FlappyBird (학습) — 메인기획서

> 재구성 기획서 (역설계) / 작성 기준 시점: 2021-03-29 (프로젝트 시작 시점) / 문서 버전: Ver 0.1 프로젝트 ID: P036 / 근거 수준: C / 작성: Claude (재구성) 이 문서는 결과물과 제작 기록으로부터 기획 의도를 역추정해 당시 작성한 것처럼 재구성한 것이다. 확인된 사실과 추정을 [근거]/[추정] 태그로 구분한다.

1. 핵심 메카닉스

새 점프/회전

- 의도(Why): 탭 한 번에 즉각적이고 일정한 상승으로 조작 난이도를 낮춘다.
- 설계(How): `jump()` 호출 시 플래그를 세우고, `FixedUpdate`에서 속도를 `Vector2.up * jumpSpeed`(씬 값 8)로 덮어쓰고 기울기 `rotationAngle`(30도)을 준다. 진행 중 매 물리 프레임 z 회전을 1.5 감소시켜 점차 고개를 숙인다. 가속 누적 대신 속도 덮어쓰기를 택해 연타해도 상승량이 일정하다. Rigidbody2D 중력 배율 3.5. [근거: BirdController.cs, FlappyBird.unity]
- 필요(What): Rigidbody2D, Collider2D, 태그 `Player`.
- 작업 방법: PlayMaker가 입력을 받아 `jump()` 호출(FlappyBirdJump 액션) → 에디터 플레이로 높이 체감 확인. [근거: FlappyBirdJump.cs 존재]
- 완료 기준: 탭마다 같은 높이로 상승, 낙하 시 고개가 숙는다.
- 리스크/대안: 회전 하한이 없어 낙하 중 과회전 가능(코드에 클램프 없음). 클램프 추가 가능.
- 근거: [근거: BirdController.cs]

시작 대기와 게임 시작

- 의도(Why): 첫 탭 전에는 새가 떨어지지 않아 플레이어가 준비할 시간을 준다.
- 설계(How): `Start`에서 중력을 저장 후 `Vector2.zero`로 설정, `isRunning=false`. 첫 `jump()`가 `GameController.jump()`를 통해 `startGame()`을 불러 `isRunning=true`, 중력 복원. [근거: GameController.cs]
- 필요(What): 정적 플래그 `isRunning`, `isFinished`.
- 작업 방법: FSM의 시작 상태에서 `FlappyBirdStartGame` 액션 호출. [추정: 액션 이름]
- 완료 기준: 첫 탭 전 정지, 이후 정상 낙하.
- 리스크/대안: `Physics2D.gravity`는 전역이라 재로드 시 0으로 남을 수 있다. 재시작은 씬 재로드로 매번 Start가 다시 돈다.
- 근거: [근거: GameController.cs]

파이프 스폰과 점진 가속

- 의도(Why): 규칙 추가 없이 시간이 갈수록 어려워지는 곡선을 만든다.
- 설계(How): `isRunning` 중 타이머가 `timeDelay`(초기 1초)를 넘으면 `startPosition`에서 Y에 `Random.Range(pipeMinY=-1, pipeMaxY=3)` 정수 오프셋을 더해 파이프를 `Instantiate`. 스폰마다 `speed += speedPace(5)`, `timeDelay -= timePace(0.01)`. 파이프는 `moveSpeed`(기본 5)로 좌측

이동하며 `Destroyer` 트리거에서 파괴된다. [근거: GameController.cs, PipeController.cs, FlappyBird.unity]
- 필요(What): `pipes` 프리팹, `Destroyer` 오브젝트, `startPosition`.
- 작업 방법: 프리팹 생성 → 스폰 간격 플레이 테스트.
- 완료 기준: 파이프가 끊기지 않고 나타나고 화면 밖에서 제거된다.
- 리스크/대안: `speed`는 증가하나 파이프 이동에 적용되는 코드가 없어 가속이 실제로는 스폰 간격 단축뿐일 수 있다(미확인). 파이프는 사망 시 `isKinematic`으로 정지.
- 근거: [근거: GameController.cs, PipeController.cs]

충돌/사망

- 의도(Why): 장애물 접촉 즉시 판정해 규칙을 명확히 한다.
- 설계(How): 새의 `OnTriggerEnter2D`/`OnCollisionEnter2D`에서 태그에 `Obsticles`(철자 그대로)가 포함되면 `stopGame` 호출. `isRunning=false`, `isFinished=true`, `DiedBird` 델리게이트 호출. 충돌 시 새는 kinematic으로 고정된다. [근거: BirdController.cs, GameController.cs]
- 필요(What): 파이프/바닥에 `Obsticles` 태그.
- 작업 방법: 사망 이벤트 `Bird Died`를 FSM이 수신해 게임오버 UI로 전환. [근거: FlappyBirdDiedChecker.cs]
- 완료 기준: 접촉 즉시 정지, 이벤트 1회 발생.
- 리스크/대안: 트리거와 콜리전이 둘 다 있어 이중 호출 가능. 중복 방지 플래그 필요.
- 근거: [근거: BirdController.cs]

점수

- 의도(Why): 파이프 통과를 명확한 보상으로 만든다.
- 설계(How): 파이프 내부 트리거에 `ScoreController`가 있어 `Player` 진입 시 한 번만(`isScored`) `updateScore` 호출. `score++`, `PlayerPrefs "Last Score"` 저장, `UpdatedScore` 델리게이트로 FSM에 `Score Updated` 전달. 최고점은 FSM이 `UpdateHighScore(int)`로 설정하고 `High Score` 키에 저장. [근거: ScoreController.cs, GameController.cs, FlappyBirdScoreUpdater.cs]
- 필요(What): 통과 트리거 콜라이더, 점수 UI.
- 작업 방법: FSM에서 점수 갱신 이벤트로 텍스트 갱신.
- 완료 기준: 통과당 정확히 +1, 재실행 후 최고점 유지.
- 리스크/대안: 최고점 비교 코드는 주석 처리돼 있어 비교는 FSM에 위임된 것으로 추정(FSM 미확인).
- 근거: [근거: GameController.cs]

2. 구현 방법

- 구조: C# 컴포넌트(물리, 스폰, 루프)와 PlayMaker FSM(흐름/UI)의 2층. 연결은 `GameController`의 델리게이트 `UpdatedScore`, `DiedBird`를 `FlappyBird*Checker/Updater`가 받아 `PlayMakerFSM`에 이벤트 전송. [근거: PlayMakerExtension 폴더]
- 상태 머신(추정): 대기 → 플레이 → 게임오버 → 재시작. 실제 FSM 내용은 미확인. [추정: 이벤트 이름 `Bird Died`, `Score Updated`]

- 스크롤: 바닥은 Rigidbody2D 속도 `groundSpeed(5)`로 좌측 이동, `Destroyer` 접촉 시 `startPosition`으로 순간이동해 무한 루프. 사망 시 속도 0. 배경은 트리거 시 위치 리셋만 있고 속도 코드는 주석 처리. [근거: GroundController.cs, BackgroundController.cs]
- 재시작: `Application.LoadLevel(loadedLevel)`로 씬 재로드. [근거: GameController.cs]
- `DontDestroyOnLoad` 싱글턴으로 오브젝트 중복 방지. [근거: DontDestroyOnLoad.cs]

3. 시스템 간 상호작용과 밸런스 기준

- 핵심 변수: `jumpSpeed 8`, 중력 배율 3.5, `timeDelay 1`, `timePace 0.01`, 파이프 Y 범위 -1~2, `speedPace 5`. [근거: FlappyBird.unity]
- 곡선: 스폰 간격은 1초에서 스폰당 0.01초 감소. 약 100회 스폰 시 0 이하가 되므로 하한 처리가 없다는 점이 위험 요소다(산술 추정). [추정: 코드에 하한 없음]
- 통과 가능성: 상승량과 파이프 간격의 관계는 수치 검증 기록 없음.

4. 게임의 나머지 부분의 방향

- 콘텐츠: 단일 무한 모드. 스테이지/메타 진행 없음 [추정: 과제 범위].
- UI/UX: 시작 안내, 점수, 최고점, 게임오버/재시작 버튼 [근거: 주석 처리된 `restartMenu`, `tutorButton`, `scoreText` 필드].
- 아트/사운드: 03 문서. 저장: PlayerPrefs.

5. 제작 과정 설계

- 순서: 새 → 바닥/배경 → 파이프 → 충돌/점수 → FSM 연동 → UI/사운드 → 정리. 병렬: 아트 배치와 FSM 작성.
- 도구 분담: 기획/제작 모두 수작업, AI 사용 기록 없음.
- 검증: 에디터 플레이로 한 판 반복. 인수인계 문서 없음.

6. 세부기획서 목록

- `03_세부기획서_플레이흐름과수치.md`: 상태 전이, 이벤트 계약, 수치 명세, UI/리소스 목록, 개선 과제.