

Pixel Block Puzzle 게임 기획서

1. 게임 개요 및 장르 배경

- 장르: 퍼즐 (블록 퍼즐, 테트리스류)
- 플랫폼: 모바일(Android)
- 출시 플랫폼: [원스토어](#)
- 엔진/툴: Unity, Visual Studio Code, Git, Aseprite, Bosca Ceoil
- 개발 인원: 1인 개발
- 타겟 유저: 캐주얼 퍼즐 게임을 선호하는 전 연령층

1.1. 장르 배경

블록 퍼즐 게임은 1980년대 테트리스의 성공 이후 전 세계적으로 사랑받는 장르입니다. 직관적인 규칙과 반복 플레이의 재미, 점수 경쟁 요소로 인해 남녀노소 누구나 쉽게 접근할 수 있습니다. 최근에는 모바일 환경에 최적화된 조작과 짧은 플레이 타임, 다양한 스킨/미션 등으로 진화하고 있습니다.

1.2. 차별화 기획 의도 및 핵심 특징

- 블록 형태의 다양성:** 기존 블록 퍼즐 게임과 달리, 본 게임은 대다수의 게임에서 보기 힘든 독특한 블록 형태(총 54종)를 적극적으로 도입하여, 반복적인 플레이에서도 새로운 전략과 재미를 제공합니다.
- 픽셀 블록의 감성적 디자인:** 각 블록의 단위(픽셀)는 귀여운 얼굴 표정을 가진 픽셀로 디자인되어, 단순한 도형이 아닌 감정이 담긴 캐릭터처럼 느껴지도록 하여, 반복 플레이의 지루함을 줄이고 친근함을 더합니다.
- 블록 제거 이펙트의 쾌감 강화:** 블록이 줄을 완성해 터질 때, 각 픽셀 블록이 실제로 부풀어 오르며 터지는 듯한 애니메이션 이펙트를 적용하여, 시각적 쾌감과 타격감을 극대화하였습니다.
- 차별화된 시각/감성 경험:** 픽셀 표정, 색상, 이펙트 등에서 기존 블록 퍼즐과 차별화된 감성적 경험을 제공하는 것이 본 게임의 핵심 목표입니다.

2. 주요 시스템 및 구성 요소

2.1. 게임 플레이

- 목표:** 다양한 모양의 블록을 9x9 그리드에 배치하여 가로/세로 줄을 완성하면 해당 줄이 사라짐. 최대한 많은 점수를 획득하는 것이 목표.
- 조작 방식:** 무작위로 주어진 3개의 블록 중 하나를 선택해 드래그 앤 드롭으로 그리드에 이동시켜 배치
- 게임 오버:** 블록을 더 이상 배치할 공간이 없을 때
- 알고리즘:** 블록 배치 가능 여부는 2중 for문으로 그리드의 모든 위치에 대해 블록이 들어갈 수 있는지 검사. 효율성을 위해 블록의 최소/최대 범위만 탐색.

```
// 블록 배치 가능 여부 확인 예시 (실제 구현 코드)
public bool CanPlaceBlock(int x, int y, BlockShape shape)
{
    if (shape == null) return false;
    if (x < 0 || y < 0 || x + shape.width > gridWidth || y + shape.height >
        gridHeight)
        return false;
```

```
for (int i = 0; i < shape.width; i++)
{
    for (int j = 0; j < shape.height; j++)
    {
        if (shape.shape[j, i] == 1)
        {
            if (!IsValidIndex(x + i, y + j) || grid[x + i, y + j] != null)
                return false;
        }
    }
}
return true;
}
```

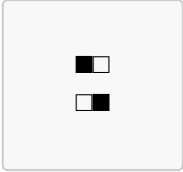
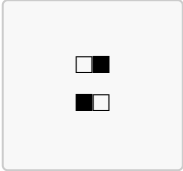

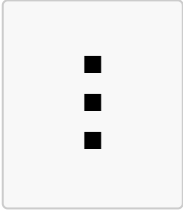
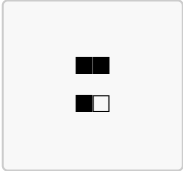
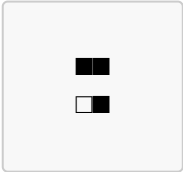
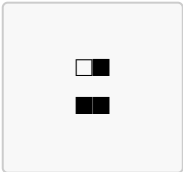
- **효율성:** 블록의 크기만큼 탐색하므로 불필요한 연산을 줄임. 블록이 3개 제공되므로, 3개 중 하나라도 배치 가능하면 게임이 계속됨.

2.2. 블록

- **다양한 형태의 블록:** 1~5칸 크기의 직선, 정사각형, L자, T자, Z자, 십자, 2x2, 3x3 등 다양한 모양 제공
- **10가지 색상 정책 및 색상 중복 방지:** 각 픽셀 블록은 10가지의 다양한 색상으로 구성되며, 하단에 제시되는 3개의 블록은 항상 서로 다른 색상으로 등장하도록 설계하여, 시각적 구별성과 게임의 완성도를 높였습니다.

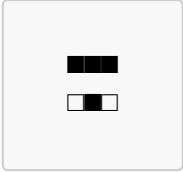
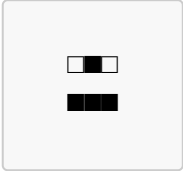
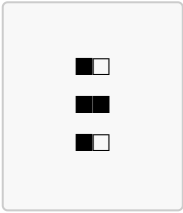
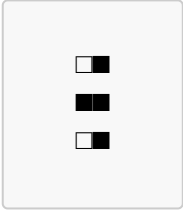
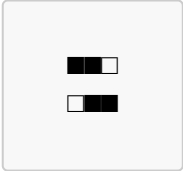
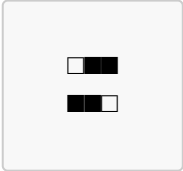
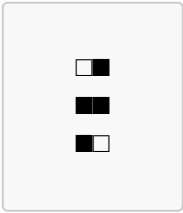
아래 표는 BlockManager.cs에 실제 정의된 54개 블록의 도식, 배열, 크기, 설명을 모두 정리한 것입니다. 각 블록은 실제 게임에서 사용되는 형태와 동일하게 2차원 배열로 표현되며, 도식은 1=■, 0=공백으로 시각화했습니다.

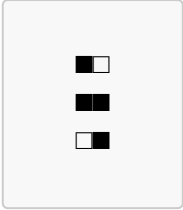


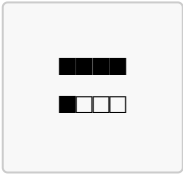
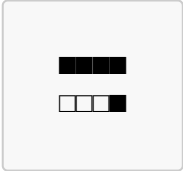
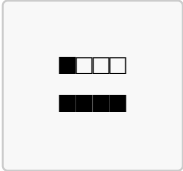
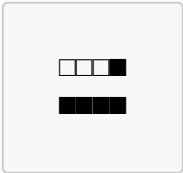
번호	도식	배열	크기 (WxH)	블록 설명
1	<div><div>■</div></div>	{{1}}	1x1	단일 블록
2	<div><div>■■</div></div>	{{1,1}}	2x1	가로 2칸
3	<div><div>■ ■</div></div>	{{1},{1}}	1x2	세로 2칸

번호	도식	배열	크기 (WxH)	블록 설명
4		{ {1,0}, {0,1} }	2x2	대각선 2칸 (↘)
5		{ {0,1}, {1,0} }	2x2	대각선 2칸 (↗)
6		{ {1,1,1} }	3x1	가로 3칸
7		{ {1}, {1}, {1} }	1x3	세로 3칸
8		{ {1,1}, {1,0} }	2x2	└자 (좌상)
9		{ {1,1}, {0,1} }	2x2	┐자 (우상)
10		{ {0,1}, {1,1} }	2x2	┐자 (좌하)

번호	도식	배열	크기 (WxH)	블록 설명
11		{ {1,0}, {1,1} }	2x2	└자 (우하)
12		{ {1,0,0}, {0,1,0}, {0,0,1} }	3x3	대각선 3칸 (↘)
13		{ {0,0,1}, {0,1,0}, {1,0,0} }	3x3	대각선 3칸 (↖)
14		{ {1,1}, {1,1} }	2x2	정사각형 4칸
15		{ {1,1,1,1} }	4x1	가로 4칸
16		{ {1}, {1}, {1}, {1} }	1x4	세로 4칸
17		{ {1,0}, {1,0}, {1,1} }	2x3	└자 (좌상, 3칸)

번호	도식	배열	크기 (WxH)	블록 설명
18		{ {1,1}, {1,0}, {1,0} }	2x3	└ 자 (좌하, 3칸)
19		{ {0,1}, {0,1}, {1,1} }	2x3	┐ 자 (우상, 3칸)
20		{ {1,1}, {0,1}, {0,1} }	2x3	┐ 자 (우하, 3칸)
21		{ {1,1,1}, {1,0,0} }	3x2	└ 자 (좌상, 4칸)
22		{ {1,1,1}, {0,0,1} }	3x2	┐ 자 (우상, 4칸)
23		{ {1,0,0}, {1,1,1} }	3x2	└ 자 (좌하, 4칸)
24		{ {0,0,1}, {1,1,1} }	3x2	┐ 자 (우하, 4칸)

번호	도식	배열	크기 (WxH)	블록 설명
25		{ {1,1,1}, {0,1,0} }	3x2	└자
26		{ {0,1,0}, {1,1,1} }	3x2	┐자
27		{ {1,0}, {1,1}, {1,0} }	2x3	┘자
28		{ {0,1}, {1,1}, {0,1} }	2x3	┙자
29		{ {1,1,0}, {0,1,1} }	3x2	S자 (좌상)
30		{ {0,1,1}, {1,1,0} }	3x2	S자 (우상)
31		{ {0,1}, {1,1}, {1,0} }	2x3	Z자 (좌상)

번호	도식	배열	크기 (WxH)	블록 설명
32		{ {1,0}, {1,1}, {0,1} }	2x3	Z자 (우상)
33		{ {1,1,1,1,1} }	5x1	가로 5칸
34		{ {1}, {1}, {1}, {1}, {1} }	1x5	세로 5칸
35		{ {1,1,1,1}, {1,0,0,0} }	4x2	└자 (좌상, 5칸)
36		{ {1,1,1,1}, {0,0,0,1} }	4x2	┐자 (우상, 5칸)
37		{ {1,0,0,0}, {1,1,1,1} }	4x2	└자 (좌하, 5칸)
38		{ {0,0,0,1}, {1,1,1,1} }	4x2	┐자 (우하, 5칸)

번호	도식	배열	크기 (WxH)	블록 설명
39		{ {1,1,1}, {1,0,0}, {1,0,0} }	3x3	└자 (좌상, 6칸)
40		{ {1,1,1}, {0,0,1}, {0,0,1} }	3x3	┐자 (우상, 6칸)
41		{ {1,0,0}, {1,0,0}, {1,1,1} }	3x3	└자 (좌하, 6칸)
42		{ {0,0,1}, {0,0,1}, {1,1,1} }	3x3	┐자 (우하, 6칸)
43		{ {1,1,1}, {0,1,0}, {0,1,0} }	3x3	┣자 (5블록)
44		{ {0,1,0}, {0,1,0}, {1,1,1} }	3x3	┣자 (5블록)

번호	도식	배열	크기 (WxH)	블록 설명
45		{ {1,0,0}, {1,1,1}, {1,0,0} }	3x3	┼ 자 (5블록)
46		{ {0,0,1}, {1,1,1}, {0,0,1} }	3x3	┴ 자 (5블록)
47		{ {0,1,0}, {1,1,1}, {0,1,0} }	3x3	십자(+)
48		{ {1,0,1}, {1,1,1} }	3x2	┌자 변형
49		{ {1,1,1}, {1,0,1} }	3x2	┐자 변형
50		{ {1,1}, {1,0}, {1,1} }	2x3	┼ 자 변형
51		{ {1,1}, {0,1}, {1,1} }	2x3	┴ 자 변형

번호	도식	배열	크기 (WxH)	블록 설명
52		{ {1,1,1}, {1,1,1} }	3x2	6블록 정사각형
53		{ {1,1}, {1,1}, {1,1} }	2x3	6블록 직사각형
54		{ {1,1,1}, {1,1,1}, {1,1,1} }	3x3	9블록 정사각형

- 도식에서 ■는 블록, □는 빈칸(0)입니다.
- 배열은 실제 BlockManager.cs의 정의와 동일하게 표기했습니다.
- 크기는 배열의 가로(W)x세로(H)입니다.
- 설명은 블록의 형태와 용도를 간단히 명시했습니다.

2.2.1. 블록 스폰(Spawn) 및 선택 알고리즘 예시

```
// 블록 3개를 하나씩 선택하는 재귀 알고리즘 (BlockManager.cs)
private bool SelectBlocksRecursive(bool[,] grid, int blockIdx, List<BlockShape>
selected)
{
    if (blockIdx >= 3)
        return true;

    // 현재 그리드 상태에서 배치 가능한 블록 후보 추출
    var validBlocks = possibleBlockShapes.Where(block =>
IsValidPlacementInTest(grid, block)).ToList();
    List<BlockShape> remainingBlocks = new List<BlockShape>(validBlocks);

    while (remainingBlocks.Count > 0)
    {
        // 가중치 기반 무작위 선택
        BlockShape selectedBlock = SelectWeightedRandomBlock(remainingBlocks);
        remainingBlocks.Remove(selectedBlock);

        bool[,] gridCopy = CloneGrid(grid);
        if (TryPlaceBlockAnywhere(gridCopy, selectedBlock))
        {
```

```

        CheckAndClearLines(gridCopy);
        selected.Add(selectedBlock);
        if (SelectBlocksRecursive(gridCopy, blockIndex + 1, selected))
            return true;
        selected.RemoveAt(selected.Count - 1);
    }
}
return false;
}

// 가중치 기반 무작위 블록 선택 (BlockManager.cs)
private BlockShape SelectWeightedRandomBlock(List<BlockShape> validBlocks)
{
    float totalWeight = 0;
    foreach (var block in validBlocks)
    {
        totalWeight += Mathf.Pow(random_weight, block.count - 1);
    }

    float randomValue = UnityEngine.Random.Range(0f, totalWeight);
    float currentSum = 0;

    foreach (var block in validBlocks)
    {
        currentSum += Mathf.Pow(random_weight, block.count - 1);
        if (randomValue <= currentSum)
        {
            return block;
        }
    }
    return validBlocks[validBlocks.Count - 1];
}

```

- 위 알고리즘은 실제 BlockManager.cs에서 3개의 블록을 하나씩, 그리드 상황에 따라 배치 가능성을 시뮬레이션하며 선택하는 구조입니다.
- 각 블록은 가중치(블록 크기 등)에 따라 무작위로 선택되며, 반드시 배치 가능한 조합만 생성되도록 설계되어 있습니다.
- **가중치 기반 무작위 선택을 사용하는 이유:** 더 많은 블록 단위(큰 블록, 많은 칸을 차지하는 블록)로 구성된 블록이 더 높은 확률로 등장하도록 하여, 게임의 전략성과 다양성을 높이고, 단순한 1~2칸짜리 블록만 반복적으로 등장하는 것을 방지하기 위함입니다.

2.3. 이펙트

- **블록 이펙트:** 블록 생성, 이동, 회전, 삭제 시 이펙트
- **점수 이펙트:** 점수 획득 시 애니메이션 및 사운드 재생
- **콤보 이펙트:** 연속으로 줄을 지울 때 콤보 수치 표시 및 보너스 점수

2.4. UI/UX

- **메인 화면:** 게임 시작, 설정, 랭킹, 종료 버튼
- **게임 화면:** 9x9 그리드, 블록 선택 영역, 점수 표시, 일시정지/재시작 버튼

- **게임 오버 화면:** 최종 점수, 재시작, 메인으로 돌아가기
- **UX 설계:**
 - 직관적인 드래그 앤 드롭
 - 블록을 놓을 수 있는 상황일 때, 해당 위치의 그리드에 shadow(그림자)를 표시하여 플레이어가 즉시 배치 가능 여부를 시각적으로 확실히 인지할 수 있도록 구현
 - 블록이 터질 때(줄이 완성되어 사라질 때) 진동(Vibration) 효과를 제공하여 타격감과 피드백을 강화
 - 콤보가 쌓일 때, 콤보 수치를 화면 중앙에 크게 표시하여 플레이어가 연속 성공을 즉각적으로 인지할 수 있도록 설계
 - 블록 배치 가능/불가 시 색상 변화 등 시각적 피드백

3. 아트 에셋

- 모든 픽셀 아트(블록, 그리드, UI 등)는 **Aseprite**를 활용하여 직접 제작하였습니다. 각 블록의 픽셀 표정, 색상, 이펙트 등도 Aseprite로 세밀하게 작업하여 게임의 감성적 완성도를 높였습니다.
- **블록 스프라이트:** 다양한 색상/모양의 블록 이미지(**Assets/Sprites/**)
- **UI 이미지:** 버튼, 배경, 아이콘 등(**Assets/Sprites/**, **Assets/Fonts/**)
- **배경 이미지:** 게임 배경, 대시보드 등
- **이펙트:** 줄 삭제, 콤보 등 시각 효과(**Assets/Shaders/**, **Assets/Prefabs/**)

4. 사운드 에셋

- 게임의 배경음악(BGM)과 효과음(Sound Effect)은 ****Bosca Ceoil****을 활용하여 직접 제작하였습니다. 게임의 분위기와 플레이 템포에 맞는 사운드를 자체적으로 작곡 및 편집하여, 상용 음원 없이 독창적인 사운드 경험을 제공합니다.
- **효과음:** 블록 배치, 줄 삭제, 버튼 클릭 등(**Assets/Sounds/**)
- **배경음악:** 게임 진행 중 BGM

5. 주요 폴더 구조

- **Assets/Scenes/** : 게임 씬, UI 씬 등
- **Assets/Scripts/** : 게임 로직, UI, 오브젝트 풀링 등 스크립트
- **Assets/Prefabs/** : 블록, UI 등 프리팹
- **Assets/Sprites/** : 블록, UI, 배경 등 스프라이트
- **Assets/Sounds/** : 효과음, 배경음악
- **Assets/Localization/** : 다국어 지원 리소스

6. 확장 요소 및 기타 시스템

- **랭킹 시스템(Leaderboard):**
 - 플레이어의 최고 점수를 서버에 저장하고, 상위 랭커를 실시간으로 조회할 수 있는 시스템이 구현되어 있습니다.
 - Unity Services의 Leaderboards 패키지를 활용하여, 익명 인증 및 점수 제출, 닉네임 입력, 금지어 필터, 실시간 랭킹 조회, 내 순위 표시, 100위까지의 랭커 리스트, 닉네임 검증(한글/영문/숫자/특수문자, 금지어, 바이트 제한) 등 다양한 기능을 제공합니다.
 - UI는 **LeaderboardEntryUI**, **LeaderboardPanel** 등으로 구성되어 있으며, 닉네임 입력 패널, 피드백 메시지, 내 순위 강조 등 UX가 강화되어 있습니다.

◦ 예시 코드:

```
// 닉네임 검증 (외부 금지어 파일 대조 포함, 실제 코드 기반)
private bool IsNicknameValid(string nickname)
{
    // 1. 허용 문자(영문, 숫자, 밑줄, 공백, 한글)
    string pattern = @"^[a-zA-Z0-9_ \uAC00-\uD7A3]+$";
    if (!System.Text.RegularExpressions.Regex.IsMatch(nickname, pattern))
        return false;
    // 2. 금지어(외부 파일) 대조
    foreach (string bad in bannedWords)
    {
        if (string.IsNullOrEmpty(bad)) continue;
        if (nickname.IndexOf(bad, StringComparison.OrdinalIgnoreCase) >= 0)
            return false;
    }
    // 3. 바이트 수 제한(UTF-8 기준 20바이트)
    if (System.Text.Encoding.UTF8.GetByteCount(nickname) > 20) return false;
    return true;
}

// 점수 제출 (비동기, 닉네임 검증 포함, 실제 구조 기반)
public async void SubmitScore(int score, string nickname)
{
    if (!IsNicknameValid(nickname))
    {
        Debug.LogWarning("닉네임이 유효하지 않습니다.");
        return;
    }
    try
    {
        var options = new AddPlayerScoreOptions { Metadata = new
Dictionary<string, string> { { "nickname", nickname } } };
        await
LeaderboardsService.Instance.AddPlayerScoreAsync(leaderboardId, score,
options);
        Debug.Log($"Score submitted: {score} with nickname: {nickname}");
    }
    catch (Exception ex)
    {
        Debug.LogError("Failed to submit score: " + ex.Message);
    }
}

// 랭킹 조회 (비동기, 실제 구조 기반)
public async void GetLeaderboard()
{
    if (isFetchingLeaderboard) return;
    isFetchingLeaderboard = true;
    try
    {
        var response = await
LeaderboardsService.Instance.GetScoresAsync(leaderboardId, new
```

```

GetScoresOptions { Limit = 100, IncludeMetadata = true });
    // ...데이터 파싱 및 UI 갱신 로직...
}
catch (Exception ex)
{
    Debug.LogError("Failed to retrieve leaderboard: " + ex.Message);
}
finally
{
    isFetchingLeaderboard = false;
}
}

```

- **효율성:** 중복 호출 방지, 캐싱, 비동기 처리, 금지어 로컬 파일 캐싱 등으로 최적화되어 있습니다.
- **확장성:** Firebase, PlayFab 등 외부 서비스로의 확장도 고려할 수 있습니다.

- **광고 시스템(Unity Ads):**

- 현재 ****보상형 광고(Rewarded Ads)****만 사용하고 있습니다.
- **보상형 광고**는 게임 오버 시 1회에 한해 클릭할 수 있으며, 동영상 광고를 끝까지 시청하면 1회에 한해 부활(continue)이 가능합니다.
- 2번째 게임 오버부터는 더 이상 보상형 광고(부활)가 제공되지 않습니다.
- 광고 시청 및 보상 지급의 실제 핵심 로직은 다음과 같습니다.

```

// RewardedAdsButton.cs
public void ShowAd()
{
    _showAdButton.interactable = false;
    Advertisement.Show(_adUnitId, this);
}

public void OnUnityAdsShowComplete(string adUnitId,
UnityAdsShowCompletionState showCompletionState)
{
    if (adUnitId.Equals(_adUnitId) &&
showCompletionState.Equals(UnityAdsShowCompletionState.COMPLETED))
    {
        Debug.Log("Unity Ads Rewarded Ad Completed");
        // 실제 보상 지급: 1회 부활
        GameManager.Instance.ReviveGame();
    }
}

// GameManager.cs
public void ReviveGame()
{
    reviveCount++;
    isGameOver = false;
    gridManager.ClearGameOverBlocks();
    gridManager.ReactivateGridBlocks();
    blockManager.GenerateNewBlocks();
}

```

```
uiManager.HideGameOverPanel();  
soundManager.PlayReviveSound();  
soundManager.PlayBGM();  
inputHandler.EnableDrag();  
}
```

- **효율성**: 광고 로딩 상태 체크, 중복 노출 방지, 네트워크 예외 처리 등 안정성을 고려한 설계
- **수익화**: 광고 시청률, 클릭률, 보상형 광고 활용 등으로 수익 극대화 가능

본 기획서는 실제 구현된 랭킹 시스템(leaderboard) 및 광고 시스템(Unity Ads) 구조와 코드를 기반으로 작성되었습니다. 실제 구현 내용에 따라 일부 내용은 변경될 수 있습니다.