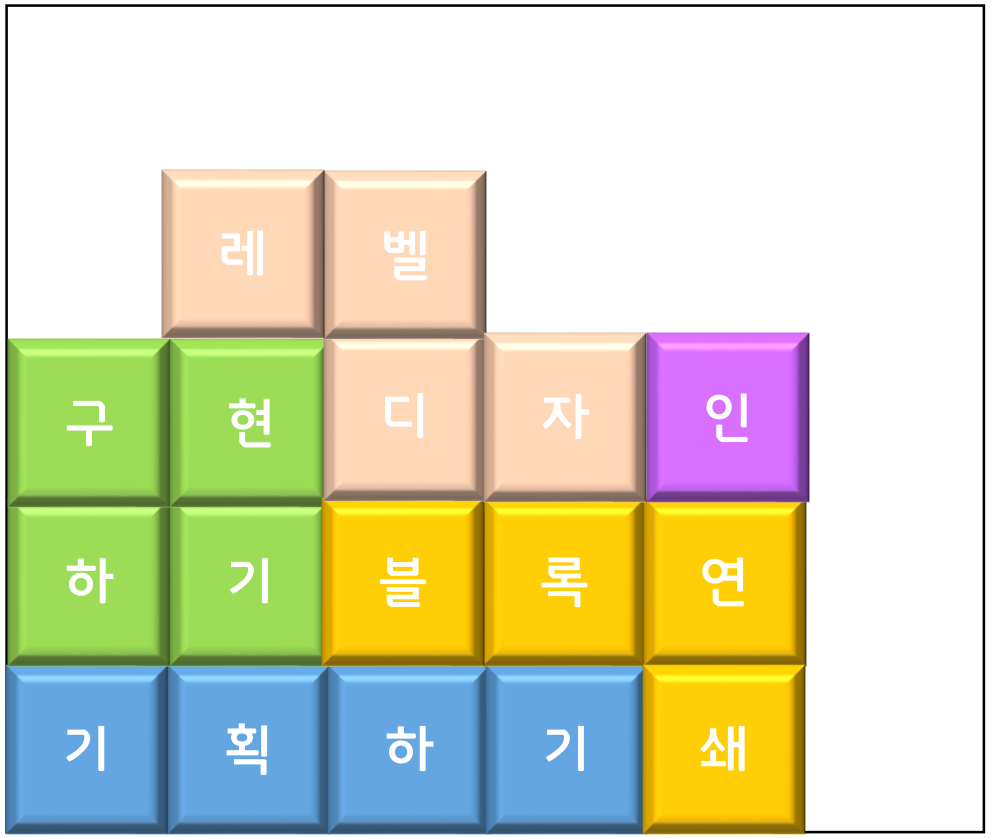
유니티
게임 제작 입문

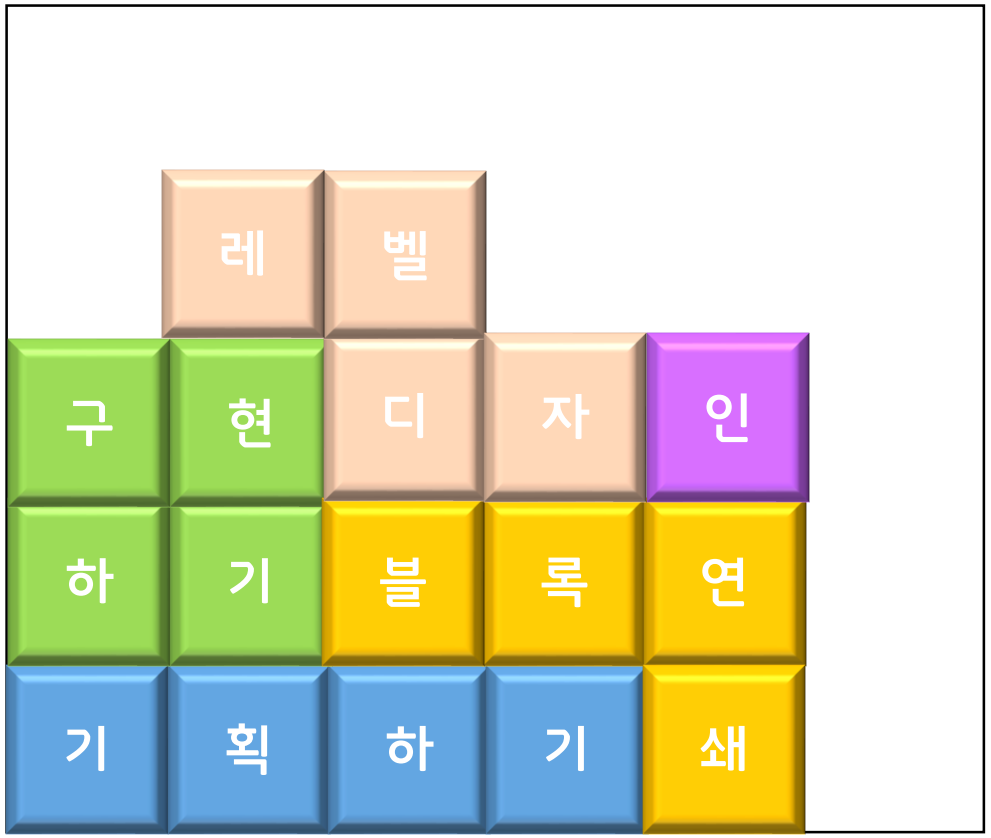
퍼즐 게임 만들기

길벗

퍼즐게임 만들기



퍼즐게임 만들기



* 퍼즐게임 기획하기

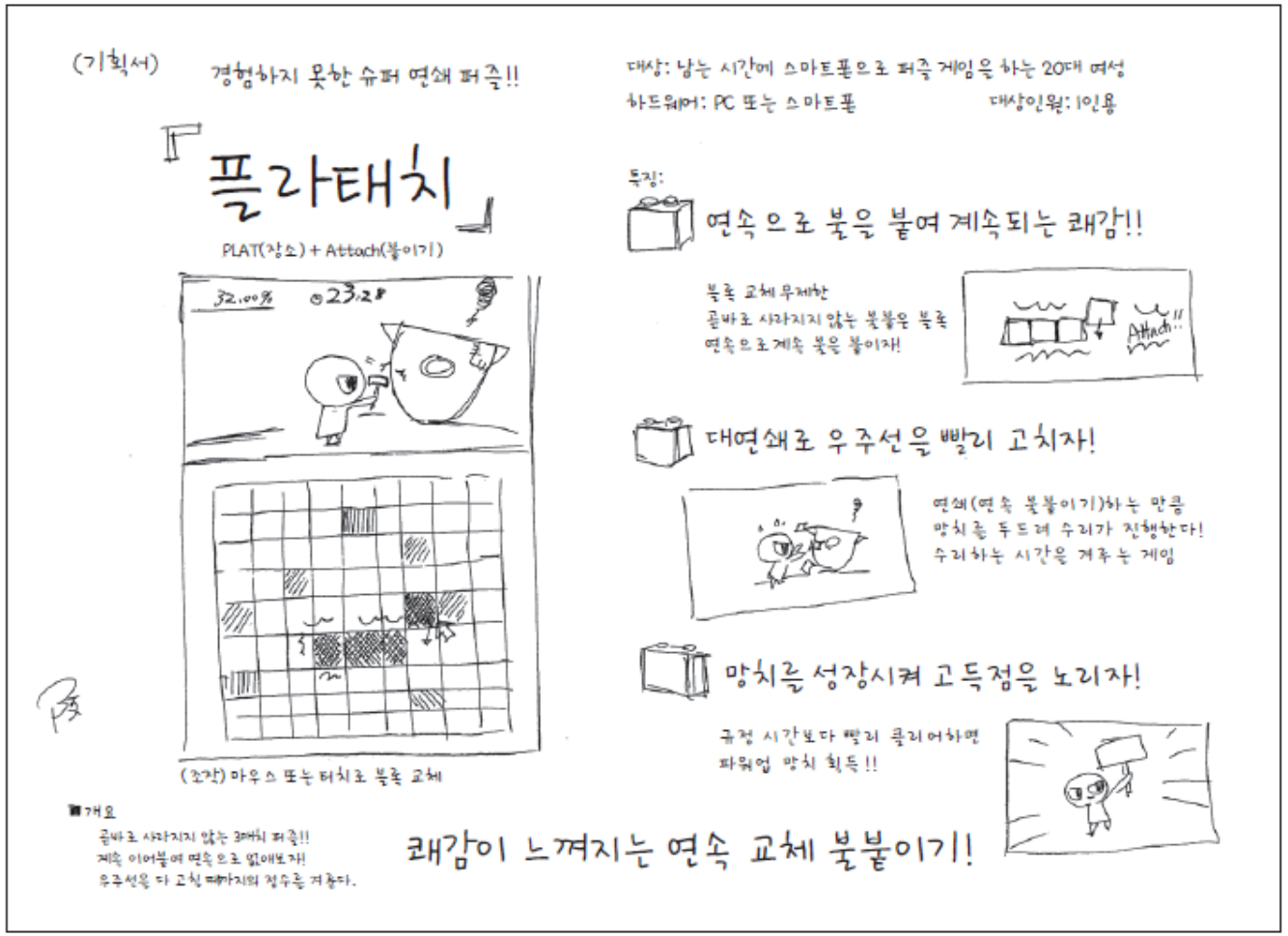
게임 주제	아이디어 정리	기획서 작성	사양서 작성
연달아 있어도 사라지지 않는 3매치 퍼즐 ex) 헥사	종이에 연필로 마구잡이식으로 써본다	규칙, 캐릭터 등	게임 실행화면 각 요소 별로 만들기

* 퍼즐게임 기획 - 규칙

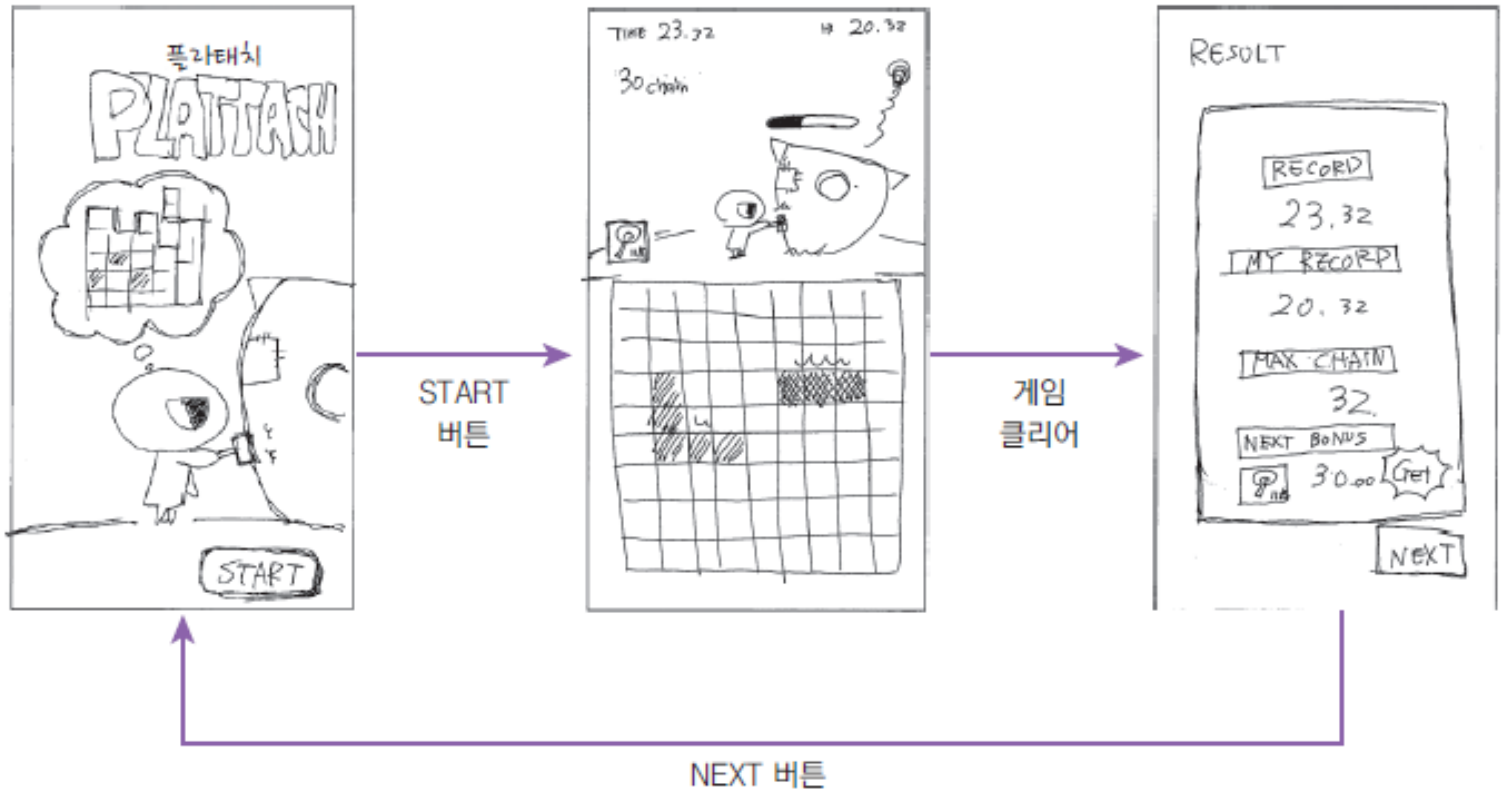
다음과 같은 규칙을 가지고 게임을 제작하도록 한다.

- 모아야 할 블록 수는 **세 개**
- 사라질 때까지 시간이 걸린다 (투명도 조절을 해서 사라지는 것을 표현)
- 점수는 **연쇄했을 때 더 높은 점수**를 얻을 수 있도록 한다
- 블록은 언제든지 자유롭게 교체할 수 있다
- **연속 교체 불 붙이기** (블록이 사라지기 전에 연속적으로 엮어서 사라지도록)
- 사라진 블록은 위에서 보충
- **게임 종료 규칙**을 만든다. (제한시간, 10턴 이내에 1만 점 획득 등)

* 퍼즐게임 기획서

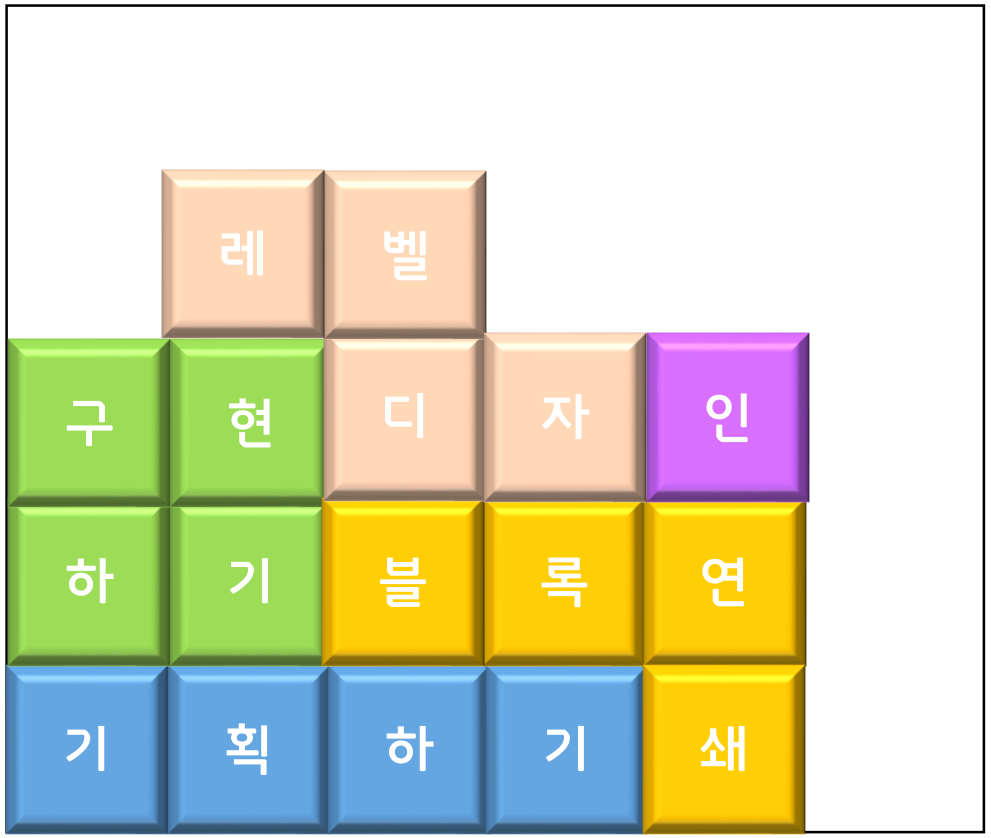

(기획서)
경험하지 못한 슈퍼 연쇄 퍼즐!!
「플라타치」
PLAT(장소) + Attach(붙이기)
대상: 남는 시간에 스마트폰으로 퍼즐 게임을 하는 20대 여성
하드웨어: PC 또는 스마트폰
대상인원: 1인용
특징:
연속으로 볼을 붙여 계속되는 쾌감!!
블록 교체 무제한
곧바로 사라지지 않는 볼볼을 블록
연속으로 계속 볼을 붙이자!
Attach!!
대연쇄로 우주선을 빨리 고치자!
연쇄(연속 볼볼이기)하는 만큼
망치를 두드려 수리가 진행한다!
수리하는 시간을 겨루는 게임
망치를 성장시켜 고득점을 노리자!
규정 시간보다 빨리 클리어하면
파워업 망치 획득!!
(조작) 마우스 또는 터치로 블록 교체
■개요
곧바로 사라지지 않는 3매치 퍼즐!!
계속 이어붙여 연속으로 없애보자!
우주선을 다 고칠 때까지의 점수를 겨룰다.
쾌감이 느껴지는 연속 교체 볼붙이기!

* 퍼즐게임 사양서


플라태시
PLATTASH
START
START
버튼
Time 23.32
H 20.32
30chain
게임
클리어
RESULT
RECORD
23.32
MY RECORD
20.32
MAX CHAIN
32
NEXT BONUS
30.00
Get
NEXT
NEXT 버튼

* 게임에 필요한 데이터 정리

퍼즐게임 만들기



* 프로그램 작성 순서

1. 블록을 바닥에 채운다.
2. 블록을 마우스로 잡는다.
3. 잡은 블록을 상하좌우로 교체한다.
4. 가로세로 같은 색 세 개가 모이면 블록에 불을 붙여 사라지게 한다.
5. 사라진 만큼 블록을 위에서 채운다.
6. 점수가 더해지도록 한다.
7. 레벨 디자인(난이도 조정)을 한다.
8. 시퀀스를 연결한다.

1. 블록을 바닥에 채운다.

- 프로젝트 만들기

① [File] → [New Project]로 원하는 위치에 Plattach 폴더를 만들고 지정한다.
② 유니티 에디터가 재시작된다.
③ [GameObject] → [Create Other] → [Directional Light]로 조명을 생성한다.
④ [File] → [Save Scene]를 실행하고 '**GameScene**'으로 저장한다.

- 필요한 스크립트 준비

① [GameObject] → [Create Empty]을 선택하고 'GameRoot'로 이름을 변경한다.
② [GameObject] → [Create Other] → [Cube]를 선택하고 'Block'으로 이름을 변경한다.
③ [Assets] → [Create] → [C# Script]로 'SceneControl', 'BlockRoot'. 'BlockControl' 스크립트를 만든다.
④ [GameRoot]에 SceneControl 스크립트와 BlockRoot 스크립트를 컴포넌트로 등록한다.
⑤ Block에 BlockControl 스크립트를 컴포넌트로 등록한다.
⑥ Block을 Project 브라우저로 드래그 앤 드롭하여 프리팹으로 만든다.
⑦ Project 브라우저의 Block을 'BlockPrefab'으로 이름 변경한다.
⑧ 씬에 있는 BlockPrefab을 삭제한다(Hierarchy에서 삭제).

1. 블록을 바닥에 채운다.

* 블록 나열하기

BlockControl.cs

```
// 블록에 관한 정보를 다룬다.
public class Block {
public static float COLLISION_SIZE = 1.0f; // 블록의 충돌 크기.
public static float VANISH_TIME = 3.0f; // 불 붙고 사라질 때까지의 시
간.
public struct iPosition { // 그리드에서의 좌표를 나타내는 구조체.
    public int x; // X 좌표.
    public int y;} // Y 좌표.

public enum COLOR { // 블록 색상.
    NONE = -1, // 색 지정 없음.
    PINK = 0, // 분홍색.
    BLUE, // 파란색.
    YELLOW, // 노란색.
    GREEN, // 녹색.
    MAGENTA, // 마젠타.
    ORANGE, // 주황색.
    GRAY, // 그레이.
    NUM, // 컬러가 몇 종류인지 나타낸다(=7).
    FIRST = PINK, // 초기 컬러(분홍색).
    LAST = ORANGE, // 최종 컬러(주황색).
    NORMAL_COLOR_NUM = GRAY.};
    // 보통 컬러(회색 이외의 색)의 수.

public enum DIR4 { // 상하좌우 네 방향.
    NONE = -1, // 방향지정 없음.
    RIGHT, // 우.
    LEFT, // 좌.
    UP, // 상.
    DOWN, // 하.
    NUM, // 방향이 몇 종류 있는지 나타낸다(=4).
};
public static int BLOCK_NUM_X = 9;
// 블록을 배치할 수 있는 X방향 최대수.
public static int BLOCK_NUM_Y = 9;
// 블록을 배치할 수 있는 Y방향 최대수.
}
```

1. 블록을 바닥에 채운다.

* 블록 나열하기

BlockControl.cs

```
public class BlockControl : MonoBehaviour {
public Block.COLOR color = (Block.COLOR); // 블록 색.
public BlockRoot block_root = null; // 블록의 신.
public Block.iPosition i_pos; // 블록 좌표.
void Start() {
    this.setColor(this.color); // 색칠을 한다.
}
void Update() {
}
// 인수 color의 색으로 블록을 칠한다.
public void setColor(Block.COLOR color)
{
    this.color = color; // 이번에 지정된 색을 멤버 변수에 보관한다.
    Color color_value; // Color 클래스는 색을 나타낸다.
switch(this.color) { // 칠할 색에 따라서 갈라진다.
default:
case Block.COLOR.PINK:
    color_value = new Color(1.0f, 0.5f, 0.5f);
    break;
case Block.COLOR.BLUE:
    color_value = Color.blue;
    break;
case Block.COLOR.YELLOW:
    color_value = Color.yellow;
    break;
case Block.COLOR.GREEN:
    color_value = Color.green;
    break;
case Block.COLOR.MAGENTA:
    color_value = Color.magenta;
    break;
case Block.COLOR.ORANGE:
    color_value = new Color(1.0f, 0.46f, 0.0f);
    break;}
// 이 게임 오브젝트의 머티리얼 색상을 변경한다.
this.renderer.material.color = color_value;}}
```

1. 블록을 바닥에 채운다.

* 블록 나열하기 - BlockRoot.cs

그림 9-5 그리드로 관리되는 블록들

블록을 가로세로 바둑판 모양으로 관리

1. 블록을 바닥에 채운다.

* 블록 나열하기 - BlockRoot.cs

BlockRoot.cs

```
public class BlockRoot : MonoBehaviour {
public GameObject BlockPrefab = null; // 만들어낼 블록의 프리팹.
public BlockControl[,] blocks; // 그리드.
void Start() {}
void Update() {}

// 블록을 만들어 내고 가로 9칸, 세로 9칸에 배치한다.
public void initialSetUp(){
    // 그리드의 크기를 9×9로 한다.
    this.blocks = new BlockControl [Block.BLOCK_NUM_X,
    Block.BLOCK_NUM_Y];
    // 블록의 색 번호.
    int color_index = 0;
    for(int y = 0; y < Block.BLOCK_NUM_Y; y+ +) { // 처음~마지막행
    for(int x = 0; x < Block.BLOCK_NUM_X; x+ +) { // 왼쪽~오른쪽
    // BlockPrefab의 인스턴스를 씬에 만든다.
    GameObject game_object = Instantiate(this.BlockPrefab) as
    GameObject;
    // 위에서 만든 블록의 BlockControl 클래스를 가져온다.
    BlockControl block =
    game_object.GetComponent<BlockControl>();
    // 블록을 그리드에 저장한다.
    this.blocks[x, y] = block;
    // 블록의 위치 정보(그리드 좌표)를 설정한다.
    block.i_pos.x = x;
    block.i_pos.y = y;
    // 각 BlockControl이 연계할 GameRoot는 자신이라고 설정한다.
    block.block_root = this;
    // 그리드 좌표를 실제 위치(씬의 좌표)로 변환한다.
    Vector3 position = BlockRoot.calcBlockPosition(block.i_pos);
    // 씬의 블록 위치를 이동한다.
    block.transform.position = position;
    // 블록의 색을 변경한다.
    block.setColor((Block.COLOR)color_index);
    // 블록의 이름을 설정(후술)한다.
    block.name = "block(" + block.i_pos.x.ToString() +
    "," + block.i_pos.y.ToString() + ")";
    // 전체 색 중에서 임의로 하나의 색을 선택한다.
    color_index =
    Random.Range(0, (int)Block.COLOR.NORMAL_COLOR_NUM);
    }}
}}
```

1. 블록을 바닥에 채운다.

* 블록 나열하기 - BlockRoot.cs

BlockRoot.cs

```
// 지정된 그리드 좌표로 씬에서의 좌표를 구한다.
public static Vector3 calcBlockPosition(Block.iPosition i_pos) {
    // 배치할 왼쪽 위 구석 위치를 초기값으로 설정한다.
    Vector3 position = new Vector3(-(Block.BLOCK_NUM_X / 2.0f - 0.5f),
    -(Block.BLOCK_NUM_Y / 2.0f - 0.5f), 0.0f);
    // 초깃값 + 그리드 좌표 × 블록 크기.
    position.x += (float)i_pos.x * Block.COLLISION_SIZE;
    position.y += (float)i_pos.y * Block.COLLISION_SIZE;
    return(position); // 씬에서의 좌표를 반환한다.
}
```

그림 9-6 blocks[x, y] 배열에서 어느 블록인지 알기 쉽게 한다

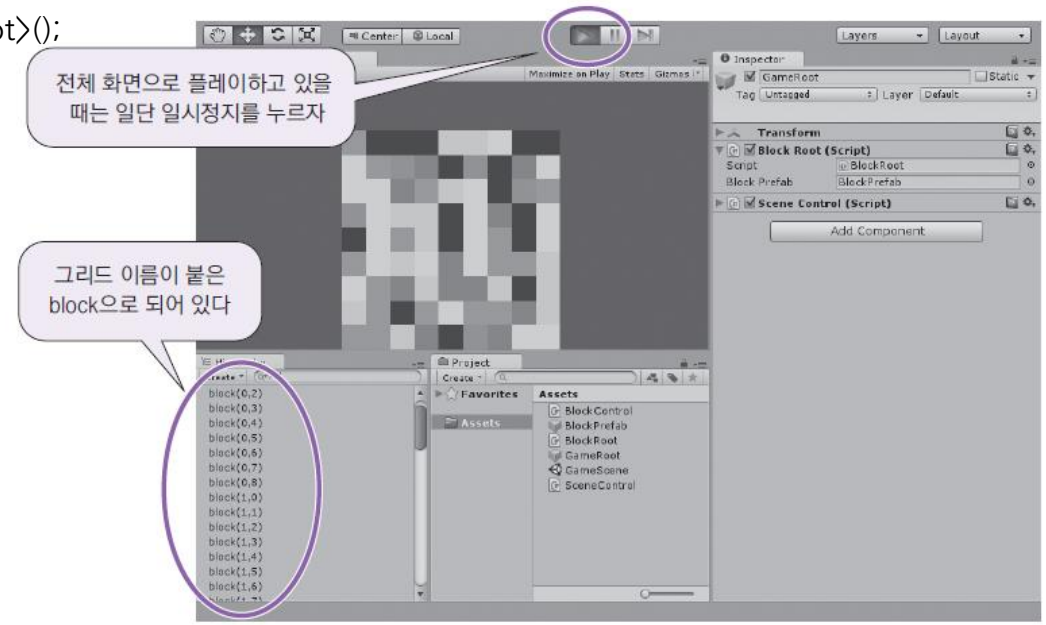
1. 블록을 바닥에 채운다.

* 블록 나열하기

SceneControl.cs

```
public class SceneControl : MonoBehaviour {
private BlockRoot block_root = null;
void Start() {
    // BlockRoot 스크립트를 가져온다.
    this.block_root = this.gameObject.GetComponent<BlockRoot>();
    // BlockRoot 스크립트의 initialSetUp()을 호출한다.
    this.block_root.initialSetUp();
}
void Update() {}
}
```

그림 9-8 Hierarchy에 그리드 좌표 이름으로 표시된다



2. 마우스로 블록 잡기

BlockControl.cs / Block 클래스

```
// 블록이 어던 상태인지 알려주는 클래스
public enum STEP {
    NONE = -1, // 상태 정보 없음.
    IDLE = 0, // 대기 중.
    GRABBED, // 잡혀 있음.
    RELEASED, // 떨어진 순간.
    SLIDE, // 슬라이드 중.
    VACANT, // 소멸 중.
    RESPAWN, // 재생성 중.
    FALL, // 낙하 중.
    LONG_SLIDE, // 크게 슬라이드 중.
    NUM, // 상태가 몇 종류인지 표시.
};
```

BlockControl.cs / BlockControl 클래스

```
public Block.STEP step = Block.STEP.NONE; // 지금 상태.
public Block.STEP next_step = Block.STEP.NONE; // 다음 상태.
private Vector3 position_offset_initial = Vector3.zero; // 교체 전 위치.
public Vector3 position_offset = Vector3.zero; // 교체 후 위치.

void Start() {
    this.setColor(this.color);
    this.next_step = Block.STEP.IDLE; // 다음 블록을 대기중으로.
}
```

2. 마우스로 블록 잡기

Update() : 블록이 잡혔을 때에 블록을 크게 하고 그렇지 않을 때는 원래대로 돌아간다.

BlockControl.cs / BlockControl 클래스

```
void Update() {
    Vector3 mouse_position; // 마우스 위치.
    this.block_root.unprojectMousePosition( // 마우스 위치 획득.
    out mouse_position, Input.mousePosition);
    // 획득한 마우스 위치를 X와 Y만으로 한다.
    Vector2 mouse_position_xy =
    new Vector2(mouse_position.x, mouse_position.y);
    // '다음 블록' 상태가 '정보 없음' 이외인 동안.
    // ='다음 블록' 상태가 변경된 경우.
    while(this.next_step != Block.STEP.NONE) {
    this.step = this.next_step;
    this.next_step = Block.STEP.NONE;
    switch(this.step) {
    case Block.STEP.IDLE: // '대기' 상태.
        this.position_offset = Vector3.zero;
        // 블록 표시 크기를 보통 크기로 한다.
        this.transform.localScale = Vector3.one * 1.0f; break;
    case Block.STEP.GRABBED: // '잡힌' 상태.
        // 블록 표시 크기를 크게 한다.
        this.transform.localScale = Vector3.one * 1.2f; break;
    case Block.STEP.RELEASED: // '떨어져 있는' 상태.
        this.position_offset = Vector3.zero;
        // 블록 표시 크기를 보통 사이즈로 한다.
        this.transform.localScale = Vector3.one * 1.0f;
        break;}}
    // 그리드 좌표를 실제 좌표(씬의 좌표)로 변환하고.
    // position_offset을 추가한다.
    Vector3 position =
    BlockRoot.calcBlockPosition(this.i_pos) + this.position_offset;
    // 실제 위치를 새로운 위치로 변경한다.
    this.transform.position = position;
}
```

2. 마우스로 블록 잡기

BlockControl.cs / BlockControl 클래스

```
public void beginGrab(){ // 잡혔을 때 호출
    this.next_step = Block.STEP.GRABBED;
}
public void endGrab() // 놓았을 때 호출
{
    This.next_step = Block.STEP.IDLE;
}
public bool isGrabbable() // 잡을 수 있는 상태 인지 판단.
{
    bool is_grabbable = false;
    switch(this.step) {
    case Block.STEP.IDLE: // '대기' 상태일 때만.
        is_grabbable = true; // true(잡을 수 있다)를 반환한다.
        break;
    }
    return(is_grabbable);
}
```

```
public bool isContainedPosition(Vector2 position){
// 지정된 마우스 좌표가 자신과 겹치는 지 확인
    bool ret = false;
    Vector3 center = this.transform.position;
    float h = Block.COLLISION_SIZE / 2.0f;
    do {
    // X 좌표가 자신과 겹치지 않으면 break로 루프를 빠져 나간다.
    if(position.x < center.x - h || center.x + h < position.x) {
    break; }
    // Y 좌표가 자신과 겹치지 않으면 break로 루프를 빠져 나간다.
    if(position.y < center.y - h || center.y + h < position.y) {
    break;
    }
    // X 좌표, Y 좌표 모두 겹쳐 있으면 true(겹쳐 있다)를 반환한다.
    ret = true;
    } while(false);
    return(ret);
}
```

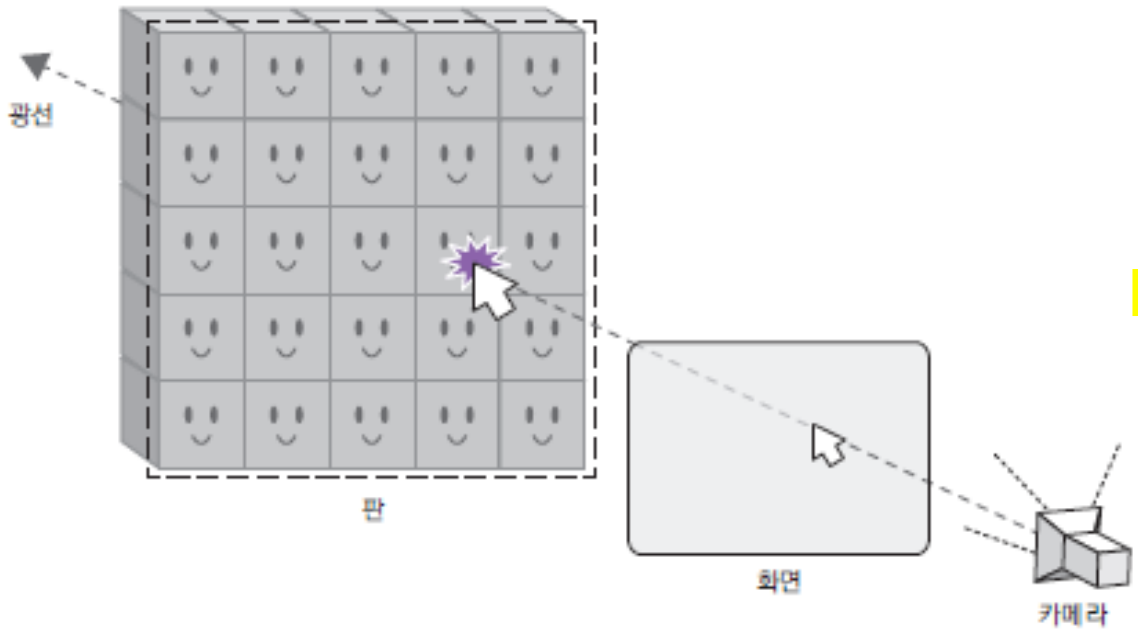
2. 마우스로 블록 잡기

BlockRoot.cs

```
// 블록을 잡는데 필요한 멤버 변수 선언
private GameObject main_camera = null; // 메인 카메라.
private BlockControl grabbed_block = null; // 잡은 블록.

void Start() {
    this.main_camera = GameObject.FindGameObjectWithTag("MainCamera");
    //카메라로부터 마우스 커서를 통과하는 광선을 쏘기 위해서 필요
}

void Update() {
    Vector3 mouse_position; // 마우스 위치.
    this.unprojectMousePosition( // 마우스 위치를 가져온다.
    out mouse_position, Input.mousePosition);
    // 가져온 마우스 위치를 하나의 Vector2로 모은다.
    Vector2 mouse_position_xy =
        new Vector2(mouse_position.x, mouse_position.y);
    if(this.grabbed_block == null) { // 잡은 블록이 비었으면.
    // if(!this.is_has_falling_block()) {      나중에 주석처리 벗긴다.
    if(Input.GetMouseButtonDown(0)) { // 마우스 버튼이 눌렸으면
    // blocks 배열의 모든 요소를 차례로 처리한다.
    foreach(BlockControl block in this.blocks) {
    if(! block.isGrabbable()) { // 블록을 잡을 수 없다면.
        continue; } // 루프의 처음으로 점프한다.
    } // 마우스 위치가 블록 영역 안이 아니면.
    if(!block.isContainedPosition(mouse_position_xy)) {
        continue; }// 루프의 처음으로 점프한다.
    // 처리 중인 블록을 grabbed_block에 등록한다.
    this.grabbed_block = block;
    // 잡았을 때의 처리를 실행한다.
    this.grabbed_block.beginGrab();
    break; } } // }
    } else { // 블록을 잡았을 때.
        if(! Input.GetMouseButton(0)) { // 마우스 버튼이 눌려져 있지 않으면.
            this.grabbed_block.endGrab(); // 블록을 놨을 때의 처리를 실행.
            this.grabbed_block = null; } // grabbed_block을 비우게 설정.
}
```

2. 마우스로 블록 잡기

그림 9-10 카메라와 마우스의 위치를 통과하는 광선이 판에 부딪히면 3D 공간의 위치를 계산해낼 수 있다



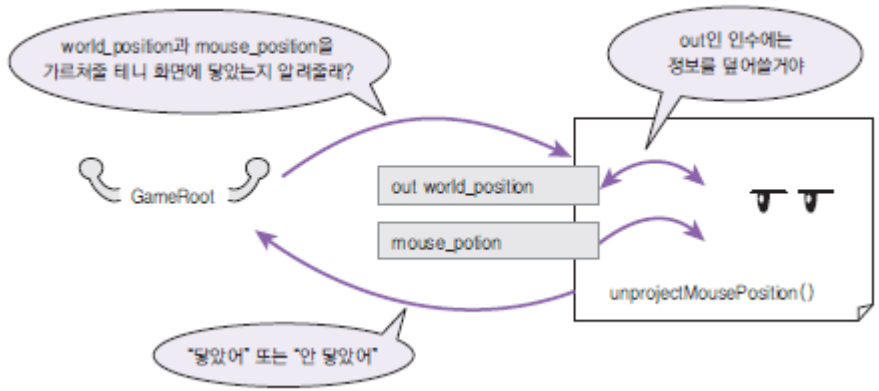
unprojectMousePosition( )

2. 마우스로 블록 잡기

BlockRoot.cs

```
public bool unprojectMousePosition(
out Vector3 world_position, Vector3 mouse_position)
{
    bool ret;
    // 판을 작성한다. 이 판은 카메라에 대해서 뒤로 향해서(Vector3.back).
    // 블록의 절반 크기만큼 앞에 둔다.
    Plane plane = new Plane(Vector3.back, new Vector3(
    0.0f, 0.0f, -Block.COLLISION_SIZE / 2.0f));
    // 카메라와 마우스를 통과하는 빛을 만든다.
    Ray ray = this.main_camera.GetComponent<Camera>().ScreenPointToRay(
    mouse_position);
    float depth;
    // 광선(ray)이 판(plane)에 닿았다면.
    if(plane.Raycast(ray, out depth)) { //광선이 닿았으면 depth에 정보가 기록된다
        // 인수 world_position을 마우스 위치로 덮어쓴다.
        world_position = ray.origin + ray.direction * depth;
        ret = true;
    // 닿지 않았다면.
    } else {
        // 인수 world_position을 0인 벡터로 덮어쓴다.
        world_position = Vector3.zero;
        ret = false;
    }
    return(ret); // 카메라를 통과하는 광선이 블록에 닿았는지를 반환
}
```

그림 9-11 out 사용법



3. 블록 교체

그림 9-15 마우스 왼쪽 클릭으로 블록을 잡아보자

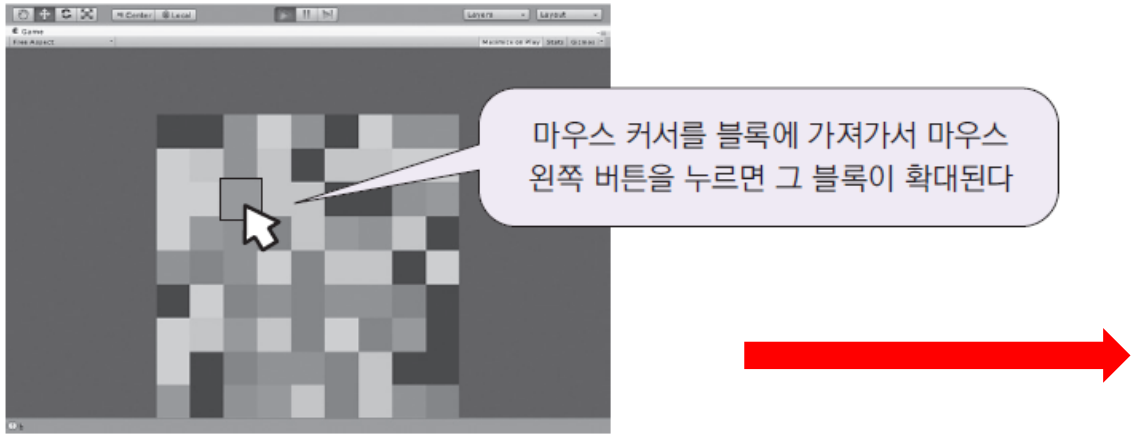


그림 9-16 이 절의 완성도 – 블록을 교체한다

지금까지의 프로그래밍을 통해 블록을 잡을 수 있게 되었다.

게임을 진행하기 위해서는 주변 블록을 교체할 수 있어야 한다.

3. 블록 교체

* calcSlideDir() : 인수로 지정한 마우스 위치를 바탕으로 어느 쪽으로 슬라이드 되었는 지 판단하고 그 방향을 Block.DIR4형 값으로 반환

BlockControl.cs

```
public float vanish_timer = -1.0f; // 블록이 사라질 때까지의 시간.
public Block.DIR4 slide_dir = Block.DIR4.NONE; // 슬라이드된 방향.
public float step_timer = 0.0f; // 블록이 교체된 때의 이동시간 등.

public Block.DIR4 calcSlideDir(Vector2 mouse_position){
    Block.DIR4 dir = Block.DIR4.NONE;
    // 지정된 mouse_position과 현재 위치의 차를 나타내는 벡터.
    Vector2 v = mouse_position -
    new Vector2(this.transform.position.x, this.transform.position.y);
    // 벡터의 크기가 0.1보다 크면.
    // (그보다 작으면 슬라이드하지 않은 걸로 간주한다).
if(v.magnitude > 0.1f) {
    if(v.y > v.x) {
        if(v.y > -v.x) {
            dir = Block.DIR4.UP;
        } else {
            dir = Block.DIR4.LEFT;
        }
    } else {
        if(v.y > -v.x) {
            dir = Block.DIR4.RIGHT;
        } else {
            dir = Block.DIR4.DOWN;
        }
    }
return(dir);
}
```

3. 블록 교체

* calcDirOffset(): 지정된 위치와 방향을 근거로 현재 위치와 슬라이드 할 곳의 거리가 어느 정도인지 반환한다.

BlockControl.cs

```
public float calcDirOffset(Vector2 position, Block.DIR4 dir){
    float offset = 0.0f;
    // 지정된 위치와 블록의 현재 위치의 차를 나타내는 벡터.
    Vector2 v = position - new Vector2(
    this.transform.position.x, this.transform.position.y);
    switch(dir) { // 지정된 방향에 따라 갈라진다.
        case Block.DIR4.RIGHT: offset = v.x;
        break;
        case Block.DIR4.LEFT: offset = -v.x;
        break;
        case Block.DIR4.UP: offset = v.y;
        break;
        case Block.DIR4.DOWN: offset = -v.y;
        break;
    }
    return(offset);}
```

BlockControl.cs

```
//이동 시작을 알리는 메서드
public void beginSlide(Vector3 offset)
{
    this.position_offset_initial= offset;
    this.position_offset = this.position_offset_initial;
    // 상태를 SLIDE로 변경.
    this.next_step = Block.STEP.SLIDE;
}
```

3. 블록 교체

* Update() 메서드 수정

BlockControl.cs

```
void Update() {
    Vector3 mouse_position;
    this.block_root.unprojectMousePosition(
    out mouse_position, Input.mousePosition);
    Vector2 mouse_position_xy =
    new Vector2(mouse_position.x, mouse_position.y);
    this.step_timer += Time.deltaTime;
    float slide_time = 0.2f;
    if(this.next_step == Block.STEP.NONE) { // '상태 정보 없음'의 경우.
        switch(this.step) {
          case Block.STEP.SLIDE:
            if(this.step_timer >= slide_time) {
            // 슬라이드 중인 블록이 소멸되면 VACANT(사라진) 상태로 이행.
                if(this.vanish_timer == 0.0f) {
                    this.next_step = Block.STEP.VACANT;
                // vanish_timer가 0이 아니면 IDLE(대기) 상태로 이행.}
    else {
        this.next_step = Block.STEP.IDLE; }}
        break; }}
// 변했을 때만 시행할 부분----------.
    while(this.next_step != Block.STEP.NONE) {
        this.step = this.next_step;
        this.next_step = Block.STEP.NONE;
        switch(this.step) {
            case Block.STEP.IDLE:
                this.position_offset = Vector3.zero;
                this.transform.localScale = Vector3.one * 1.0f;
                break;
            case Block.STEP.GRABBED:
                this.transform.localScale = Vector3.one * 1.2f;
                break;
            case Block.STEP.RELEASED:
                this.position_offset = Vector3.zero;
                this.transform.localScale = Vector3.one * 1.0f;
            break;
```

3. 블록 교체

* Update() 메서드 수정

BlockControl.cs

```
        case Block.STEP.VACANT:
            this.position_offset = Vector3.zero;
            break;}
        this.step_timer = 0.0f; }
    switch(this.step) {
        case Block.STEP.GRABBED: // 잡힌 상태.
        // 잡힌 상태일 때는 항상 슬라이드 방향을 체크.
            this.slide_dir = this.calcSlideDir(mouse_position_xy);
            break;
        case Block.STEP.SLIDE: // 슬라이드(교체) 중.
            // 블록을 서서히 이동하는 처리.
            // (어려운 부분이니 지금은 몰라도 괜찮다).
            float rate = this.step_timer / slide_time;
            rate = Mathf.Min(rate, 1.0f);
            rate = Mathf.Sin(rate*Mathf.PI / 2.0f);
            this.position_offset = Vector3.Lerp(
            this.position_offset_initial, Vector3.zero, rate);
            break; }
        Vector3 position = BlockRoot.calcBlockPosition(this.i_pos) +
        this.position_offset;
        this.transform.position = position;
    }
```

3. 블록 교체

* BlockRoot 스크립트에 새 메서드 추가

- getNextBlock( ): 블록이 슬라이드할 곳에 어느 블록이 있는지 반환한다.
  인수 - '현재 잡고 있는 블록'. '슬라이드 방향'
- getDirVector( ): 인수로 지정된 방향을 바탕으로 이동량의 벡터를 반환한다.
  현재 블록에서 지정 방향으로 이동하는 양 반환
- getOpposite( ): 인수로 지정된 방향의 반대 방향을 반환한다.
  블록을 서로 교체할 때, 이동할 곳에 있는 블록은 역방향 이동
- swapBlock( ): 실제로 블록을 교체한다.

3. 블록 교체

* getNextBlock( ) : 블록이 슬라이드할 곳에 어느 블록이 있는지 반환

BlockRoot.cs

```
public BlockControl getNextBlock(
    BlockControl block, Block.DIR4 dir){
    BlockControl next_block = null; // 슬라이드할 곳의 블록을 여기에 저장.
    switch(dir) {
    case Block.DIR4.RIGHT:
    if(block.i_pos.x < Block.BLOCK_NUM_X - 1) { // 그리드 안이라면.
        next_block = this.blocks[block.i_pos.x + 1, block.i_pos.y];}
    break;
    case Block.DIR4.LEFT:
    if(block.i_pos.x > 0) { // 그리드 안이라면.
        next_block = this.blocks[block.i_pos.x - 1, block.i_pos.y]; }
    break;
    case Block.DIR4.UP:
    if(block.i_pos.y < Block.BLOCK_NUM_Y - 1) { // 그리드 안이라면.
        next_block = this.blocks[block.i_pos.x, block.i_pos.y + 1];}
    break;
    case Block.DIR4.DOWN:
    if(block.i_pos.y > 0) { // 그리드 안이라면.
        next_block = this.blocks[block.i_pos.x, block.i_pos.y - 1];}
    break;
    }
return(next_block);
}
```

3. 블록 교체

* getDirVector( ) : 인수로 지정된 방향을 바탕으로 이동량의 벡터를 반환
* getOpposite( ) : 인수로 지정된 방향의 반대 방향을 반환

BlockRoot.cs

```
public static Vector3 getDirVector(Block.DIR4 dir)
{
    Vector3 v = Vector3.zero;
    switch(dir) {
    case Block.DIR4.RIGHT: v = Vector3.right; break; // 오른쪽으로 1단위 이동.
    case Block.DIR4.LEFT: v = Vector3.left; break; // 왼쪽으로 1단위 이동.
    case Block.DIR4.UP: v = Vector3.up; break; // 위로 1단위 이동.
    case Block.DIR4.DOWN: v = Vector3.down; break; // 아래로 1단위 이동.
    }
    v *= Block.COLLISION_SIZE; // 블록의 크기를 곱한다.
    return(v);
}
```

BlockRoot.cs

```
public static Block.DIR4 getOppositDir(Block.DIR4 dir)
{
    Block.DIR4 opposit = dir;
    switch(dir) {
        case Block.DIR4.RIGHT: opposit = Block.DIR4.LEFT; break;
        case Block.DIR4.LEFT: opposit = Block.DIR4.RIGHT; break;
        case Block.DIR4.UP: opposit = Block.DIR4.DOWN; break;
        case Block.DIR4.DOWN: opposit = Block.DIR4.UP; break;
    }
    return(opposit);
}
```

3. 블록 교체

* swapBlock( ) : 실제로 블록을 교체한다.

BlockRoot.cs

```
public void swapBlock(
BlockControl block0, Block.DIR4 dir, BlockControl block1)
{
    // 각각의 블록 색을 기억해 둔다.
    Block.COLOR color0 = block0.color;
    Block.COLOR color1 = block1.color;

    // 각각의 블록의 확대율을 기억해 둔다.
    Vector3 scale0 = block0.transform.localScale;
    Vector3 scale1 = block1.transform.localScale;

    // 각각의 블록의 '사라지는 시간'을 기억해 둔다.
    float vanish_timer0 = block0.vanish_timer;
    float vanish_timer1 = block1.vanish_timer;

    // 각각의 블록의 이동할 곳을 구한다.
    Vector3 offset0 = BlockRoot.getDirVector(dir);
    Vector3 offset1 = BlockRoot.getDirVector(BlockRoot.getOppositDir(dir));

    // 색을 교체한다.
    block0.setColor(color1);
    block1.setColor(color0);

    // 확대율을 교체한다.
    block0.transform.localScale = scale1;
    block1.transform.localScale = scale0;

    // '사라지는 시간'을 교체한다.
    block0.vanish_timer = vanish_timer1;
    block1.vanish_timer = vanish_timer0;
    block0.beginSlide(offset0); // 원래 블록 이동을 시작한다.
    block1.beginSlide(offset1); // 이동할 위치의 블록 이동을 시작한다.
}
```

3. 블록 교체

* update() 메서드에 교체 파트 추가

```
        } else {
            //-----------------------------------------
            // 여기에 do~while문을 추가한다!.
            //-----------------------------------------
            if(! Input.GetMouseButton(0)) {     // 마우스 버튼이 눌려져 있지 않으면.
                this.grabbed_block.endGrab();   // 블록을 놨을 때의 처리를 실행.
                this.grabbed_block = null;      // grabbed_block을 비우게 설정.
            }
        }
    }
```

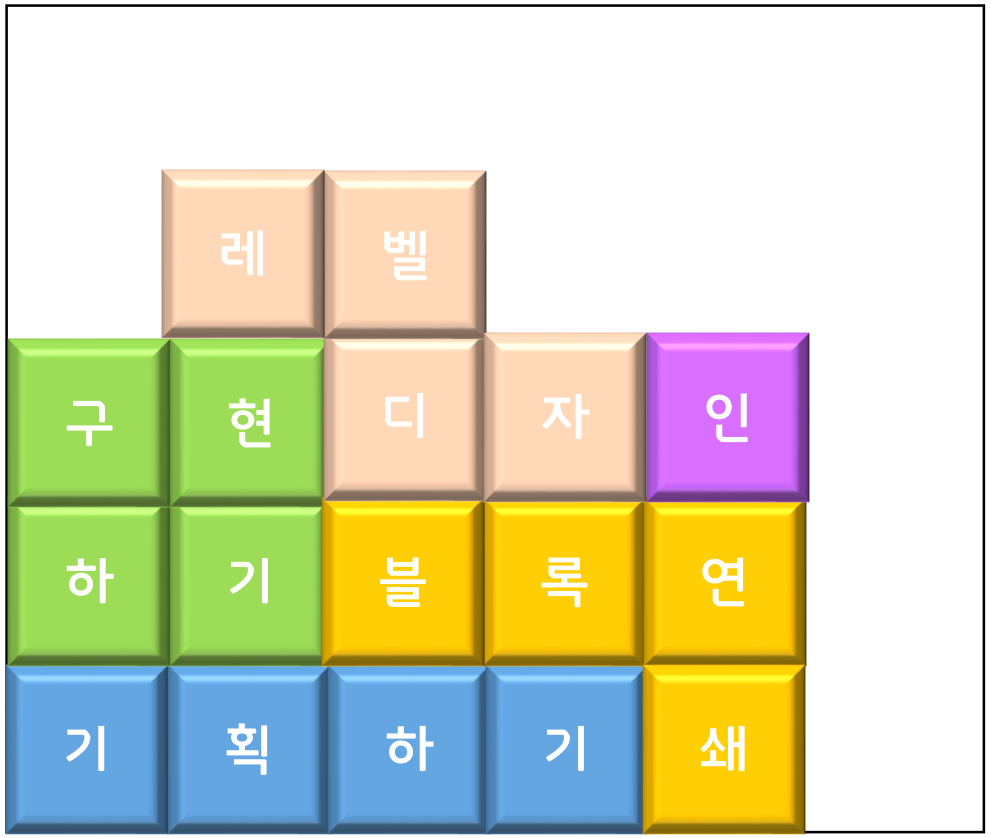
3. 블록 교체

* update() 메서드에 교체 파트 추가

BlockRoot.cs

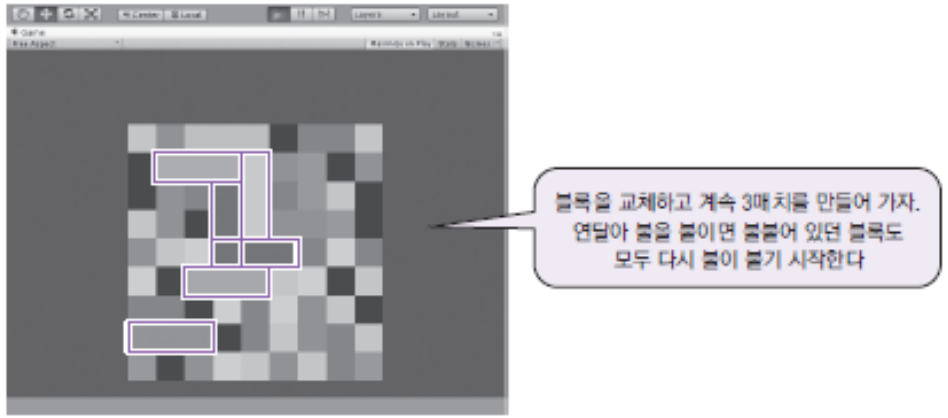
```
do {
    // 슬라이드할 곳의 블록을 가져온다.
    BlockControl swap_target =
    this.getNextBlock(grabbed_block, grabbed_block.slide_dir);
    // 슬라이드할 곳 블록이 비어 있으면.
    if(swap_target == null) {
    break; // 루프 탈출.
    }
    // 슬라이드할 곳의 블록이 잡을 수 있는 상태가 아니라면.
    if(! swap_target.isGrabbable()) {
        break; // 루프 탈출.
    }
    // 현재 위치에서 슬라이드 위치까지의 거리를 얻는다.
    float offset = this.grabbed_block.calcDirOffset(
    mouse_position_xy, this.grabbed_block.slide_dir);
    // 수리 거리가 블록 크기의 절반보다 작다면.
    if(offset < Block.COLLISION_SIZE / 2.0f) {
    break; } // 루프 탈출.
    // 블록을 교체한다.
    this.swapBlock(
    grabbed_block, grabbed_block.slide_dir, swap_target);
    this.grabbed_block = null; // 지금은 블록을 잡고 있지 않다.
} while(false);
```

퍼즐게임 만들기



1. 3매치로 블록 지우기

그림 10-1 이 절의 완성도 – 불을 붙인다



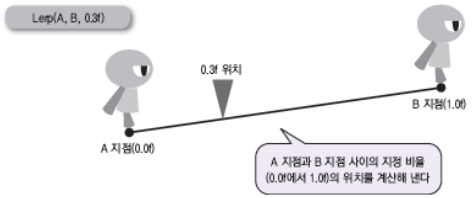
BlockControl.cs

```
public Material opague_material; // 불투명 머티리얼.
public Material transparent_material; // 반투명 머티리얼.

void Update() { //굵은 부분 추가
…
    Vector2 mouse_position_xy =
        new Vector2(mouse_position.x, mouse_position.y);
    if(this.vanish_timer >= 0.0f) { // 타이머가 0 이상이면.
        this.vanish_timer -= Time.deltaTime; // 타이머의 값을 줄인다.
        if(this.vanish_timer < 0.0f) { // 타이머가 0 미만이면.
            if(this.step != Block.STEP.SLIDE) { // 슬라이드 중이 아니라면.
                this.vanish_timer = -1.0f;
                this.next_step = Block.STEP.VACANT; // 상태를 '소멸 중'으로.
            } else {
                this.vanish_timer = 0.0f;
            }
        }
    }
…
```

1. 3매치로 블록 지우기

BlockControl.cs

```
void Update() { //굵은 부분 추가
...
    while(this.next_step != Block.STEP.NONE) {
    ...
    switch(this.step) {
        case Block.STEP.RELEASED:
            this.position_offset = Vector3.zero;
            this.transform.localScale = Vector3.one * 1.0f;
            break;
        case Block.STEP.VACANT:
            this.position_offset = Vector3.zero;
            this.setVisible(false); // 블록을 표시하지 않게 한다.
            break;
        }
    this.step_timer = 0.0f;
...
```




```
Vector3 position = BlockRoot.calcBlockPosition(this.i_pos) +
    this.position_offset;
    this.transform.position = position;
    this.setColor(this.color);
    if(this.vanish_timer >= 0.0f) {
        Color color0 = // 현재 색과 흰색의 중간 색.
        Color.Lerp(this.renderer.material.color, Color.white, 0.5f);
        Color color1 = // 현재 색과 검은색의 중간 색.
        Color.Lerp(this.renderer.material.color, Color.black, 0.5f);
        // 불붙는 연출 시간이 절반을 지났다면.
        if(this.vanish_timer < Block.VANISH_TIME / 2.0f) {
            // 투명도(a)를 설정.
            color0.a = this.vanish_timer / (Block.VANISH_TIME / 2.0f);
            color1.a = color0.a;
            // 반투명 머티리얼을 적용.
            this.renderer.material = this.transparent_material; }
    // vanish_timer가 줄어들수록 1에 가까워진다.
    float rate = 1.0f - this.vanish_timer / Block.VANISH_TIME;
    // 서서히 색을 바꾼다.
    this.renderer.material.color = Color.Lerp(color0, color1, rate);}}
```

1. 3매치로 블록 지우기

BlockControl.cs

```
public void toVanishing(){
    // '사라질 때까지 걸리는 시간'을 규정값으로 리셋.
    this.vanish_timer = Block.VANISH_TIME;}

public bool isVanishing(){
    // vanish_timer가 0보다 크면 true.
    bool is_vanishing = (this.vanish_timer > 0.0f);
    return(is_vanishing);}

public void rewindVanishTimer(){
    // '사라질 때까지 걸리는 시간'을 규정값으로 리셋.
    this.vanish_timer = Block.VANISH_TIME;}

public bool isVisible(){
    // 그리기 가능(renderer.enabled가 true) 상태라면.
    // 표시되고 있다.
    bool is_visible = this.renderer.enabled;
    return(is_visible);}

public void setVisible(bool is_visible){
    // 그리기 가능 설정에 인수를 대입.
    this.renderer.enabled = is_visible;}

public bool isIdle(){
    bool is_idle = false;
    // 현재 블록 상태가 '대기 중'이고.
    // 다음 블록 상태가 '없음'이면.
    if(this.step = = Block.STEP.IDLE &&
        this.next_step = = Block.STEP.NONE) {
            is_idle = true;}
    return(is_idle);
}
```

1. 3매치로 블록 지우기

BlockRoot.cs

```
void Update() {
    …
    if(! Input.GetMouseButton(0)) {
        …}}
    // 낙하 중 또는 슬라이드 중이면.
    if(this.is_has_falling_block() || this.is_has_sliding_block()) {
    // 아무것도 하지 않는다.
    // 낙하 중도 슬라이드 중도 아니면.
    } else {
        int ignite_count = 0; // 불붙은 개수.
        // 그리드 안의 모든 블록에 대해서 처리.
        foreach(BlockControl block in this.blocks) {
            if(! block.isIdle()) { // 대기 중이면 루프의 처음으로 점프하고.
            continue; } // 다음 블록을 처리한다.
        // 세로 또는 가로에 같은 색 블록이 세 개 이상 나열했다면.
        if(this.checkConnection(block)) {
            ignite_count++; // 불붙은 개수를 증가.
        }}
        if(ignite_count > 0) { // 불붙은 개수가 0보다 크면.
        // =한 군데라도 맞춰진 곳이 있으면.
            int block_count = 0; // 불붙는 중인 블록 수(다음 장에서 사용한다).
            // 그리드 내의 모든 블록에 대해서 처리.
            foreach(BlockControl block in this.blocks) {
                if(block.isVanishing()) { // 타는 중이면.
                    block.rewindVanishTimer(); // 다시 점화!.
                }
            }
        }
    }
}
```

1. 3매치로 블록 지우기

BlockRoot.cs

```
// 인수로 받은 블록이 세 개의 블록 안에 들어가는 지 파악하는 메서드
public bool checkConnection(BlockControl start) {
  bool ret = false;
  int normal_block_num = 0;
  // 인수인 블록이 불붙은 다음이 아니면.
  if(! start.isVanishing()) {
    normal_block_num = 1; }

  // 그리드 좌표를 기억해 둔다.
  int rx = start.i_pos.x;
  int lx = start.i_pos.x;
```

```
// 블록의 왼쪽을 검사.
 for(int x = lx - 1; x > 0; x--) {
  BlockControl next_block = this.blocks[x, start.i_pos.y];
  if(next_block.color != start.color) { // 색이 다르면.
    break;} // 루프를 빠져나간다.

  if(next_block.step = = Block.STEP.FALL || // 낙하 중이면.
    next_block.next_step = = Block.STEP.FALL) {
    break;}

  if(next_block.step = = Block.STEP.SLIDE || // 슬라이드 중이면.
    next_block.next_step = = Block.STEP.SLIDE) {
    break;}

  if(! next_block.isVanishing()) { // 불붙은 상태가 아니면.
    normal_block_num++;} // 검사용 카운터를 증가.

  lx = x; }
```

1. 3매치로 블록 지우기

BlockRoot.cs

```
// 블록의 오른쪽을 검사.
  for(int x = rx + 1; x < Block.BLOCK_NUM_X; x+ +) {
    BlockControl next_block = this.blocks[x, start.i_pos.y];
    if(next_block.color != start.color) {
      break; }

    if(next_block.step = = Block.STEP.FALL ||
      next_block.next_step = = Block.STEP.FALL) {
      break; }

    if(next_block.step = = Block.STEP.SLIDE ||
      next_block.next_step = = Block.STEP.SLIDE) {
      break; }

    if(! next_block.isVanishing()) {
      normal_block_num++; }

    rx = x; }

do {
  // 오른쪽 블록의 그리드 번호 - 왼쪽 블록의 그리드 번호 +.
  // 중앙 블록(1)을 더한 수가 3 미만이면.
  if(rx - lx + 1 < 3) {
    break; } // 루프 탈출.

  if(normal_block_num = = 0) { // 불붙지 않은 블록이 하나도 없으면.
    break; }

  for(int x = lx; x < rx + 1; x+ +) {
  // 나열된 같은 색 블록을 불붙은 상태로.
    this.blocks[x, start.i_pos.y].toVanishing();
    ret = true; }
  } while(false);
```

1. 3매치로 블록 지우기

BlockRoot.cs

```
normal_block_num = 0;
 if(! start.isVanishing()) {
   normal_block_num = 1; }

 int uy = start.i_pos.y;
 int dy = start.i_pos.y;
```

```
// 블록의 위쪽을 검사.
 for(int y = dy - 1; y > 0; y--) {
   BlockControl next_block = this.blocks[start.i_pos.x, y];
   if(next_block.color != start.color) {
     break; }

   if(next_block.step = = Block.STEP.FALL ||
     next_block.next_step = = Block.STEP.FALL) {
     break; }

   if(next_block.step = = Block.STEP.SLIDE ||

     next_block.next_step = = Block.STEP.SLIDE) {
     break; }

   if(! next_block.isVanishing()) {
     normal_block_num++; }

   dy = y; }
```

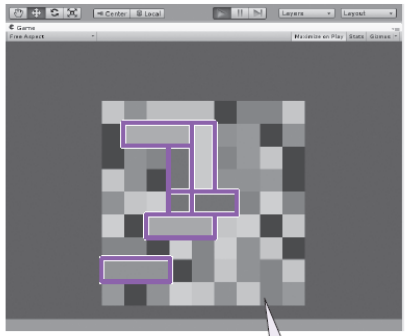
1. 3매치로 블록 지우기

BlockRoot.cs

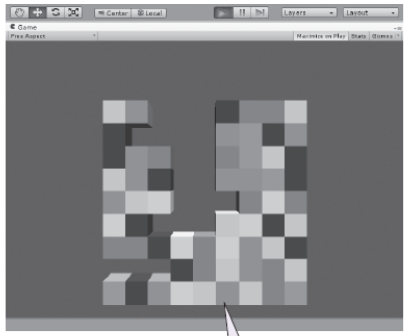
```
// 블록의 아래쪽을 검사.
  for(int y = uy + 1; y < Block.BLOCK_NUM_Y; y + +) {
    BlockControl next_block = this.blocks[start.i_pos.x, y];
    if(next_block.color != start.color) {
      break; }

    if(next_block.step = = Block.STEP.FALL ||
      next_block.next_step = = Block.STEP.FALL) {
      break; }

    if(next_block.step = = Block.STEP.SLIDE ||
      next_block.next_step = = Block.STEP.SLIDE) {
      break; }

    if(! next_block.isVanishing()) {
      normal_block_num++;}
    uy = y;
  }
```

```
do {
    if(uy - dy + 1 < 3) {
    break; }

    If(normal_block_num == 0) {
      bre

    for(ir
      this
      ret
    }
  } while

  return(
```

그림 10-4 3매치를 완성해 블록을 지우자



블록을 교체해서 계속 3매치를 만들어 가자. 연속 교체 불붙이기하면 불에 타던 블록도 다시 점화된다!

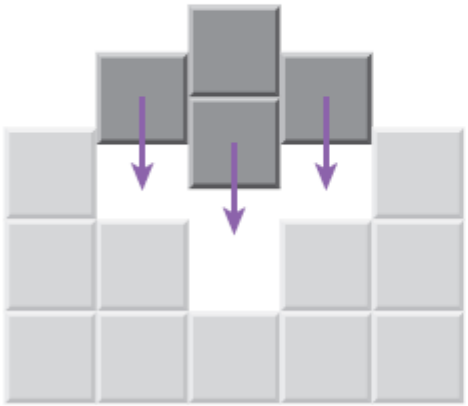


사라진 블록은 원래대로 돌아오지 않는다! 이래서는 게임이 진행될 수 없다!

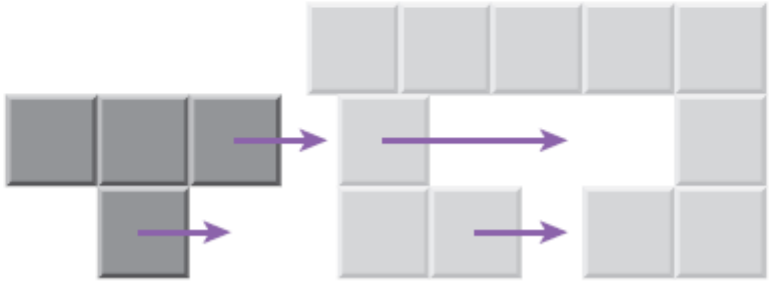
2. 블록 보충하기

그림 10-5 다양한 블록 보충 방법

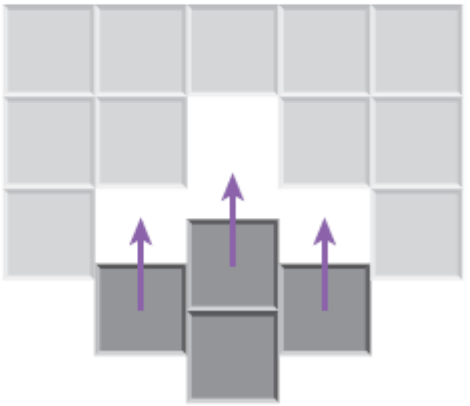
이번에 채용한 흔히 있는
'위에서 내려오는 형태'!



왼쪽이나 오른쪽에서 채워져가는
방법도 조금 재미있을 것 같다



게임의 진행에 맞춰 왼쪽이나
아래에서 채워지면?



2. 블록 보충하기

BlockControl.cs

```
private struct StepFall {
    public float velocity; // 낙하 속도.
}
private StepFall fall;
void Update() {
    ...
    if(this.next_step = = Block.STEP.NONE) {
      ...
        case Block.STEP.IDLE:
            this.renderer.enabled = true;
            break;
        case Block.STEP.FALL:
            if(this.position_offset.y <= 0.0f) {
                this.next_step = Block.STEP.IDLE;
                this.position_offset.y = 0.0f;
            }
            break;}}
...
```

```
while(this.next_step != Block.STEP.NONE) {
    ...
    switch(this.step) {
    ...
    case Block.STEP.VACANT:
        ...
    case Block.STEP.RESPAWN:
    // 색을 랜덤하게 선택하여 블록을 그 색으로 설정.
        int color_index = Random.Range(
            0, (int)Block.COLOR.NORMAL_COLOR_NUM);
            this.setColor((Block.COLOR)color_index);
            this.next_step = Block.STEP.IDLE;
            break;
    case Block.STEP.FALL:
        this.setVisible(true); // 블록을 표시.
        this.fall.velocity = 0.0f; // 낙하 속도 리셋.
            break;}
    this.step_timer = 0.0f;
}
...
```

2. 블록 보충하기

BlockControl.cs

```
void Update() {
    ...
    switch(this.step) {
        ...
        case Block.STEP.SLIDE:
        float rate = this.step_timer / slide_time;

        case Block.STEP.FALL:
        // 속도에 중력의 영향을 부여한다.
        this.fall.velocity += Physics.gravity.y * Time.deltaTime * 0.3f;
        // 세로 방향 위치를 계산.
        this.position_offset.y += this.fall.velocity * Time.deltaTime;
        if(this.position_offset.y < 0.0f) { // 다 내려왔다면.
            this.position_offset.y = 0.0f; // 그 자리에 머무른다.
        }
    break;
}
...
```

2. 블록 보충하기

BlockRoot.cs

```
public void fallBlock(
    BlockControl block0, Block.DIR4 dir, BlockControl block1)
{
    // block0과 block1의 색, 크기, 사라질 때까지 걸리는 시간, 표시, 비표시, 상태를 기록.
    Block.COLOR color0 = block0.color;
    Block.COLOR color1 = block1.color;
    Vector3 scale0 = block0.transform.localScale;
    Vector3 scale1 = block1.transform.localScale;
    float vanish_timer0 = block0.vanish_timer;
    float vanish_timer1 = block1.vanish_timer;
    bool visible0 = block0.isVisible();
    bool visible1 = block1.isVisible();
    Block.STEP step0 = block0.step;
    Block.STEP step1 = block1.step;
    // block0과 block1의 각종 속성을 교체.
    block0.setColor(color1);
    block1.setColor(color0);
    block0.transform.localScale = scale1;
    block1.transform.localScale = scale0;
    block0.vanish_timer = vanish_timer1;
    block1.vanish_timer = vanish_timer0;
    block0.setVisible(visible1);
    block1.setVisible(visible0);
    block0.step = step1;
    block1.step = step0;
    block0.beginFall(block1);
}
```

```
private bool is_has_sliding_block_in_column(int x)
{
    bool ret = false;
    for(int y = 0; y < Block.BLOCK_NUM_Y; y++) {
        if(this.blocks[x, y].isSliding()) {      // 슬라이드 중인 블록이 있으면.
            ret = true;                          // true를 반환.
            break;
        }
    }
    return(ret);
}

void Update() {
    Vector3 mouse_position;
    this.unprojectMousePosition(out mouse_position, Input.mousePosition);
    Vector2 mouse_position_xy =
        new Vector2(mouse_position.x, mouse_position.y);

    if(this.grabbed_block == null) {
        if(!this.is_has_falling_block()) {
            if(Input.GetMouseButtonDown(0)) {
                ...
            }
        }
    } else {
    ...
```

주석 처리를 벗겨낸다

2. 블록 보충하기

BlockRoot.cs

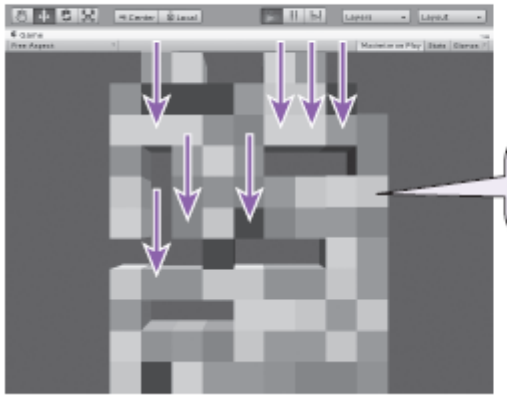
```
void Update() {
    ...
        if(ignite_count > 0) {

    ...
    // 하나라도 연소 중인 블록이 있는가?.
    bool is_vanishing = this.is_has_vanishing_block();
    // 조건이 만족되면 블록을 떨어뜨리고 싶다.
    do {
        if(is_vanishing) { // 연소 중인 블록이 있다면.
            break;} // 낙하 처리를 실행하지 않는다.
        if(this.is_has_sliding_block()) { // 교체 중인 블록이 있다면.
            break; } // 낙하 처리를 실행하지 않는다.
        for(int x = 0; x < Block.BLOCK_NUM_X; x++) {
            // 열에 교체 중인 블록이 있다면 그 열은 처리하지 않고
            // 다음 열로 진행한다. 주석 처리를 벗겨낸다
            if(this.is_has_sliding_block_in_column(x)) {
                continue;}
            // 그 열에 있는 블록을 위에서부터 검사한다.
            for(int y = 0; y < Block.BLOCK_NUM_Y - 1; y++) {
                // 지정 블록이 비표시라면 다음 블록으로.
                if(! this.blocks[x, y].isVacant()) {
                    continue; }
                // 지정 블록 아래에 있는 블록을 검사.
                for(int y1 = y + 1; y1 < Block.BLOCK_NUM_Y; y1++) {
                    // 아래에 있는 블록이 비표시라면 다음 블록으로.
                    if(this.blocks[x, y1].isVacant()) {
                        continue; }

                // 블록을 교체한다.
                this.fallBlock(this.blocks[x, y], Block.DIR4.UP,
                    this.blocks[x, y1]);
                break;
                }
            }
        }
```

2. 블록 보충하기

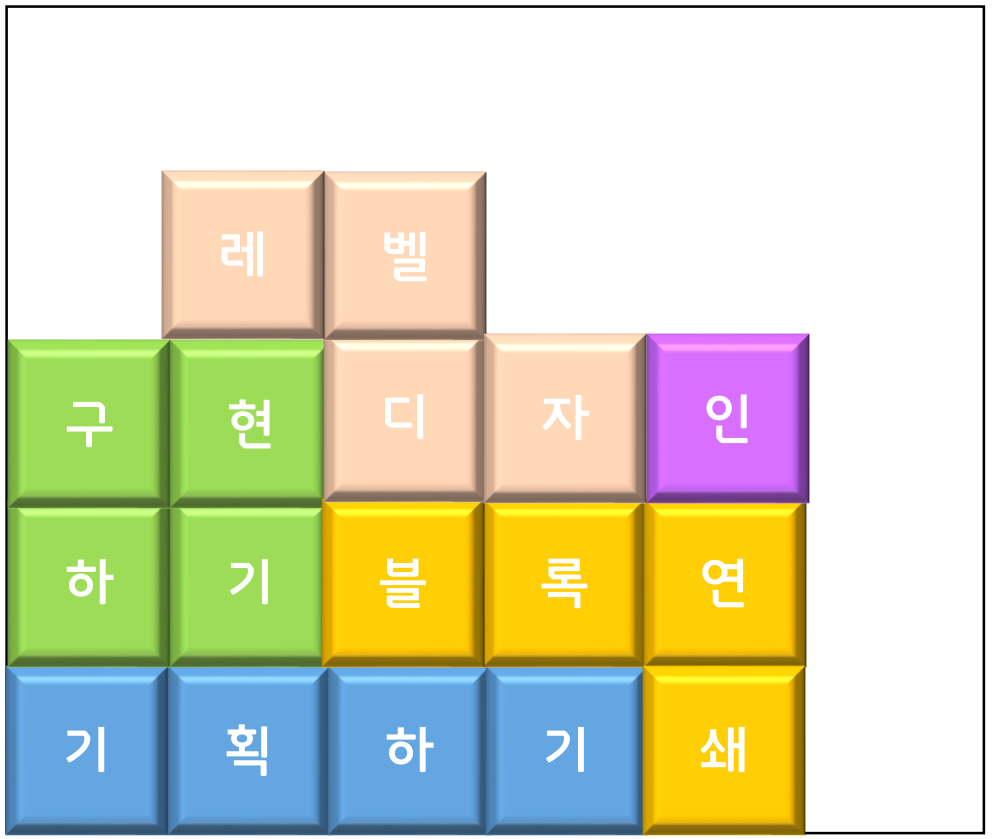
BlockRoot.cs

```
    // 보충처리.
    for(int x = 0; x < Block.BLOCK_NUM_X; x++) {
        int fall_start_y = Block.BLOCK_NUM_Y;
        for(int y = 0; y < Block.BLOCK_NUM_Y; y++) {
            // 비표시 블록이 아니라면 다음 블록으로.
            if(! this.blocks[x, y].isVacant()) {
                continue;
            }
            this.blocks[x, y].beginRespawn(fall_start_y); // 블록 부활.
            fall_start_y++;
            }
        }
    } while(false);
}
```

그림 10-7 블록이 추가된다



퍼즐게임 만들기



1. 점수와 시간 삽입

ScoreCounter.cs

```
public struct Count {                   // 점수 관리용 구조체.
    public int ignite;                  // 연쇄 수.
    public int score;                   // 점수.
    public int total_socre;             // 합계 점수.
};

public Count last;                      // 마지막(이번) 점수.
public Count best;                      // 최고 점수.

public static int QUOTA_SCORE = 1000;   // 클리어에 필요한 점수.
public GUIStyle guistyle;               // 폰트 스타일.

void Start() {    // 멤버변수 초기화
    this.last.ignite = 0;
    this.last.score = 0;
    this.last.total_socre = 0;
    this.guistyle.fontSize = 16;
}
```

1. 점수와 시간 삽입

ScoreCounter.cs

```
public void print_value(int x, int y, string label, int value)
{ // 지정된 데이터를 두 개의 행에 나눠 표시
      // label을 표시.
      GUI.Label(new Rect(x, y, 100, 20), label, guistyle);
      y += 15;

      // 다음 행에 value를 표시.
      GUI.Label(new Rect(x + 20, y, 100, 20), value.ToString(), guistyle);
      y += 15;
}

public void addIgniteCount(int count){
      this.last.ignite += count; // 연쇄 수에 count를 합산.
      this.update_score(); // 점수 계산.
}

public void clearIgniteCount(){
      this.last.ignite = 0; // 연쇄 횟수 리셋.
}

private void update_score(){
      this.last.score = this.last.ignite * 10; // 점수 갱신.
}
```

```
public void print_value(int x, int y, string label, int value)
public void updateTotalScore()
{
      this.last.total_socre += this.last.score; // 합계 점수 갱신.
}

public bool isGameClear()
{
      bool is_clear = false;
      // 현재 합계 점수가 클리어 기준보다 크면.
      if(this.last.total_socre > QUOTA_SCORE) {
            is_clear = true;
      }
      return(is_clear);
}
```

1. 점수와 시간 삽입

SceneControl.cs

```
private ScoreCounter score_counter = null;

public enum STEP {
    NONE = -1,                  // 상태 정보 없음.
    PLAY = 0,                   // 플레이 중.
    CLEAR,                      // 클리어.
    NUM,                        // 상태의 종류가 몇 개인지 나타냄(= 2).
};

public STEP step = STEP.NONE;       // 현재 상태.
public STEP next_step = STEP.NONE;  // 다음 상태.
public float step_timer = 0.0f;     // 경과 시간.
private float clear_time = 0.0f;    // 클리어 시간.
public GUIStyle guistyle;           // 폰트 스타일.

void Start() {
    this.block_root = this.gameObject.GetComponent<BlockRoot>();
    this.block_root.initialSetUp();

    // ScoreCounter 가져오기.
    this.score_counter = this.gameObject.GetComponent<ScoreCounter>();
    this.next_step = STEP.PLAY;         // 다음 상태를 '플레이 중'으로.
    this.guistyle.fontSize = 24;        // 폰트 크기를 24로.
}
```

```
void Update() {
    this.step_timer += Time.deltaTime;

    // 상태 변화 대기 -----.
    if(this.next_step == STEP.NONE) {
        switch(this.step) {
            case STEP.PLAY:
                // 클리어 조건을 만족하면.
                if(this.score_counter.isGameClear()) {
                    this.next_step = STEP.CLEAR;  // 클리어 상태로 이행.
                }
                break;
        }
    }

    // 상태가 변했다면 -----.
    while(this.next_step != STEP.NONE) {
        this.step = this.next_step;
        this.next_step = STEP.NONE;
        switch(this.step) {
            case STEP.CLEAR:
                // block_root를 정지.
                this.block_root.enabled = false;
                // 경과 시간을 클리어 시간으로 설정.
                this.clear_time = this.step_timer;
                break;
        }
        this.step_timer = 0.0f;
    }
}
```

1. 점수와 시간 삽입

SceneControl.cs

```
void OnGUI()    // 화면에 클리어한 시간과 메시지를 표시
{
    switch(this.step) {
    case STEP.PLAY:
        GUI.color = Color.black;
        // 경과 시간을 표시.
        GUI.Label(new Rect(40.0f, 10.0f, 200.0f, 20.0f),
            "시간" + Mathf.CeilToInt(this.step_timer).ToString() + "초",
            guistyle);
        GUI.color = Color.white;
        break;

    case STEP.CLEAR:
        GUI.color = Color.black;
        // 「☆클리어-!☆」라는 문자열을 표시.
        GUI.Label(new Rect(
            Screen.width/2.0f - 80.0f, 20.0f, 200.0f, 20.0f),
            "☆클리어-!☆", guistyle);
        // 클리어 시간을 표시.
        GUI.Label(new Rect(
            Screen.width/2.0f - 80.0f, 40.0f, 200.0f, 20.0f),
            "클리어 시간" + Mathf.CeilToInt(this.clear_time).ToString() +
            "초", guistyle);
        GUI.color = Color.white;
        break;
    }
}
```

1. 점수와 시간 삽입

BlackRoot.cs 점수에 관련된 부분을 변경

```
  private ScoreCounter score_counter = null;        // ScoreCounter
  protected bool is_vanishing_prev = false;         // 앞에서 점화했는가?.

  void Start() {
      this.main_camera = GameObject.FindGameObjectWithTag("MainCamera");
      this.score_counter = this.gameObject.GetComponent<ScoreCounter>();
  }
```

```
void Update() {
    ...
    } while(false);
    this.is_vanishing_prev = is_vanishing;
}
```

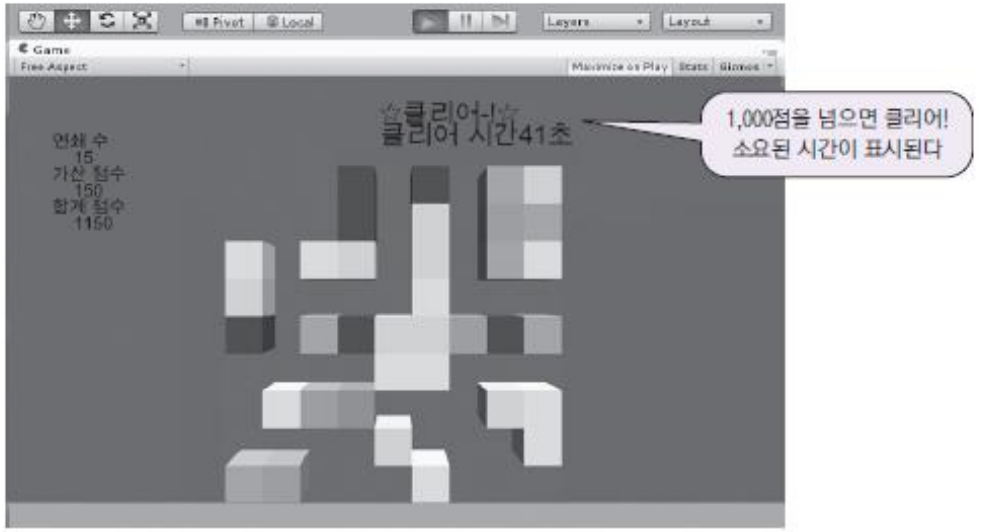
불타는 중엔 리셋되지 않는다 = 연쇄 수가 된다

```
void Update() {
    ...
    if(ignite_count > 0) {
        if(! this.is_vanishing_prev) {
            // 연속 점화가 아니라면 연쇄 횟수를 리셋.
            this.score_counter.clearIgniteCount();
        }
        // 연쇄 횟수 증가.
        this.score_counter.addIgniteCount(ignite_count);
        // 합계 점수 갱신.
        this.score_counter.updateTotalScore();

        int block_count = 0;
        foreach(BlockControl block in this.blocks) {
            if(block.isVanishing()) {
                block.rewindVanishTimer();
                block_count++;    // 불붙은 블록 개수 증가.
            }
        }
    }
...
}
```

연쇄된 블록 수만큼 점수 계산

1. 점수와 시간 삽입

* 완성화면 *



2. 텍스트 읽어오기

LevelControl.cs

```
using System.Collections.Generic;  // List를 사용하기 위함

public class LevelData {
    public float[] probability;     // 블록의 출현빈도를 저장하는 배열.
    public float heat_time;         // 연소 시간.

    public LevelData()              // 생성자.
    {
        // 블록의 종류 수와 같은 크기로 데이터 영역을 확보.
        this.probability = new float[(int)Block.COLOR.NORMAL_COLOR_NUM];

        // 모든 종류의 출현 확률을 균등하게 해둠.
        for(int i = 0; i < (int)Block.COLOR.NORMAL_COLOR_NUM; i++) {
            this.probability[i] =
                1.0f / (float)Block.COLOR.NORMAL_COLOR_NUM;
        }
    }

    // 모든 종류의 출현 확률을 0으로 리셋하는 메서드.
    public void clear()
    {
        for(int i = 0;i < this.probability.Length; i++) {
            this.probability[i] = 0.0f;
        }
    }

    // 모든 종류의 출현 확률의 합계를 100%(=1.0)로 하는 메서드.
    public void normalize()
    {
        float sum = 0.0f;

        // 출현 확률의 '임시 합계값'을 계산한다.
        for(int i = 0; i < this.probability.Length; i++) {
            sum += this.probability[i];
        }

        for(int i = 0;i < this.probability.Length; i++) {
            // 각 출현 확률을 '임시 합계값'으로 나누면 합계가 100%(=1.0)로 딱 떨어짐.
            this.probability[i] /= sum;

            // 만약 그 값이 무한대라면.
            if(float.IsInfinity(this.probability[i])) {
                this.clear();                   // 모든 확률을 0으로 리셋하고.
                this.probability[0] = 1.0f;     // 최초의 요소만 1.0으로 해둔다.
                break;                          // 그리고 루프를 빠져나간다.
            }
        }
    }
}
```

2. 텍스트 읽어오기

* LevelControl 클래스에 필요한 메서드

- level_datas: 각 레벨의 레벨 데이터를 저장하는 List 값
- select_level: 선택된 레벨을 저장하는 int 값
- initialize( ): 초기화 처리를 한다(level_datas를 초기화할 뿐).
- loadLevelData( ): 텍스트 데이터를 읽어와서 그 내용을 해석하고 데이터를 보관한다.
- selectLevel( ): 몇 개의 레벨 패턴에서 지금 사용할 패턴을 선택한다.
- getCurrentLevelData( ): 선택되어 있는 레벨 패턴의 레벨 데이터를 반환한다.
- getVanishingTime( ): 선택되어 있는 레벨 패턴의 연소 시간을 반환한다.

2. 텍스트 읽어오기

LevelControl.cs

```
public class LevelControl {
    private List<LevelData> level_datas = null;  // 각 레벨의 레벨 데이터.
    private int select_level = 0;                // 선택된 레벨.

    public void initialize()
    {
        // List를 초기화한다.
        this.level_datas = new List<LevelData>();   // 각 레벨에 데이터를 저장
    }

    public void loadLevelData(
        TextAsset level_data_text)
    {
        // 텍스트 데이터를 문자열로서 받아들인다.
        string level_texts = level_data_text.text;

    // 개행 코드'\'마다 나누어 문자열 배열에 집어넣는다.

string[] lines = level_texts.Split('\n');

    // lines 안의 각 행에 대하여 차례로 처리해가는 루프.
    foreach(var line in lines) {
        if(line == "") {  // 행이 비었으면.
            continue;     // 아래 처리는 하지 않고 루프의 처음으로 점프한다.
        }
        string[] words = line.Split();  // 행 내의 워드를 배열에 저장.
        int n = 0;

// LevelData 변수를 작성한다.
// 여기에 현재 처리하는 행의 데이터를 넣는다.
LevelData level_data = new LevelData();

// words 내의 각 워드에 대해서 순서대로 처리해 가는 루프.
foreach(var word in words) {

    if(word.StartsWith("#")) { // 워드의 시작 문자가 #이면.
        break;                 // 루프 탈출.
    }
    if(word == "") {           // 워드가 비었으면.
        continue;              // 루프 시작으로 점프.
    }

    // 'n'의 값을 0,1,2,...6으로 변화시켜 일곱 개 항목을 처리한다.
    // 각 워드를 float값으로 변환하고 level_data에 저장.
    switch(n) {
        case 0:
            level_data.probability[(int)Block.COLOR.PINK] =
                float.Parse(word); break;
        case 1:
            level_data.probability[(int)Block.COLOR.BLUE] =
                float.Parse(word); break;
        case 2:
            level_data.probability[(int)Block.COLOR.GREEN] =
                float.Parse(word); break;
```

2. 텍스트 읽어오기

LevelControl.cs

```
            case 3:
                level_data.probability[(int)Block.COLOR.ORANGE] =
                    float.Parse(word); break;
            case 4:
                level_data.probability[(int)Block.COLOR.YELLOW] =
                    float.Parse(word); break;
            case 5:
                level_data.probability[(int)Block.COLOR.MAGENTA] =
                    float.Parse(word); break;
            case 6:
                level_data.heat_time =
                    float.Parse(word); break;
        }
        n++;
    }

    if(n >= 7) {  // 8항목(이상)이 제대로 처리되었다면.
        // 출현 확률의 합계가 정확히 100%가 되도록 하고 나서.
        level_data.normalize();
        // List 구조의 level_datas에 level_data를 추가한다.
        this.level_datas.Add(level_data);
    } else {              // 그렇지 않으면(오류 가능성이 있다).
        if(n == 0) {      // 1워드도 처리하지 않은 경우는 주석이므로.
                          // 문제없음. 아무것도 하지 않는다.
        } else {          // 그 이외라면 오류.
                          // 데이터의 개수가 맞지 않는다는 오류 메시지를 표시한다.
            Debug.LogError("[LevelData] Out of parameter.\n");
        }
    }
}

// level_datas에 데이터가 하나도 없으면.
if(this.level_datas.Count == 0) {
    // 오류 메시지를 표시한다.
    Debug.LogError("[LevelData] Has no data.\n");
    // level_datas에 LevelData를 하나 추가해 둔다.
    this.level_datas.Add(new LevelData());
}
}

public void selectLevel()
{
    // 0~패턴 사이의 값을 임의로 선택한다.
    this.select_level = Random.Range(0, this.level_datas.Count);
    Debug.Log("select level = " + this.select_level.ToString());
}
public LevelData getCurrentLevelData()
{
    // 선택된 패턴의 레벨 데이터를 반환한다.
    return(this.level_datas[this.select_level]);
}

public float getVanishTime()
{
    // 선택된 패턴의 연소 시간을 반환한다.
    return(this.level_datas[this.select_level].heat_time);
}
}
```

2. 텍스트 읽어오기

* BlackRoot 클래스에 필요한 메서드

- levelData: 레벨 데이터의 텍스트를 저장한다.
- level_control: LevelControl을 저장한다.
- create( ): 레벨 데이터의 초기화, 로드, 패턴 설정까지 시행한다.
- selectBlockColor( ): 현재 패턴의 출현 확률을 바탕으로 색을 산출해서 반환한다.

2. 텍스트 읽어오기

BlackRoot.cs

```
public TextAsset levelData = null;      // 레벨 데이터의 텍스트를 저장.
public LevelControl level_control;      // LevelControl을 저장.

public void create()
{
    this.level_control = new LevelControl();
    this.level_control.initialize();                    // 레벨 데이터 초기화.
    this.level_control.loadLevelData(this.levelData);   // 데이터 읽기.
    this.level_control.selectLevel();                   // 레벨 선택.
}

public Block.COLOR selectBlockColor()
{
    Block.COLOR color = Block.COLOR.FIRST;

    // 이번 레벨의 레벨 데이터를 가져온다.
    LevelData level_data =
        this.level_control.getCurrentLevelData();

    float rand = Random.Range(0.0f, 1.0f);      // 0.0~1.0 사이의 난수.
    float sum = 0.0f;                           // 출현 확률의 합계.
    int i = 0;

    // 블록의 종류 전체를 처리하는 루프.
    for(i = 0; i < level_data.probability.Length - 1; i++) {
        if(level_data.probability[i] == 0.0f) {
            continue;    // 출현 확률이 0이면 루프의 처음으로 점프.
        }
        sum += level_data.probability[i];   // 출현 확률을 더한다.
        if(rand < sum) {                    // 합계가 난숫값을 웃돌면.
            break;                          // 루프를 빠져나온다.
        }
    }

    color = (Block.COLOR)i;                 // i번째 색을 반환한다.
    return(color);
}
```

2. 텍스트 읽어오기

LevelControl.cs

나열할 초기 배치 블록도 선택된 레벨의
출현 패턴을 따르게 하는 수정

```
public void initialSetUp()
{
    this.blocks =
        new BlockControl [Block.BLOCK_NUM_X, Block.BLOCK_NUM_Y];
    int color_index = 0;
    Block.COLOR color = Block.COLOR.FIRST;

    for(int y = 0; y < Block.BLOCK_NUM_Y; y++) {
        for(int x = 0; x < Block.BLOCK_NUM_X; x++) {
            ...
            Vector3 position = BlockRoot.calcBlockPosition(block.i_pos);
            block.transform.position = position;

            // block.setColor((Block.COLOR)color_index); ———— 주석 처리 혹은 삭제한다

            // 현재 출현 확률을 바탕으로 색을 결정한다.
            color = this.selectBlockColor();
            block.setColor(color);
            block.name = "block(" + block.i_pos.x.ToString() + "," +
                block.i_pos.y.ToString() + ")";
            color_index = Random.Range(
                0, (int)Block.COLOR.NORMAL_COLOR_NUM);
        }
    }
}
```

2. 텍스트 읽어오기

SceneControl.cs

```
void Start() {
    this.block_root = this.gameObject.GetComponent<BlockRoot>();
    this.block_root.create(); ———— create() 메서드에서 초기 설정
    this.block_root.initialSetUp();
    this.score_counter = this.gameObject.GetComponent<ScoreCounter>();
    this.next_step = STEP.PLAY;
    this.guistyle.fontSize = 24;
}
```

씬이 시작될 때 텍스트 데이터를 읽고 레벨 선택을 할 수 있게 함

BlockControl.cs

```
void Update() {          // 레벨 데이터의 연소 시간이 반영되도록 함
    ...
    this.setColor(this.color);
    if(this.vanish_timer >= 0.0f) {
        // 현재 레벨의 연소 시간으로 설정.
        float vanish_time =
            this.block_root.level_control.getVanishTime();
        Color color0 = // 현재 색과 흰색의 중간색.
            Color.Lerp(this.renderer.material.color, Color.white, 0.5f);
        Color color1 = // 현재 색과 검은색의 중간색.
            Color.Lerp(this.renderer.material.color, Color.black, 0.5f);
        ...
        }
}


public void toVanishing()
{
    // this.vanish_timer = Block.VANISH_TIME; ———— 주석 처리 또는 삭제
    // 현재 레벨의 연소 시간으로 설정.
    float vanish_time = this.block_root.level_control.getVanishTime();
    this.vanish_timer = vanish_time;
}
```

2. 텍스트 읽어오기

BlockControl.cs

```
public void rewindVanishTimer()
{
    // this.vanish_timer = Block.VANISH_TIME; ———— 주석 처리 또는 삭제
    // 현재 레벨의 연소 시간으로 설정.
    float vanish_time = this.block_root.level_control.getVanishTime();
    this.vanish_timer = vanish_time;
}


public void beginRespawn(int start_ipos_y)
{
    this.position_offset.y =
        (float)(start_ipos_y - this.i_pos.y) * Block.COLLISION_SIZE;
    this.next_step = Block.STEP.FALL;
    // int color_index = Random.Range(
    //     (int)Block.COLOR.FIRST, (int)Block.COLOR.LAST + 1);
    // this.setColor((Block.COLOR)color_index);
    // 현재 레벨의 출현 확률을 바탕으로 블록의 색을 결정한다.
    Block.COLOR color = this.block_root.selectBlockColor();
    this.setColor(color);
}
```

주석 처리 또는 삭제

3. 레벨디자인

정책 결정

난이도를 비슷하게 한다

패턴 별로 색상 수와 연소 시간을 바꾼다

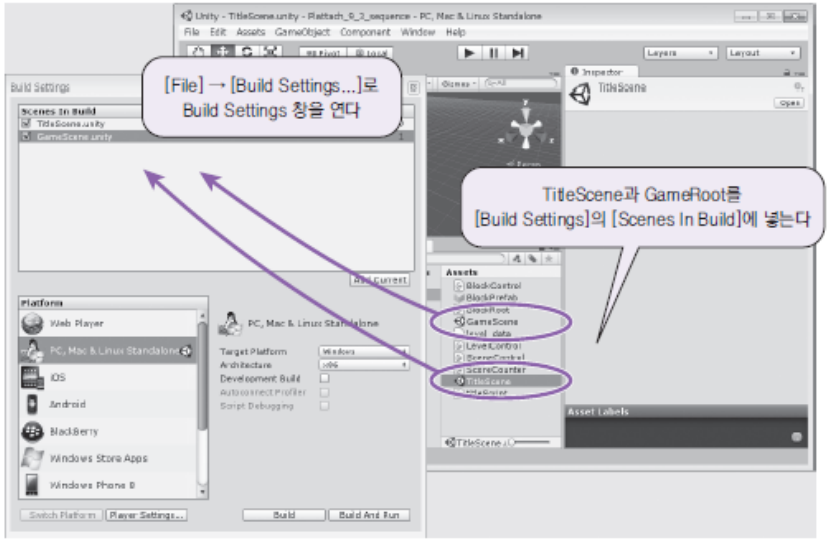
패턴 종류

- 패턴 A: 평균적인 색상 수·출현 확률·연소 시간
- 패턴 B: 색상 수는 조금 적지만 연소 시간이 조금 짧다.
- 패턴 C: 색상 수가 적지만 연소 시간이 짧다.
- 패턴 D: 색상 수는 조금 많지만 연소 시간이 조금 길다.
- 패턴 E: 색상 수는 많지만 연소 시간이 길다.
- 패턴 F: 특정 색이 나올 확률이 조금 높지만 연소 시간이 다소 짧다.
- 패턴 G: 특정 색이 나올 확률이 높지만 연소 시간이 짧다.

4. 게임 시퀀스 연결

TitleScript.cs

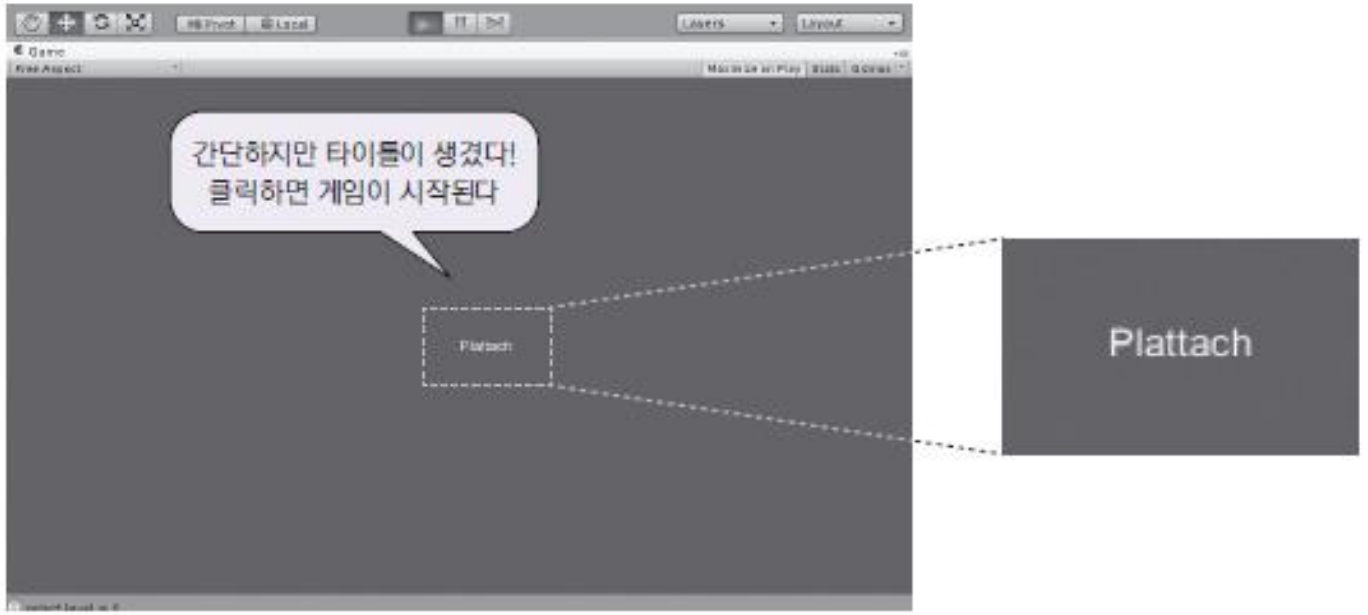
```
void Update() {
    if(Input.GetMouseButtonDown(0)) {
        Application.LoadLevel("GameScene");
    }
}

void OnGUI() {
    GUI.Label(new Rect(Screen.width / 2, Screen.height / 2, 128, 32),
        "Plattach");
}
```



SceneControl.cs

```
void Update() {
    this.step_timer += Time.deltaTime;
    switch(this.step) {
        case STEP.CLEAR:
            if(Input.GetMouseButtonDown(0)) {
                Application.LoadLevel("TitleScene");
            }
            break;
    }
    switch(this.step){
    case STEP.CLEAR:
        if(Input.GetMouseButtonDown(0)){
            Application.LoadLevel("TitleScene");
        }
        break;
    }
    ......
```

* 완성!



유니티
게임 제작 입문

이번에도 수고하셨습니다.

다음 페이지에서 보아요!

길벗