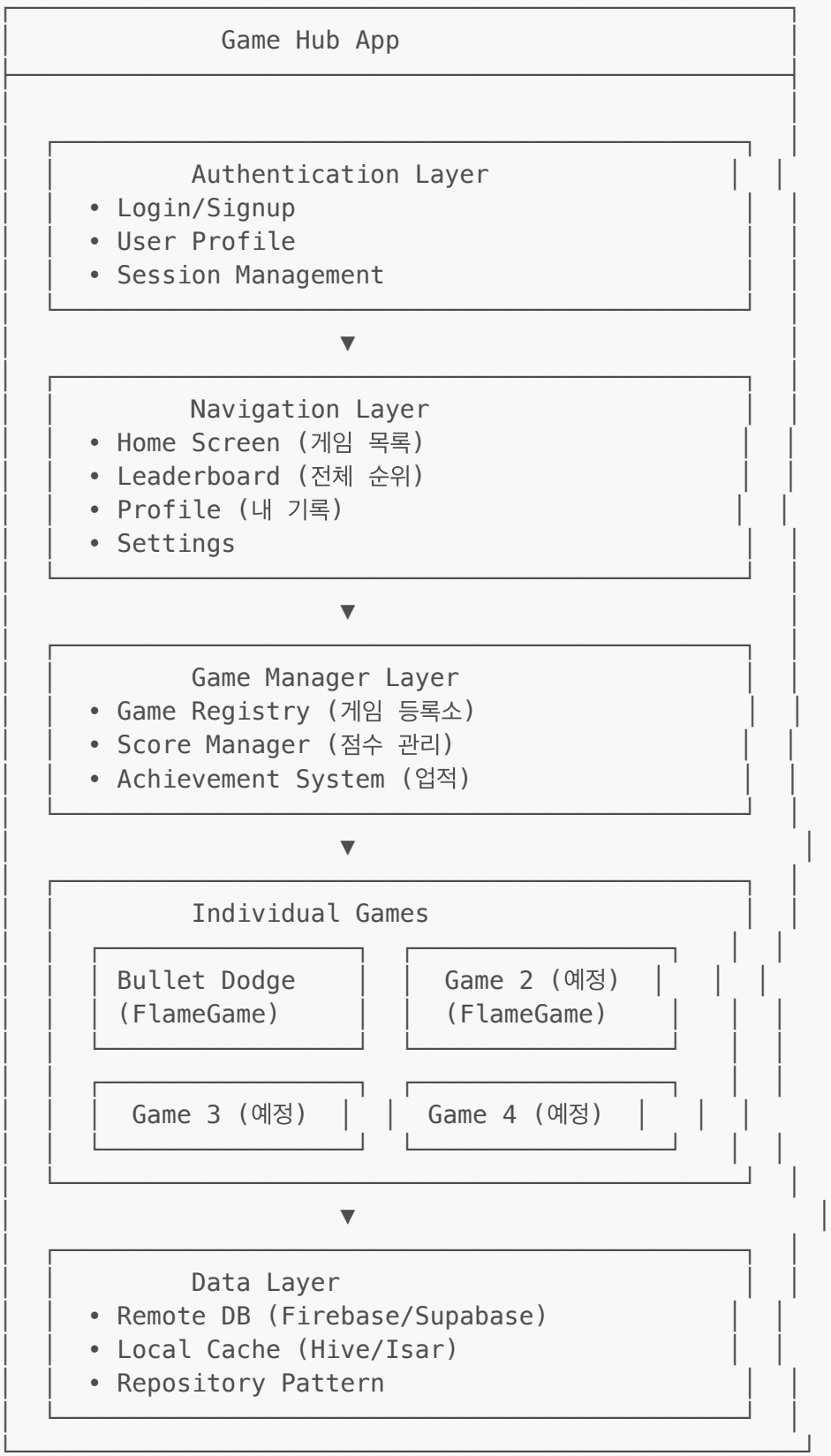
- [ ] 앱 전체 구조 설계
- [ ] 데이터베이스 스키마 설계
- [ ] 게임 추가 확장성 고려
- [ ] 상태 관리 방식 결정

0.2 기술 스택 선정

```
Frontend:
  ✓ Flutter (UI 프레임워크)
  ✓ Flame (게임 엔진)
  ✓ Riverpod (상태 관리)
  ✓ go_router (라우팅)

Backend/Database: => 서버는 싱글플레이 선행 작업 후 도입
  - Firebase (추천)
    • Authentication (계정 관리)
    • Firestore (데이터 저장)
    • Cloud Functions (서버 로직)

  또는

  - Supabase (오픈소스 대안)
    • PostgreSQL 기반
    • 실시간 기능
    • Row Level Security

Local Storage:
  - shared_preferences (설정)
  - hive/isar (로컬 캐시)
```

🏗️ 아키텍처 설계

전체 앱 구조



폴더 구조 (Clean Architecture)

```
lib/
├── core/
│   ├── constants/           # 상수, 설정값
│   ├── theme/               # 앱 테마, 스타일
│   ├── utils/               # 유틸리티 함수
│   ├── errors/              # 에러 처리
│   └── network/             # API 클라이언트
│
├── features/
│   ├── auth/
│   │   ├── data/
│   │   │   ├── models/
│   │   │   ├── repositories/
│   │   │   └── data_sources/
│   │   ├── domain/
│   │   │   ├── entities/
│   │   │   ├── repositories/
│   │   │   └── usecases/
│   │   └── presentation/
│   │       ├── screens/
│   │       ├── widgets/
│   │       └── providers/
│   │
│   ├── home/                # 게임 목록 화면
│   │   ├── data/
│   │   ├── domain/
│   │   └── presentation/
│   │
│   ├── leaderboard/         # 순위표
│   │   ├── data/
│   │   ├── domain/
│   │   └── presentation/
│   │
│   ├── profile/             # 프로필
│   │   ├── data/
│   │   ├── domain/
│   │   └── presentation/
│   │
│   └── games/               # 개별 게임들
│       ├── shared/          # 게임 공통 요소
│       │   ├── base_game.dart
│       │   ├── game_score_manager.dart
│       │   └── game_ui_overlay.dart
│       │
│       ├── bullet_dodge/    # 총알 피하기 게임
│       │   ├── data/
│       │   │   ├── models/
│       │   │   └── repositories/
│       │   ├── domain/
│       │   │   ├── entities/
```

```
│   │   │   └── usecases/
│   │   └── presentation/
│   │       ├── game/
│   │       │   ├── bullet_dodge_game.dart
│   │       │   ├── components/
│   │       │   │   ├── player.dart
│   │       │   │   ├── bullet.dart
│   │       │   │   ├── bullet_spawner.dart
│   │       │   │   └── world.dart
│   │       │   └── systems/
│   │       │       ├── collision_system.dart
│   │       │       ├── difficulty_system.dart
│   │       │       └── score_system.dart
│   │       ├── screens/
│   │       │   ├── game_screen.dart
│   │       │   └── game_over_screen.dart
│   │       └── widgets/
│   │           ├── game_hud.dart
│   │           └── pause_menu.dart
│   │
│   └── game_2/          # 두 번째 게임 (미래)
│       └── ...
│
├── shared/
│   ├── widgets/         # 공통 위젯
│   ├── models/          # 공통 모델
│   └── providers/       # 공통 프로바이더
│
└── main.dart
```

🗄️ 데이터베이스 스키마

Users (사용자)

```
{
  id: string,              // UUID
  email: string,           // 이메일
  username: string,        // 닉네임
  avatar_url?: string,     // 프로필 이미지
  created_at: timestamp,
  updated_at: timestamp,
  total_score: number,     // 전체 게임 총점
  level: number,           // 사용자 레벨
  exp: number             // 경험치
}
```

Games (게임 정보)

```
{
  id: string,              // 게임 ID
  name: string,            // 게임 이름
  description: string,     // 설명
  thumbnail_url: string,   // 썸네일
  is_active: boolean,      // 활성화 여부
  min_version: string,     // 최소 앱 버전
  created_at: timestamp
}
```

Scores (게임 기록)

```
{
  id: string,
  user_id: string,         // FK: Users
  game_id: string,         // FK: Games
  score: number,           // 점수
  rank: number,            // 순위
  play_time: number,       // 플레이 시간 (초)
  metadata: json,          // 게임별 추가 정보
  created_at: timestamp
}
```

Leaderboards (순위표)

```
{
  game_id: string,
  user_id: string,
  username: string,
  best_score: number,      // 최고 점수
  total_plays: number,     // 플레이 횟수
  average_score: number,   // 평균 점수
  last_played: timestamp,
  rank: number             // 전체 순위
}
```

Achievements (업적)

```
{
  id: string,
  user_id: string,
  game_id: string,
  achievement_type: string, // 업적 종류
  unlocked_at: timestamp,
  metadata: json
}
```

📅 구현 단계별 상세 계획

Phase 1: 프로젝트 기반 구축 (3-4일)

1.1 프로젝트 셋업

```
# 의존성 추가
flutter pub add riverpod flutter_riverpod
flutter pub add go_router
flutter pub add firebase_core firebase_auth cloud_firestore
flutter pub add shared_preferences
flutter pub add freezed_annotation json_annotation
flutter pub add google_fonts

# Dev 의존성
flutter pub add --dev build_runner freezed json_serializable
```

1.2 폴더 구조 생성

- [ ] Clean Architecture 폴더 구조 생성
- [ ] 각 feature별 기본 파일 생성
- [ ] 공통 모듈 셋업

1.3 Firebase/Supabase 설정

- [ ] 프로젝트 생성
- [ ] 인증 설정
- [ ] 데이터베이스 스키마 생성
- [ ] 보안 규칙 설정

1.4 기본 테마 및 상수 설정

```
// lib/core/theme/app_theme.dart
class AppTheme {
  static ThemeData get darkTheme => ThemeData.dark().copyWith(
    colorScheme: ColorScheme.dark(
      primary: Color(0xFF00D9FF),
      secondary: Color(0xFFFF006E),
    ),
  );
}

// lib/core/constants/game_constants.dart
class GameConstants {
  static const double gameWidth = 360;
```

```
  static const double gameHeight = 640;
  static const int targetFPS = 60;
}
```

Phase 2: 인증 시스템 구현 (2-3일)

2.1 Auth Feature 구현

- [ ] 로그인 화면 UI
- [ ] 회원가입 화면 UI
- [ ] Firebase Auth 연동
- [ ] 이메일/비밀번호 인증
- [ ] Google 소셜 로그인 (선택)
- [ ] 세션 관리
- [ ] 에러 처리

2.2 사용자 프로필 초기 설정

- [ ] 닉네임 설정
- [ ] 프로필 이미지 (기본값)
- [ ] Firestore에 사용자 정보 저장

```
// lib/features/auth/domain/entities/user.dart
@freezed
class User with _$User {
  const factory User({
    required String id,
    required String email,
    required String username,
    String? avatarUrl,
    @Default(0) int totalScore,
    @Default(1) int level,
    @Default(0) int exp,
  }) = _User;
}
```

Phase 3: 네비게이션 및 홈 화면 (2-3일)

3.1 라우팅 설정 (go_router)

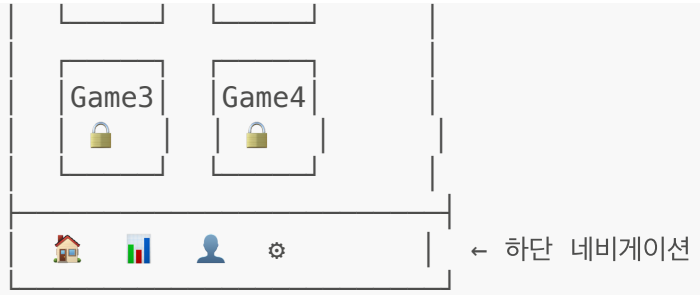
```
// lib/core/router/app_router.dart
final appRouter = GoRouter(
  initialLocation: '/splash',
  routes: [
    GoRoute(
```

```
      path: '/splash',
      builder: (context, state) => SplashScreen(),
    ),
    GoRoute(
      path: '/auth',
      builder: (context, state) => AuthScreen(),
    ),
    GoRoute(
      path: '/home',
      builder: (context, state) => HomeScreen(),
      routes: [
        GoRoute(
          path: 'game/:gameId',
          builder: (context, state) {
            final gameId = state.pathParameters['gameId']!;
            return GameScreen(gameId: gameId);
          },
        ),
      ],
    ),
    GoRoute(
      path: '/leaderboard',
      builder: (context, state) => LeaderboardScreen(),
    ),
    GoRoute(
      path: '/profile',
      builder: (context, state) => ProfileScreen(),
    ),
  ],
);
```

3.2 홈 화면 구현

- [ ] 게임 목록 그리드/리스트
- [ ] 각 게임 카드 UI
- [ ] 잠금/해금 상태 표시
- [ ] 베스트 스코어 표시
- [ ] 하단 네비게이션 바

```
홈 화면 구성:

┌─────────────────────────┐
│  Game Hub  🏆  👤       │  ← 상단 바
├─────────────────────────┤
│  Your Best: 15,230 pts  │  ← 사용자 정보
│  Level 5  ▓▓▓▓▓▓▓▓ 67%  │
├─────────────────────────┤
│                         │
│  ┌──────┐  ┌──────┐     │
│  │Game1 │  │Game2 │     │  ← 게임 목록
│  │  🎯  │  │  🔒  │     │
│  │1,234 │  │ Soon │     │
```



Phase 4: 게임 공통 시스템 (2-3일)

4.1 BaseGame 추상 클래스

```dart
// lib/features/games/shared/base_game.dart
abstract class BaseGameHub extends FlameGame {
  final String gameId;
  final GameScoreManager scoreManager;

  BaseGameHub({
    required this.gameId,
    required this.scoreManager,
  });

  // 공통 메서드
  void pauseGame();
  void resumeGame();
  void gameOver(int finalScore);
  void saveScore();

  // 추상 메서드 (각 게임이 구현)
  void setupGame();
  void resetGame();
}
```

4.2 점수 관리 시스템

```dart
// lib/features/games/shared/game_score_manager.dart
class GameScoreManager {
  int currentScore = 0;
  int highScore = 0;
  int combo = 0;

  void addScore(int points) {
    currentScore += points * (1 + combo * 0.1);
  }

  void increaseCombo() {
```

```
    combo++;
  }

  void resetCombo() {
    combo = 0;
  }

  Future<void> saveToFirestore() async {
    // Firestore 저장 로직
  }
}
```

4.3 공통 UI 오버레이

- [ ] 일시정지 메뉴
- [ ] 게임 오버 화면
- [ ] HUD (점수, 시간, 콤보)
- [ ] 카운트다운

Phase 5: 총알 피하기 게임 구현 (5-7일)

5.1 게임 기획 상세

```
게임명: Bullet Dodge (총알 피하기)

목표:
  - 위에서 떨어지는 총알을 피해 최대한 오래 살아남기
  - 시간이 지날수록 총알 속도와 개수 증가

조작:
  - 좌우 드래그/터치로 플레이어 이동
  - 화면 경계를 넘어갈 수 없음

점수 시스템:
  - 생존 시간: 1초당 10점
  - 근접 회피: 총알과 아슬아슬하게 피하면 보너스 (+50점)
  - 콤보: 연속 근접 회피시 점수 배율 증가

난이도:
  - 0-30초: 총알 2-3개, 느린 속도
  - 30-60초: 총알 4-5개, 중간 속도
  - 60-90초: 총알 6-8개, 빠른 속도
  - 90초+: 총알 10개+, 매우 빠른 속도

게임 종료:
  - 총알에 맞으면 즉시 게임 오버
  - 최종 점수 표시 및 순위 확인
```

5.2 게임 컴포넌트 구현

Player Component

```
// lib/features/games/bullet_dodge/presentation/game/components/player.dart
class Player extends SpriteComponent
    with HasGameRef, CollisionCallbacks, DragCallbacks {

  static const double speed = 300.0;
  static const double size = 40.0;

  Vector2 velocity = Vector2.zero();
  bool isInvulnerable = false;

  @override
  Future<void> onLoad() async {
    sprite = await Sprite.load('player.png');
    size = Vector2.all(size);
    anchor = Anchor.center;
    position = Vector2(gameRef.size.x / 2, gameRef.size.y - 100);

    add(CircleHitbox());
  }

  @override
  void update(double dt) {
    super.update(dt);

    // 위치 업데이트
    position += velocity * dt;

    // 경계 체크
    position.x = position.x.clamp(size.x / 2, gameRef.size.x - size.x /
2);
  }

  @override
  void onDragUpdate(DragUpdateEvent event) {
    velocity.x = event.delta.x / event.dt;
  }

  @override
  void onCollision(Set<Vector2> points, PositionComponent other) {
    if (other is Bullet && !isInvulnerable) {
      gameRef.gameOver();
    }
  }

  void takeDamage() {
    if (!isInvulnerable) {
      // 게임 오버 처리
```

```
      gameRef.gameOver();
    }
  }
}
```

Bullet Component

```
//
lib/features/games/bullet_dodge/presentation/game/components/bullet.dart
class Bullet extends SpriteComponent
    with HasGameRef, CollisionCallbacks {

  final double speed;
  final double radius;

  Bullet({
    required Vector2 position,
    this.speed = 200.0,
    this.radius = 15.0,
  }) : super(position: position);

  @override
  Future<void> onLoad() async {
    sprite = await Sprite.load('bullet.png');
    size = Vector2.all(radius * 2);
    anchor = Anchor.center;

    add(CircleHitbox());
  }

  @override
  void update(double dt) {
    super.update(dt);

    // 아래로 이동
    position.y += speed * dt;

    // 화면 밖으로 나가면 제거
    if (position.y > gameRef.size.y + radius) {
      removeFromParent();
    }
  }

  void checkNearMiss(Player player) {
    final distance = position.distanceTo(player.position);
    if (distance < 50 && distance > 30) {
      // 근접 회피 보너스
      gameRef.scoreManager.addNearMissBonus();
    }
  }
}
```

Bullet Spawner System

```
//
lib/features/games/bullet_dodge/presentation/game/components/bullet_spawne
r.dart
class BulletSpawner extends Component with HasGameRef {

  double spawnTimer = 0;
  double spawnInterval = 1.5; // 초기 1.5초마다 생성

  final DifficultySystem difficultySystem;

  BulletSpawner(this.difficultySystem);

  @override
  void update(double dt) {
    super.update(dt);

    spawnTimer += dt;

    if (spawnTimer >= spawnInterval) {
      spawnBullet();
      spawnTimer = 0;

      // 난이도에 따라 간격 조정
      spawnInterval = difficultySystem.getBulletSpawnInterval();
    }
  }

  void spawnBullet() {
    final random = Random();
    final x = random.nextDouble() * gameRef.size.x;
    final speed = difficultySystem.getBulletSpeed();

    final bullet = Bullet(
      position: Vector2(x, -20),
      speed: speed,
    );

    gameRef.world.add(bullet);
  }

  void spawnMultipleBullets(int count) {
    for (int i = 0; i < count; i++) {
      Future.delayed(Duration(milliseconds: i * 100), () {
        spawnBullet();
      });
    }
  }
}
```

5.3 게임 시스템 구현

Difficulty System

```dart
// lib/features/games/bullet_dodge/presentation/game/systems/difficulty_system.dart
class DifficultySystem {
  double gameTime = 0;

  void update(double dt) {
    gameTime += dt;
  }

  int getDifficultyLevel() {
    if (gameTime < 30) return 1;
    if (gameTime < 60) return 2;
    if (gameTime < 90) return 3;
    return 4;
  }

  double getBulletSpeed() {
    switch (getDifficultyLevel()) {
      case 1: return 150.0;
      case 2: return 250.0;
      case 3: return 350.0;
      case 4: return 450.0;
      default: return 200.0;
    }
  }

  double getBulletSpawnInterval() {
    switch (getDifficultyLevel()) {
      case 1: return 1.5;
      case 2: return 1.0;
      case 3: return 0.7;
      case 4: return 0.5;
      default: return 1.5;
    }
  }

  int getBulletsPerSpawn() {
    switch (getDifficultyLevel()) {
      case 1: return 1;
      case 2: return 2;
      case 3: return 2;
      case 4: return 3;
      default: return 1;
    }
  }
}
```

Score System

```
//
lib/features/games/bullet_dodge/presentation/game/systems/score_system.dar
t
class ScoreSystem {
  int score = 0;
  int combo = 0;
  int nearMissCount = 0;
  double survivalTime = 0;

  void update(double dt) {
    survivalTime += dt;
    // 1초당 10점
    score = (survivalTime * 10).toInt();
  }

  void addNearMissBonus() {
    combo++;
    nearMissCount++;
    final bonus = 50 * (1 + combo * 0.2);
    score += bonus.toInt();
  }

  void resetCombo() {
    combo = 0;
  }

  Map<String, dynamic> getGameStats() {
    return {
      'score': score,
      'survival_time': survivalTime,
      'near_miss_count': nearMissCount,
      'max_combo': combo,
    };
  }
}
```

Collision System

```
//
lib/features/games/bullet_dodge/presentation/game/systems/collision_system
.dart
class CollisionSystem extends Component with HasGameRef {

  final Player player;
  final List<Bullet> bullets;
```

```
    CollisionSystem({
      required this.player,
      required this.bullets,
    });

    @override
    void update(double dt) {
      super.update(dt);

      for (final bullet in bullets) {
        checkCollision(player, bullet);
        checkNearMiss(player, bullet);
      }
    }

    void checkCollision(Player player, Bullet bullet) {
      final distance = player.position.distanceTo(bullet.position);
      final minDistance = (player.size.x / 2) + (bullet.size.x / 2);

      if (distance < minDistance) {
        player.takeDamage();
      }
    }

    void checkNearMiss(Player player, Bullet bullet) {
      final distance = player.position.distanceTo(bullet.position);

      // 아슬아슬하게 피한 경우 (30-60 픽셀 사이)
      if (distance > 30 && distance < 60 &&
          bullet.position.y > player.position.y) {
        gameRef.scoreSystem.addNearMissBonus();
      }
    }
  }
```

5.4 Main Game Class

```
  //
  lib/features/games/bullet_dodge/presentation/game/bullet_dodge_game.dart
  class BulletDodgeGame extends BaseGameHub with HasCollisionDetection {

    late Player player;
    late BulletSpawner bulletSpawner;
    late DifficultySystem difficultySystem;
    late ScoreSystem scoreSystem;

    BulletDodgeGame() : super(
      gameId: 'bullet_dodge',
      scoreManager: GameScoreManager(),
    );
```

```
  @override
  Future<void> onLoad() async {
    await super.onLoad();

    // 시스템 초기화
    difficultySystem = DifficultySystem();
    scoreSystem = ScoreSystem();

    // 배경 추가
    world.add(Background());

    // 플레이어 추가
    player = Player();
    world.add(player);

    // 총알 생성 시스템
    bulletSpawner = BulletSpawner(difficultySystem);
    world.add(bulletSpawner);

    // UI 오버레이
    overlays.add('game_hud');

    setupGame();
  }

  @override
  void update(double dt) {
    super.update(dt);

    if (!paused) {
      difficultySystem.update(dt);
      scoreSystem.update(dt);
    }
  }

  @override
  void setupGame() {
    // 게임 초기 설정
    camera.viewport = FixedResolutionViewport(
      resolution: Vector2(360, 640),
    );
  }

  @override
  void pauseGame() {
    pauseEngine();
    overlays.add('pause_menu');
  }

  @override
  void resumeGame() {
    resumeEngine();
    overlays.remove('pause_menu');
```

```
  }

  @override
  void gameOver(int finalScore) {
    pauseEngine();
    saveScore();
    overlays.add('game_over');
  }

  @override
  Future<void> saveScore() async {
    final stats = scoreSystem.getGameStats();
    await scoreManager.saveToFirestore(
      userId: currentUserId,
      gameId: gameId,
      score: scoreSystem.score,
      metadata: stats,
    );
  }

  @override
  void resetGame() {
    // 게임 리셋
    scoreSystem = ScoreSystem();
    difficultySystem = DifficultySystem();

    // 모든 총알 제거
    world.children.whereType<Bullet>().forEach((bullet) {
      bullet.removeFromParent();
    });

    // 플레이어 위치 초기화
    player.position = Vector2(size.x / 2, size.y - 100);

    resumeGame();
  }
}
```

5.5 UI 오버레이

Game HUD

```
// lib/features/games/bullet_dodge/presentation/widgets/game_hud.dart
class GameHUD extends StatelessWidget {
  final BulletDodgeGame game;

  @override
  Widget build(BuildContext context) {
    return SafeArea(
      child: Padding(
        padding: EdgeInsets.all(16),
```

```
          child: Column(
            children: [
              // 상단: 점수, 시간
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text(
                        'Score',
                        style: TextStyle(fontSize: 14, color:
Colors.white70),
                      ),
                      Text(
                        '${game.scoreSystem.score}',
                        style: TextStyle(
                          fontSize: 32,
                          fontWeight: FontWeight.bold,
                          color: Colors.white,
                        ),
                      ),
                    ],
                  ),
                  Column(
                    crossAxisAlignment: CrossAxisAlignment.end,
                    children: [
                      Text(
                        'Time',
                        style: TextStyle(fontSize: 14, color:
Colors.white70),
                      ),
                      Text(

'${game.scoreSystem.survivalTime.toStringAsFixed(1)}s',
                        style: TextStyle(
                          fontSize: 24,
                          fontWeight: FontWeight.bold,
                          color: Colors.white,
                        ),
                      ),
                    ],
                  ),
                ],
              ),

              SizedBox(height: 8),

              // 콤보 표시
              if (game.scoreSystem.combo > 0)
                Container(
                  padding: EdgeInsets.symmetric(horizontal: 16, vertical:
8),
                  decoration: BoxDecoration(
```

```
                  color: Colors.orange,
                  borderRadius: BorderRadius.circular(20),
                ),
                child: Text(
                  'Combo x${game.scoreSystem.combo}',
                  style: TextStyle(
                    fontSize: 18,
                    fontWeight: FontWeight.bold,
                    color: Colors.white,
                  ),
                ),
              ),

              Spacer(),

              // 일시정지 버튼
              Align(
                alignment: Alignment.topRight,
                child: IconButton(
                  icon: Icon(Icons.pause, color: Colors.white, size: 32),
                  onPressed: () => game.pauseGame(),
                ),
              ),
            ],
          ),
        ),
      );
  }
}
```

Game Over Screen

```
//
lib/features/games/bullet_dodge/presentation/screens/game_over_screen.dart
class GameOverScreen extends StatelessWidget {
  final BulletDodgeGame game;

  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.black87,
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              'Game Over',
              style: TextStyle(
                fontSize: 48,
                fontWeight: FontWeight.bold,
                color: Colors.red,
```

```
              ),
            ),

            SizedBox(height: 40),

            // 최종 점수
            Container(
              padding: EdgeInsets.all(20),
              decoration: BoxDecoration(
                color: Colors.white10,
                borderRadius: BorderRadius.circular(16),
              ),
              child: Column(
                children: [
                  Text(
                    'Final Score',
                    style: TextStyle(fontSize: 18, color: Colors.white70),
                  ),
                  SizedBox(height: 8),
                  Text(
                    '${game.scoreSystem.score}',
                    style: TextStyle(
                      fontSize: 64,
                      fontWeight: FontWeight.bold,
                      color: Colors.amber,
                    ),
                  ),
                  SizedBox(height: 20),
                  Row(
                    mainAxisAlignment: MainAxisAlignment.spaceAround,
                    children: [
                      _StatItem(
                        label: 'Time',
                        value:
'${game.scoreSystem.survivalTime.toStringAsFixed(1)}s',
                      ),
                      _StatItem(
                        label: 'Near Miss',
                        value: '${game.scoreSystem.nearMissCount}',
                      ),
                      _StatItem(
                        label: 'Best Combo',
                        value: 'x${game.scoreSystem.combo}',
                      ),
                    ],
                  ),
                ],
              ),
            ),

            SizedBox(height: 40),

            // 버튼들
            Row(
```

```
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  ElevatedButton.icon(
                    icon: Icon(Icons.replay),
                    label: Text('Retry'),
                    onPressed: () {
                      game.resetGame();
                      game.overlays.remove('game_over');
                    },
                    style: ElevatedButton.styleFrom(
                      padding: EdgeInsets.symmetric(horizontal: 32,
vertical: 16),
                    ),
                  ),

                  SizedBox(width: 20),

                  ElevatedButton.icon(
                    icon: Icon(Icons.home),
                    label: Text('Home'),
                    onPressed: () {
                      context.go('/home');
                    },
                    style: ElevatedButton.styleFrom(
                      padding: EdgeInsets.symmetric(horizontal: 32,
vertical: 16),
                    ),
                  ),
                ],
              ),

              SizedBox(height: 20),

              // 순위 확인 버튼
              TextButton(
                child: Text('View Leaderboard'),
                onPressed: () {
                  context.push('/leaderboard?game=bullet_dodge');
                },
              ),
            ],
          ),
        ),
      );
  }
}

class _StatItem extends StatelessWidget {
  final String label;
  final String value;

  const _StatItem({required this.label, required this.value});

  @override
```

```
  Widget build(BuildContext context) {
    return Column(
      children: [
        Text(
          label,
          style: TextStyle(fontSize: 14, color: Colors.white70),
        ),
        SizedBox(height: 4),
        Text(
          value,
          style: TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.bold,
            color: Colors.white,
          ),
        ),
      ],
    );
  }
}
```

Phase 6: 리더보드 및 프로필 (2-3일)

6.1 리더보드 화면

```
// lib/features/leaderboard/presentation/screens/leaderboard_screen.dart
class LeaderboardScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Leaderboard')),
      body: Column(
        children: [
          // 게임 선택 탭
          GameSelector(),

          // 기간 선택 (오늘, 이번주, 전체)
          PeriodSelector(),

          // 순위 리스트
          Expanded(
            child: LeaderboardList(),
          ),
        ],
      ),
    );
  }
}
```

6.2 프로필 화면

```dart
// lib/features/profile/presentation/screens/profile_screen.dart
class ProfileScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Profile')),
      body: SingleChildScrollView(
        child: Column(
          children: [
            // 사용자 정보
            UserInfoCard(),

            // 전체 통계
            OverallStatsCard(),

            // 게임별 기록
            GameRecordsSection(),

            // 업적
            AchievementsSection(),
          ],
        ),
      ),
    );
  }
}
```

Phase 7: 테스트 및 최적화 (2-3일)

7.1 테스트

- [ ] 단위 테스트 (Business Logic)
- [ ] 위젯 테스트 (UI Components)
- [ ] 통합 테스트 (전체 플로우)
- [ ] 게임 밸런스 테스트
- [ ] 성능 테스트 (FPS, 메모리)

7.2 최적화

- [ ] 게임 성능 최적화
- [ ] 네트워크 요청 최적화
- [ ] 이미지/에셋 최적화
- [ ] 빌드 사이즈 최적화

7.3 버그 수정 및 폴리싱

- [ ] 버그 수정
- [ ] 애니메이션 추가
- [ ] 사운드 효과 (선택)
- [ ] 햅틱 피드백

Phase 8: 배포 준비 (1-2일)

8.1 스토어 준비

- [ ] 앱 아이콘 디자인
- [ ] 스크린샷 준비
- [ ] 스토어 설명 작성
- [ ] 개인정보 처리방침

8.2 릴리즈 빌드

- [ ] Android 빌드
- [ ] iOS 빌드 (선택)
- [ ] 베타 테스트 (Firebase App Distribution)

🎨 에셋 및 리소스 준비

필요한 에셋

```
assets/
├── images/
│   ├── backgrounds/
│   │   └── game_bg.png
│   ├── player/
│   │   ├── player.png
│   │   └── player_hit.png
│   ├── bullets/
│   │   ├── bullet_red.png
│   │   ├── bullet_blue.png
│   │   └── bullet_explosion.png
│   ├── ui/
│   │   ├── button_play.png
│   │   ├── button_pause.png
│   │   └── icons/
│   └── effects/
│       ├── particle.png
│       └── shield.png
│
├── audio/ (선택사항)
│   ├── music/
│   │   └── game_bgm.mp3
│   └── sfx/
│       ├── bullet_fire.mp3
```

```
│       ├── explosion.mp3
│       └── near_miss.mp3
│
└── fonts/
    └── game_font.ttf
```

에셋 제작 도구

- **이미지**: Figma, Adobe Illustrator, Aseprite
- **무료 에셋**: itch.io, OpenGameArt.org
- **사운드**: Freesound.org, Bfxr (효과음 생성기)

📊 개발 일정 요약

Phase	내용	예상 기간	우선순위
0	프로젝트 설계	1-2일	필수
1	프로젝트 셋업	3-4일	필수
2	인증 시스템	2-3일	필수
3	네비게이션/홈	2-3일	필수
4	게임 공통 시스템	2-3일	필수
5	총알 피하기 게임	5-7일	필수
6	리더보드/프로필	2-3일	높음
7	테스트/최적화	2-3일	높음
8	배포 준비	1-2일	중간
총계		**20-30일**	

🚀 MVP (Minimum Viable Product) 범위

첫 버전에 포함할 기능

✅ 필수 기능

- [ ] 이메일 로그인/회원가입
- [ ] 총알 피하기 게임 1개
- [ ] 점수 저장
- [ ] 간단한 리더보드 (전체 순위)
- [ ] 기본 프로필 화면

이후 버전에 추가할 기능

➡️ Phase 2 기능

- [ ] 소셜 로그인 (Google, Apple)
- [ ] 친구 시스템
- [ ] 업적 시스템
- [ ] 일일 챌린지
- [ ] 게임 2, 3 추가

🔮 향후 기능

- [ ] 아이템 시스템 (스킨, 파워업)
- [ ] 시즌/이벤트
- [ ] PvP 대전
- [ ] 리플레이 시스템
- [ ] 공유 기능 (스크린샷, 점수 공유)

💡 개발 팁 및 주의사항

1. 확장성 고려

```
// 나쁜 예
if (gameId == 'bullet_dodge') {
  // 게임별 로직 하드코딩
}

// 좋은 예
abstract class BaseGame {
  void initialize();
  void play();
}

class GameRegistry {
  Map<String, BaseGame Function()> games = {
    'bullet_dodge': () => BulletDodgeGame(),
    'game_2': () => Game2(),
  };
}
```

2. 성능 최적화

- Flame의 `onGameResize` 주의 (불필요한 재생성 방지)
- 오브젝트 풀링 사용 (총알 재사용)
- `removeFromParent()` 확실히 호출

3. 상태 관리

- 게임 상태는 Flame 내부에서 관리
- UI 상태는 Riverpod/Bloc으로 관리
- 명확한 경계 유지

4. 데이터 동기화

- 낙관적 업데이트 (Optimistic Update)
- 로컬 캐시 활용
- 오프라인 모드 고려

5. 보안

- 점수 검증 (서버 사이드)
- 치트 방지 (난독화, 서버 검증)
- API 키 보안 (환경변수)

📚 참고 자료

공식 문서

- Flame 공식 문서
- Flutter 공식 문서
- Firebase 문서

튜토리얼

- Flame 게임 예제: https://github.com/flame-engine/flame/tree/main/examples
- Flutter 게임 튜토리얼: https://docs.flutter.dev/cookbook/games

커뮤니티

- Flame Discord: https://discord.gg/pxrBmy4
- Flutter 한국 커뮤니티: https://flutter-ko.dev

🎯 다음 단계

1. **이 계획서 검토 및 조정**
    - 일정 조정
    - 우선순위 재설정
    - 기술 스택 최종 확정
2. **개발 환경 셋업**
    - Flutter/Flame 설치 확인
    - IDE 플러그인 설치
    - Firebase 프로젝트 생성
3. **폴더 구조 생성**

```
mkdir -p lib/{core,features,shared}
mkdir -p lib/core/{constants,theme,utils,network}
mkdir -p lib/features/{auth,home,leaderboard,profile,games}
```

4. **첫 번째 기능 개발 시작**
    - Phase 1: 프로젝트 셋업부터 시작

이제 시작할 준비가 되었습니다! 🚀

어느 단계부터 시작하시겠습니까?