Handy Helper

Tag: 3-match, shoot, 2000s art, pixel
URL: https://lynnezzzy.itch.io/handyhelper
Play Time: 30mins

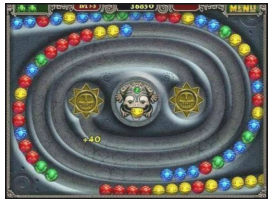
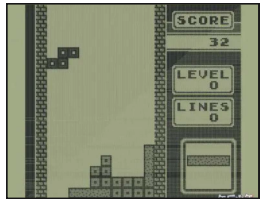
This is a 2D pixel-style match-3 game where you play as a little hand that can shoot and hover. Clear color blocks in a collapsing version of Windows Paint, find tools, and dig for memories!

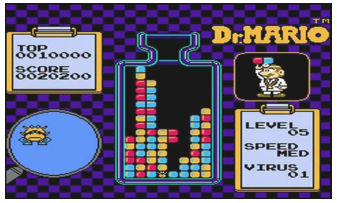
Time: 2025.7-2025.10 (4 months)
My Role: independent Developer

Skill & Tools:
Unity 2D
C#
Vibe Coding
Aseprite

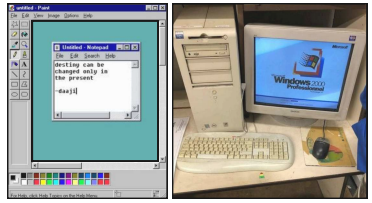
01 Background

Inspiration & Target Player



1. Satisfaction of match game!
This classic gameplay is attractive forever.



2. Nostalgia for 2000s vibe
Feel at home when see these old memories.

The classic match gameplay brought back memories and feelings from my childhood, playing in front of a huge machine. I want to make a game with an elegant mechanism based on classic match gameplay.
So here is Handy Helper, aiming to attract **casual indie game lovers (especially match game lovers)** and **2000s art fans**.

Feature

Creative Matching Gameplay	2000s Nostalgia Style	Satisfying Matching Fun

(Multidimensional Zuma + 2D Platform)

Story Background

The old system is about to vanish! The paint bucket, eraser, magnifying glass, and other tools are trapped deep under piles of colorful blocks. You're a brave little hand—this is your moment! Launch tiles, clear blocks, rescue your tool friends, and escape together! But before you go, there's still one special thing you have to find...

02 Gameplay

Core Rules

1. Goal

Shoot and match color blocks to eliminate them and reach the final area deep below.

2. Player Abilities

i. Shoot to jump

to **shoot**. Hold to enter **bullet time**.

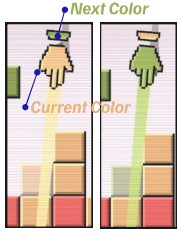
Shooting pushes you backward. Holding longer pushes you back more, up to a limit.

ii. Swap Color

You can see the next color. to **swap** between current & next color.

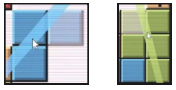
iii. Use tools

Press ␣ to use tools.



Swap Color

3. Matching Rules



Match-3 Group

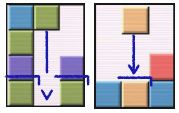
i. Match-3

If you shoot a color that matches at least 3 blocks of the same color, those blocks will be eliminated.

ii. Drop Rules

If a block has same-color blocks below or around, it drops.

Otherwise, it stays in the air.



"Around" *"Below"*

iii. Chain Match

If a dropped block lands and matches 3 blocks, they are eliminated.

This way, clearing chains of blocks will give you extra points.

4. Player Limits

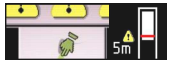
i. Gravity pulls you down.

You cannot touch blocks or the ground. If you do, you lose 1 HP. You have 3 HP total.

3 HP total

ii. Deadline

A line at the top moves down at a set speed. If it touches you, you lose the game, no matter how much HP you have.

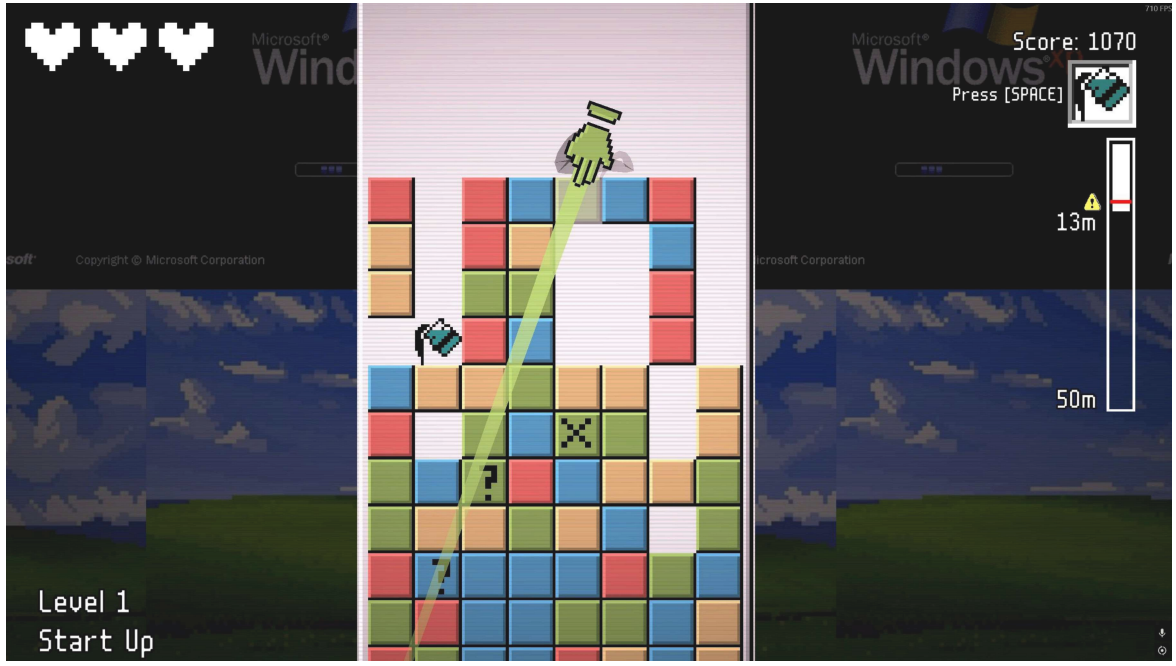


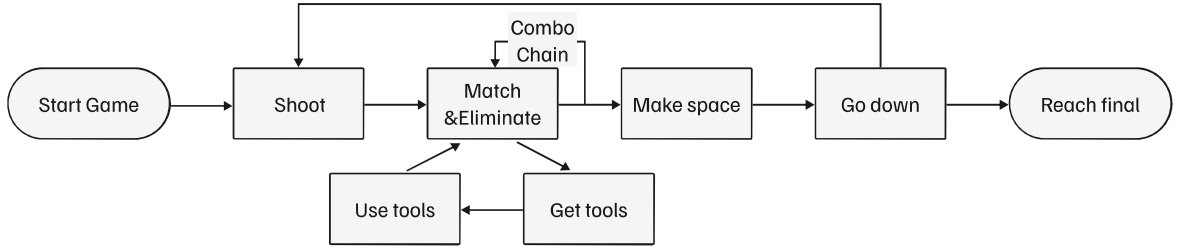
Deadline

Core Experiences

1. Eliminating Fun, especially in chain
2. A sense of floating and shuttling among the blocks.

Game View & Flow





03 Design Process

V1.0 - Jet

1. Controls

Shoot Hold to jet

2. Goal

Shoot & eliminate blocks to reach certain score without falling.

3. Rules - V1.0

i. Jet Energy: Using the jet costs energy, restored by clearing blocks.
ii. Pile: The blocks always stack up, as if pulled down by gravity.
iii. Growing Blocks: Blocks keep growing upward over time.

Problems

- Busy jetting and shooting without enjoying either.
- The score-based goal disrupted the fun of match.

V2.0 - Shoot to jump & Clear Screen

1. Controls

Shoot to jump Hold to enter bullet time

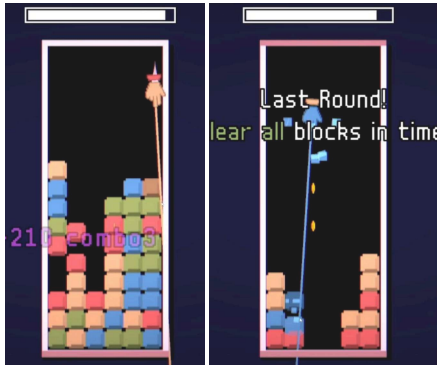
Swap Colors

2. Goal

Phase 1: Reach certain scores then enter Phase 2.
Phase 2: Blocks stop growing, players stay fixed in the air. Shooting no longer pushes them. They have 1 minute to clear all the blocks.

3. Rules - V2.0

i. Pile: The blocks always stack up, as if pulled down by gravity.
ii. Growing Blocks: Blocks keep growing upward over time.
iii. Special Blocks: They appear on random blocks in time interval. When players eliminate them, effects like boom or healing will occur.



Problems

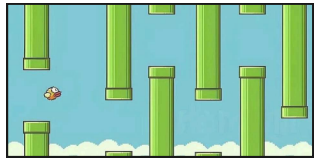
- Elimination feels shallow, lacking strategic fun.
- Floating feels stuck, no spatial progress
- Level feel either waiting or overwhelming

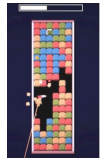
These three main problems do not occur on their own; they affect each other.
The piling blocks make space fixed to the above part, thus making the players feel trapped and floating feels pointless, which makes the whole level feel not exciting.

V3.0 - Shuttle & Dig

Exploration

"Why do I have to move? Why can't I just stay fixed in the air and clear the blocks?" When players asked this, it was clear the fun of floating was missing. I wanted to solve this first because it was related to my core experiences.
I recalled games that provide the fun of floating, such as "Flappy Bird". I realized the key is to **create threats above and below** and to **keep moving forward**.





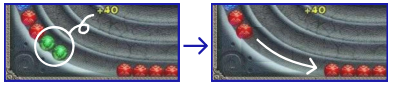
No wonder players find my dual gravity level exciting —it creates dangers above and below and gives a sense of expanding space.
But if I used dual gravity all the time, the gameplay did not feel elegant.

How to create threats above at the mechanism level?

What if there is some space among my blocks, which means my blocks won't pile up by gravity, and some of them can stay in the air?

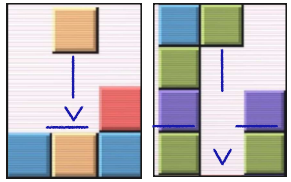
But if you do so, how could the chain of elimination occur?

What if the blocks can only drop when they can be connected to the same color block below? Wait, it's what Zuma has done!



Solution: Drop Rules & Dig Goals

Therefore, I removed the "piling blocks" rule: **Blocks will stay in the air unless there are blocks of the same color below (beneath, beneath left, and beneath right)**, in which case they will drop. When the dropped blocks match.



Beneath *Beneath left or right*

This way, there are threats above and below, making floating exciting and giving a sense of shuttle.
To help players feel like they are making progress, I changed the goal to digging deep and expanding the game view beyond a single fixed screen. All blocks now appear at the start instead of gradually.
There will also be a slowly descending deadline, creating time pressure to keep players engaged.
Players should shoot, clear blocks, and make space to go deep down to the end.

03 Design Process

V3.1 - Windows Tools

Art Style: 2000s Windows

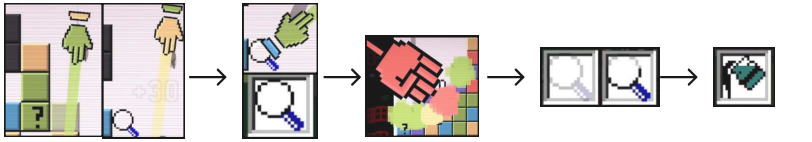
Art style was out of a flash of inspiration!

Tools?

One night before going to sleep, I imagined the old drawing tools and daydreamed about their effects. For example, in my game, there are spray paint tools to color blocks, eraser tools to erase color in specific areas, and so on. At first I thought it was just imagination, but after thinking carefully, it's actually very feasible!

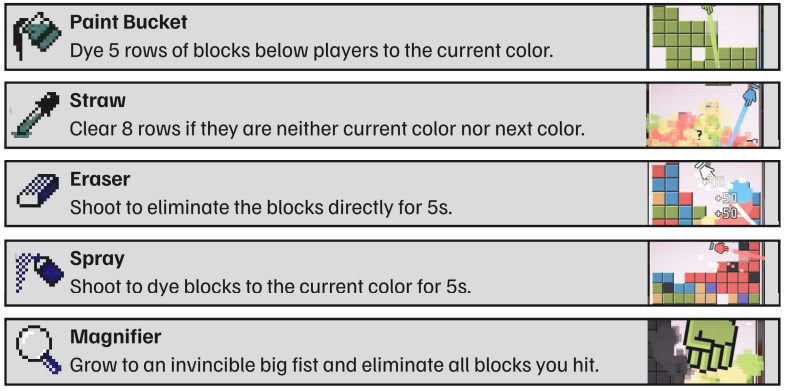
Tools System!

1. How to unlock tools



Eliminates the "Tool Block" & Reveal a random tool → *Pick up it to reset tool slot* → *Press Space to use* → *Get enough score to reactivate the tool* → *Pick up new tool to reset slot*

2. Tools Usage



Paint Bucket
Dye 5 rows of blocks below players to the current color.

Straw
Clear 8 rows if they are neither current color nor next color.

Eraser
Shoot to eliminate the blocks directly for 5s.

Spray
Shoot to dye blocks to the current color for 5s.

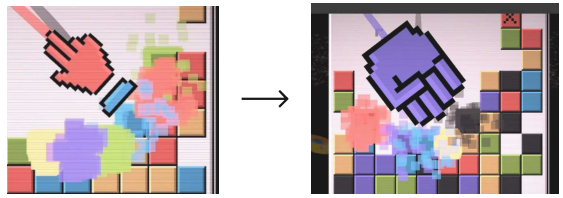
Magnifier
Grow to an invincible big fist and eliminate all blocks you hit.

V3.2 - Game Feel

1. To give fruitful feedback

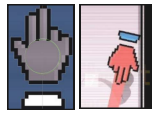
	Mechanism	Vision	Sound
Shoot	Recoil: Holding time ↑, the recoil ↑ 	Particle Spark Camera Freeze & Shake	Fast shoots ↑, sound pitch ↑
Eliminate	combos ↑, Blocks drop faster 	UI floating text: combos ↑, brighter color Particles	Combos ↑, sound pitch ↑
Tools: Magnifier	Increased Recoil Invincible & Damaging	Sprite changes to Fist, because it implies power	Zoom in; Zoom out

Initially, I didn't change the sprite and discovered that players wouldn't hit the blocks, wasting time not knowing what to do.



2. To provide some protection

i. Bouncy player



At first, players had no elasticity, so their speed dropped sharply when they hit walls or blocks, making adjustments difficult and felt clumsy. **Adding elasticity** solved this.

ii. Falling block protection: Circle collider and unharmfulness

At first, every block had a box collider, which made it easy for players to be pressed downward by falling blocks.

After changing the collider to a circle, players could be kicked and escape more easily because of the circle's multi-angle collision.

Also, I set falling blocks to be harmless so players would not feel shocked or discouraged when hit by unpredictable blocks.

iii. Face shooting protection

Initially, shooting a block in the face would hurt players and push them aside awkwardly. I set the collider to appear a little after the block becomes visible.

03 Design Process

Version 3.2 - Story & Gallery

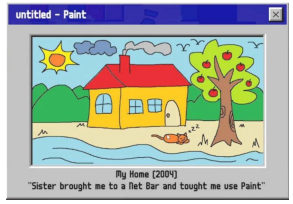
1. Main Story

> WINDOWS is going to crash down!
> I must save my friends in PAINT, and find that thing!!

The old Windows system was on the verge of crashing.
As the cursor, you had to break through layers of color block ruins, rescue tools, and retrieve that treasure.

At every bin you reached, the treasure had already been moved. The bins themselves could not hold on and vanished with the system.

Finally, you found that treasure. It was the memory of the first time your sister took you to an internet bar—you used the painting tool to draw an electric picture for the first time.



2. Tools Story

These tools are neighbors because they live in the same “building” of the Painting.
When the tragedy happens, they are buried beneath the blocks, waiting for players to save. When players collect the tools for the first time, they will see tools talking.

Samples of tools talking

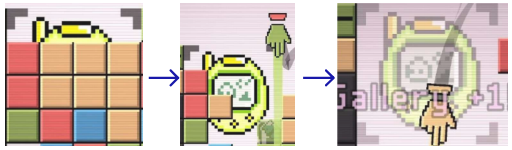
3. Gallery Collection

i. Gallery Entry

I add a gallery collection system in order to provide **extra surprises** when clearing blocks, enhancing **the sense of exploration**; and they are all retro objects, which can **increase immersion.**

ii. How to collect paintings

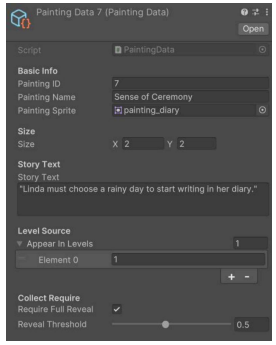
There are different paintings with 2*2 ~ 4*4 blocks size, hiding behind the blocks. Players can see and collect them after eliminating all the blocks blocking the paintings.



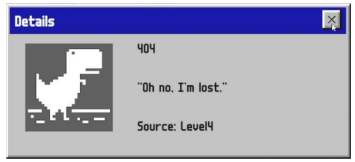
Process of collecting paintings

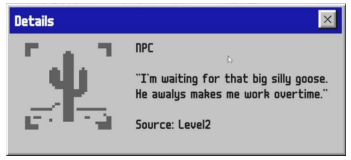
3.2. Painting Details

I added some short texts to each painting to make them more meaningful.

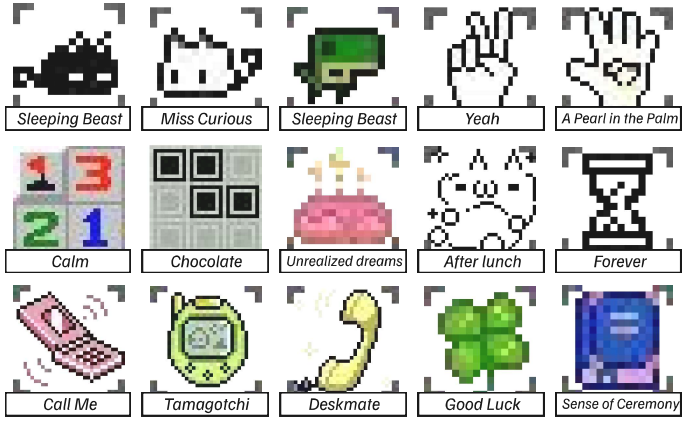


Painting Data Configuration





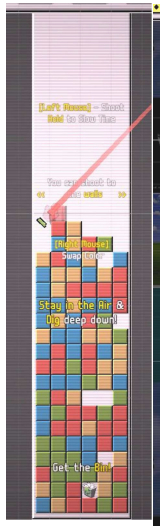
Painting Details



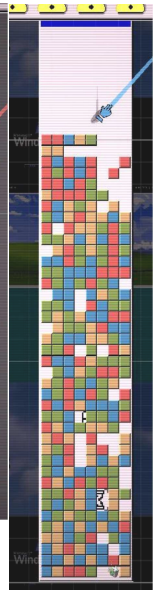
Samples of gallery paintings

03 Design Process

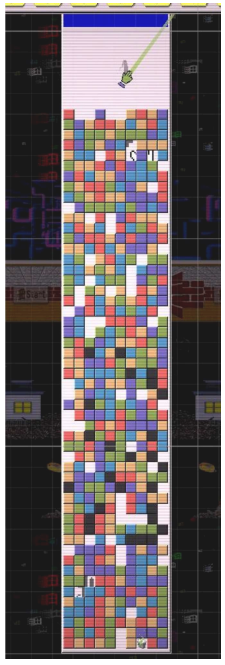
Version 3.3 - Diverse Level Design



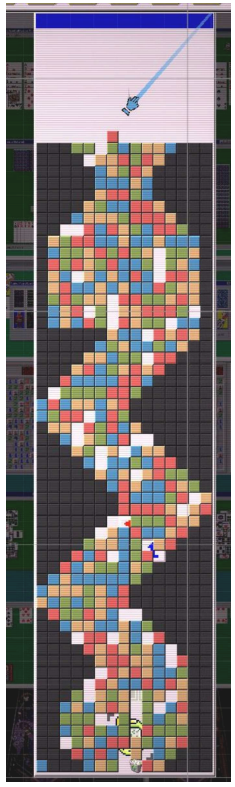
Level 0
"Hello!"
(Tutorial)
- *4-color*
- *6*34*
- *3mins*



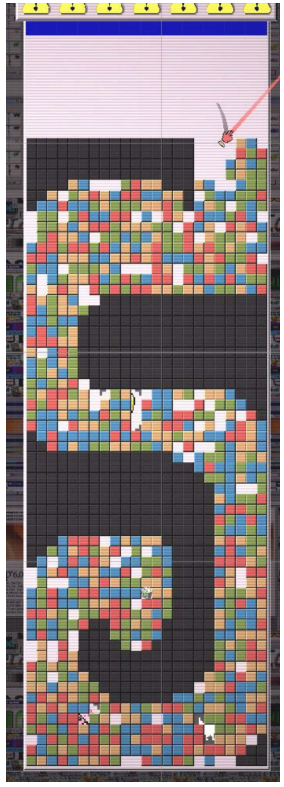
Level 1
"Start up"
(Introduce tools)
- *4-color*
- *8*50*
- *4mins*



Level 2
"Purple"
(5-color & Radom Rocks)
- *5-color*
- *10*60*
- *5mins*



Level 3
"Game box"
(Rock Cave)
- *4-color*
- *15*65*
- *6mins*



Level 4
"WordArt"
(Lateral & Upward flight)
- *4-color*
- *23*70*
- *8mins*

After testing, players feedback that all the levels feel similar.

1. Differences in mechanics

i. Map size

ii. Number of Colors

I tried 3, 4, 5, and 6 colors and found:

3 colors: too easy. Players can shoot at one point and still pass levels.

4 & 5 colors: fine. On larger maps, 5 colors feel too slow to progress and make it hard to create chains, which reduces the satisfaction of elimination.

6 colors: too complicated, blocking the fun of floating.

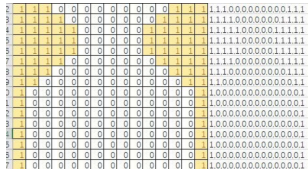
iii. Alien Terrain

Arrange stones to create stairs, caves, and upward passages.

Stones: Cannot be matched but can be eliminated, such as by a boom or eraser. This allows players to have a tricky way to pass the level.

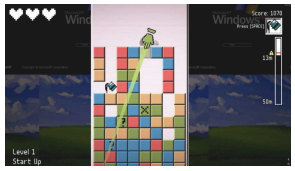
Stones can hurt players.

I implemented rock setting using an array, where 1 indicates the presence of a stone and 0 indicates no stone. Different columns are separated by [,], and different rows are separated by [;].



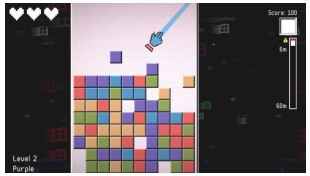
In Excel

2. Different Art elements



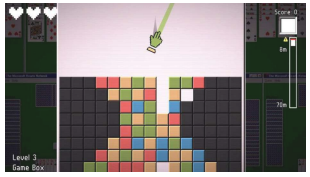
Level 1: Hello

BG: Windows Bg



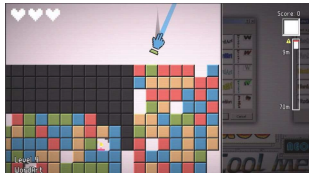
Level 2: Purple

BG: Screen Saver



Level 3: Game box

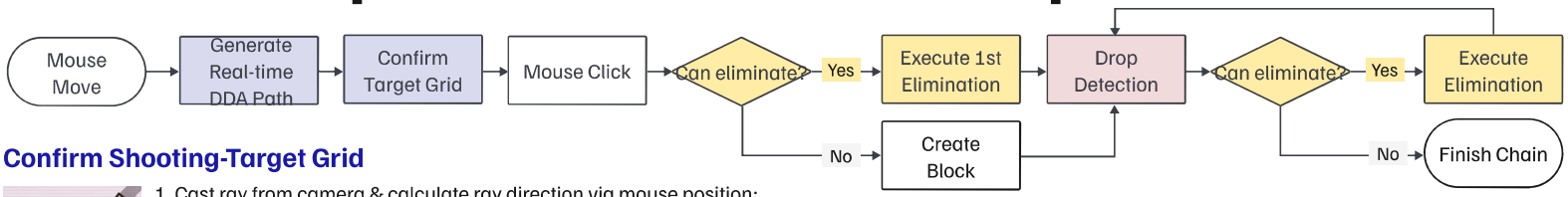
BG: Solitaire



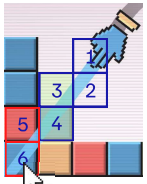
Level 4: WordArt

BG: WordArt

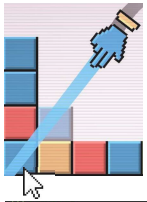
04 Code Sample: Shoot, Eliminate, Drop, Combo



Confirm Shooting-Target Grid



Mouse grid (Unplaceable)



1. Cast ray from camera & calculate ray direction via mouse position: (mouseWorld - playerPos).normalized
2. Generate DDA (Digital Differential Analyzer) path (Limited by maxSteps = 200 to avoid infinite traversal)
3. Traverse grids along the path starting from player position [1] [2] [3] ..... and perform:
   3.1. Obstacle detection (stop when encountering any blocks): [1] [2] [3] [4]
   3.2. Shooting-Target Grid Search (Three-Level Priority). Only consider placeable grids (empty grids with at least one neighboring block): [3] [4]
   Priority 1: Mouse exact grid position → Return immediately if found
   Priority 2: Last placeable grid encountered in path traversal: [4]
   Priority 3: Discard grid (first grid outside grid edges)
4. Visual Feedback:
   If valid target found: highlight it
   Else: mark as discard shot, ready to discard this bullet when click

```
/// <summary>
/// DDA Ray Traversal - Classic computer graphics algorithm for grid-based shooting
/// </summary>
2 references
List<Vector2Int> RaycastGrid(Vector3 start, Vector3 end, float cellSize, int maxStep)
{
    List<Vector2Int> path = new List<Vector2Int>();
    Vector2 dir = (Vector2)end - (Vector2)start).normalized;
    Vector2Int curGrid = gridManager.WorldToGrid(start);

    // Calculate step direction for each axis
    int stepX = dir.x > 0 ? 1 : (dir.x < 0 ? -1 : 0);
    int stepY = dir.y > 0 ? 1 : (dir.y < 0 ? -1 : 0);

    // DDA core variables: distance to next grid boundary
    float nextGridX = gridManager.GridToWorld(curGrid.x + (stepX > 0 ? 1 : 0), curGrid.y).x;
    float nextGridY = gridManager.GridToWorld(curGrid.x, curGrid.y + (stepY > 0 ? 1 : 0)).y;
    float tMaxX = (dir.x != 0) ? (nextGridX - start.x) / dir.x : float.MaxValue;
    float tMaxY = (dir.y != 0) ? (nextGridY - start.y) / dir.y : float.MaxValue;
    float tDeltaX = (dir.x != 0) ? cellSize / Mathf.Abs(dir.x) : float.MaxValue;
    float tDeltaY = (dir.y != 0) ? cellSize / Mathf.Abs(dir.y) : float.MaxValue;

    // DDA Main Loop - Greedy strategy: always choose shortest distance
    for (int i = 0; i < maxStep; i++)
    {
        path.Add(curGrid);
        if (!gridManager.IsValidGridPos(curGrid.x, curGrid.y)) break;

        if (tMaxX < tMaxY)
        {
            curGrid.x += stepX; tMaxX += tDeltaX;  // Step horizontally
        }
        else
        {
            curGrid.y += stepY; tMaxY += tDeltaY;  // Step vertically
        }
    }
    return path;
}
```

```
LaserTargetInfo GetLaserTarget()
{
    Vector3 playerPos = transform.position;
    Vector3 mouseWorld = cam.ScreenToWorldPoint(Input.mousePosition);
    mouseWorld.z = 0;
    Vector3 dir = (mouseWorld - playerPos).normalized;
    float cellSize = gridManager.cellSize;
    Vector3 rayEnd = playerPos + dir * laserLength;

    // Generate DDA path (limited to 200 steps)
    List<Vector2Int> path = RaycastGrid(playerPos, rayEnd, cellSize, 200);

    Vector2Int mouseGrid = gridManager.WorldToGrid(mouseWorld);
    Vector2Int? validTarget = null;
    Vector2Int? lastPlaceable = null;
    bool isDiscard = false;

    // Traverse path with obstacle detection and target search
    foreach (var grid in path)
    {
        // Obstacle detection: stop on existing cell
        if (gridManager.GetCell(grid.x, grid.y) != null) break;

        if (gridManager.IsPlaceableGrid(grid.x, grid.y))
        {
            lastPlaceable = grid;

            // PRIORITY 1: Mouse exact match
            if (grid == mouseGrid)
            {
                validTarget = grid;
                break;
            }
        }
    }
    // PRIORITY 2: Last placeable grid
    if (!validTarget.HasValue) validTarget = lastPlaceable;

    // PRIORITY 3: Edge discard
    if (!validTarget.HasValue)
    {
        validTarget = FindDiscardEdge(playerPos, dir, laserLength);
        isDiscard = true;
    }

    // Calculate world positions for rendering
```

Elimination: Multi-Round Chain System

Round 1: Initial Shooting Detection

1. When player shoots, use BFS(Breadth First Search) for same-color connectivity to detect if the shot block can form a match-3+ group.
   1.1. If match-3+ found, eliminate the group immediately.
   1.2. If no match-3 found, create new block.
2. Move to Next Phase: Drop Detection

Drop Detection

1. Scan each column to find suspended blocks
2. Calculate expected landing position for each block
3. Check if expected landing position has same-color neighbors.
   3.1. If has same-color neighbors, wait and drop together. Then mark newly landed blocks as "justLanded" for subsequent detection
   3.2. If no blocks should drop, then no “justLanded” mark.

Round 2+: Chain Elimination Loop

Do{

1. Collect Current “justLanded” Blocks from the previous Drop Detection
2. BFS Elimination Detection for each “justLanded” block
   Special Rule for Round 2+: Anti-Air-Elimination Protection
   Connected group must include at least one NON-justLanded block (stable block) to prevents "air elimination" where blocks in connected group falling together and eliminating themselves.
   2.1. If match-3+ found, remove qualifying connected groups
   2.2. If no match-3+ found, break loop
3. Clear justLanded Mark
4. Move to Next Phase: Drop Detection

} WHILE (there is justLanded blocks & match-3+ found)

05 Feedbacks

06 Take-home Msg

1. It’s OK to change some core features during development. Don't let sunk costs influence major decisions.
2. Good game feel needs fruitful feedback in every interaction.
3. Differences between levels can come from mechanics, as well as visual and narrative elements. These help create distinct atmospheres for each level.
4. The illusion of AI’s omniscience and power can push you out of your comfort zone. Use these qualities well. Clearly describe your demands.