討厭傳說

LEGEND OF HATE

Never gonna give you up

A game that is

counterintuitive,
infuriating,
user-unfriendly,
annoying,

yet addictive

2022 Tencent College Game Creative Production Competition Finalist Award (50/350)

2022 Yuan Qi Ludens Game Graduation Exhibition at the Communication University of China 3rd Prize (6/39)

Game Information

Keywords: Platform game, puzzle

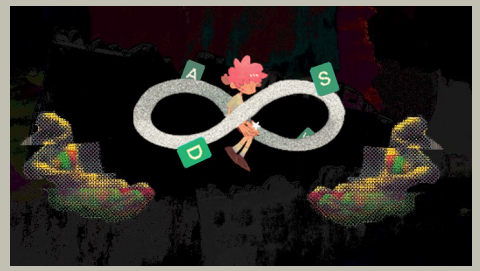
Development Engine: Unity3D 2020.3.26

Release Platforms: Windows

Development Team Size: 5

Download Link:

Game Duration: 30~45min



Overview

Years ago, a beautiful and harmonious world fell into chaos due to the arrival of a mysterious demon king — the Designer!

Hero, you once fought with your life against this foe, perishing together, your soul shattered into pieces.

Now, as darkness looms once again, we need your guidance more than ever...

Hero, imbue your spirit upon the eternal WASD, and rebuild the order...

Ah... though... there were some minor mishaps in the ritual, we still have a chance to fight back!

Yes, that's right. You are a long-dead warrior, and your unworthy descendants have dug you up from your grave to confront the demon king once more. They even botched the resurrection ritual...

Feeling your soul erratically bumping and careening within your flesh, you sigh. It's time to go and settle the score with that accursed being who keeps coming back from the dead.

The shoddy resurrection ceremony has left the 'hero's' soul restless, and the 'soul' that governs the body's 'up, down, left, right' movements has become extremely unstable, darting around within the body all the time. As a result, the hero needs to adapt to the 'soul' that 'darts' around in his body at regular intervals in order to control his body normally. However, the unstable 'soul' also means it is in a highly excited state, allowing the hero to actively release the 'soul', materializing mental power to attack enemies.

Roles and Responsibilities

Yichen Wu: Group leader, project manager, programmer

Tianle Cao: Lead designer

Leyan Wang: Lead programmer, designer

Zhuoxin Liu: Lead artist, Story writer

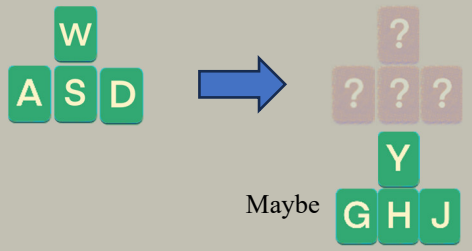
Yuyun Deng: Artist, Marketing Manager

How To Play

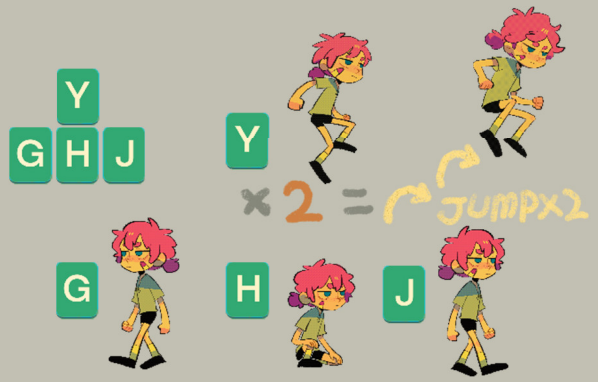
0.Players must overcome numerous challenges to reach the end of the level, defeat the Boss, and pass the level to unlock the next one.

Players have three health points. When HP are depleted, the player dies and is then resurrected at the most recent save point. (No items are lost upon death.)

1. The player's HUD includes a virtual keyboard display that shows the controls corresponding to movement direction keys. Players can execute corresponding directional actions by pressing the related keys on their physical keyboard (for instance, in the display: Y for jumping, G for moving left, J for moving right, H for crouching). When the time bar on the left side of the virtual keyboard depletes, the control buttons will maintain an inverted T shape and randomly refresh to new positions within the virtual keyboard. (Pressing the upward direction button while in the air enables a one-time double jump.)



2. When players drag a button from the virtual keyboard onto the game scene and release it with their mouse, the "hero" throws the button towards the mouse's position. The thrown button, as a materialized form of spiritual power, can be used to blast obstacles or attack enemies. However, after the button is thrown, the hero will be unable to perform the action corresponding to the lost button for a short period. After some time, the dispersed spiritual power will recongeal into a button, returning to the virtual keyboard, at which point players will regain the corresponding movement ability.experience.

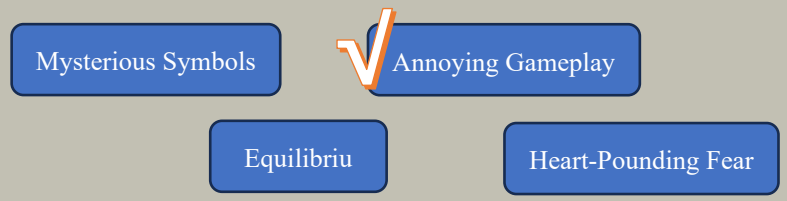


3. Throughout the game levels, numerous dashed-line frames are present, identical in size to the control buttons. These frames possess the unique feature of attracting and capturing the buttons. When a button collides with a frame, it's immediately drawn inside. Players also have the option to drag a button directly to a frame, prompting an instant attachment. While at times these frames might impede the player's capacity to attack adversaries, a new dynamic is introduced when a button is secured within a frame: players can then land on these button blocks, granting access to areas that were previously unreachable. Moreover, certain scenarios require players to strategically place specific buttons into these frames, initiating various in-game interactions

4. The lower right corner of the HUD displays the player's item inventory. Items can be acquired by opening treasure chests or defeating enemies. When players come near items dropped in the scene, they can drag them directly into their inventory using the mouse. Similarly, items obtained from treasure chests also require mouse dragging to be placed into the inventory. When players need to use an item, they must drag it from their inventory to a specific interaction point to engage. For example, as shown in the picture, players need to drag the potion to the hero to use it for health recovery. Items can also be thrown like control keys, inflicting damage on enemies.

Early Concepts

During the initiation of Game Project 2, we were presented with four themes to choose from for our project: "Mysterious Symbols," "Equilibrium," "Annoying Gameplay," and "Heart-Pounding Fear."



Among these, "Annoying Gameplay" immediately caught our attention due to its potent counter-intuitive appeal. Moreover, since Game Project 2 is an earlier course in the joint creation curriculum, experimental gameplay is highly encouraged. Consequently, my team members and I were instantly in agreement, selecting this theme as the direction for our creative endeavor.

Designing "Annoying Gameplay" was certainly not about creating a game that people would despise. Our original intention was to craft a title that was counterintuitive, infuriating, and had an anti-human operating experience, yet was addictive and difficult to put down.

We attempted to analyze some game mechanics that provide a pleasing experience (especially in platforming games), and found that many games possess qualities such as smooth controls, good character command, and intuitive use of scenery and props. So, what if we took the opposite approach, rebelling against these traits and exploring the hidden strategic and operational spaces within them? Could this provide players with a completely new experience?

Lovable Design	Annoying Design
Intuitive controls,	Sense of loss of control
Straightforward and user-friendly mechanics.	Excessive and unnecessary operational burden
Strong sense of control	Counterintuitive
Effective presentation of items	Unclear prompts
Immersive	non-immersive

Based on the points mentioned above, we have designed the gameplay as shown on the right.

Additional designs that contribute to the "annoying" experience can be found on the following pages.

Players must constantly be aware of key position changes, preparing in advance for key shifts, increasing the inherent operational burden.

Forced deprivation of operational capabilities, sometimes players have to voluntarily give up abilities.

Dotted line boxes that hinder long-range attacks, requiring players to pull enemies to the front of the box or circumvent the box themselves for effective attacks, forcing players to slow down the game pace to adapt.

The need to use cumbersome "drag-and-drop" actions to perform tasks that could be accomplished with simple clicks.

Items with positive effects can also damage the player's character upon collision, tempting players to use items with only negative effects, transferring distrust in the game world to the designers.

Counterintuitive elements, where past gaming experience does not help players succeed and can even mislead them into empiricist errors.

Spikes hidden in the background, filled with the designer's malicious humor

Having to constantly pay attention to the real-world keyboard.

Element Design

Interactive Scene Items

1. Treasure Chest

Can be opened by interacting with the dashed line box.

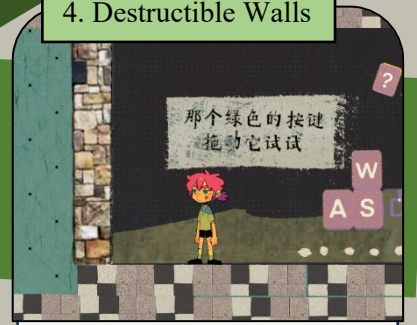
2.Spikes

As the name suggests, they're just spikes.

3. Pass-Through

Hold "Down" to pass through these platforms.

4. Destructible Walls



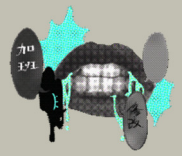
Walls that can be blasted open with keys.

5. Fans

The power of wind, interacting with the dotted line frame can change its direction, you know!

Monsters

1. Key Thief Monster



When it gets close to the player, it steals the key the player is currently using, causing the player to lose that key for a period of time.

2. Pistol Monster

Attacks the player with long-range bullet shots, which the player can easily avoid by crouching or jumping.

3. Eye Monster Brothers

Flying creatures that track and attack the player, with each type of eye monster dropping different prop rewards.

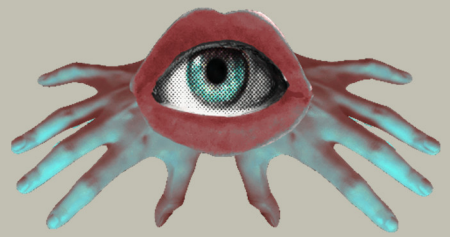
4. Dotted Line Frame Monster

A creature that shoots red dotted-line frames. When these frames hit the player, they cause significant damage and a knockback effect.

The frames don't disappear on their own and continue to hinder the player's movement and attacks. However, players can throw keys; when a key touches a frame, it's absorbed into the frame, 'neutralizing' the frame which then disappears. If the frame is 'neutralized' by a player's thrown key before it hits a wall or player after being shot, the Dotted Line Frame Monster receives significant recoil damage.

Boss

1. Mouth-Eyed Two-Handed Monster (Boss1)（The boss of the 2nd level）

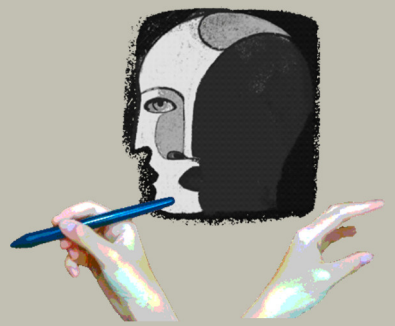


Can summon eye monsters to disrupt the player's long-range attacks.

Skill 1: Ground Slam, causing 1 point of damage to players on the ground at the moment of impact.

Skill 2: Leap, closes the distance to the player and causes area damage upon landing.

2. Designer (Boss2)　（The boss of the 3rd level）



Boss Room: Fan maze.

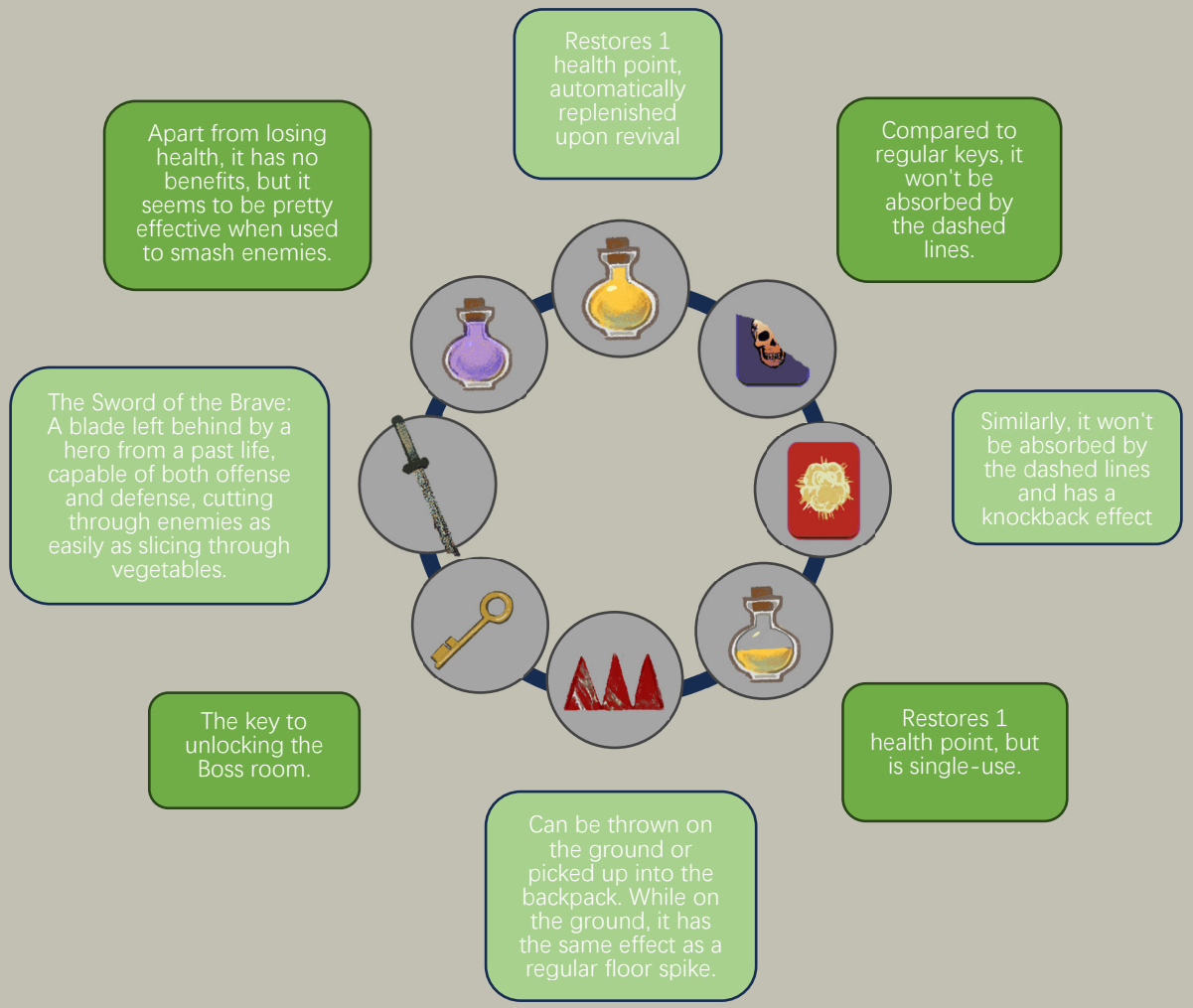
Left Hand: Shoots bullets.

Right Hand: Fires mouse cursors.

When the player's distance from the hands exceeds a set threshold, the originally single projectile turns into a barrage of projectiles

When a hand's health drops below 50%, it fires Red Dashed Line Boxes Pro, which, when "neutralized,"

Props



Level Design

The core gameplay revolves around key position changes, platforming, and puzzle-solving through key discarding.

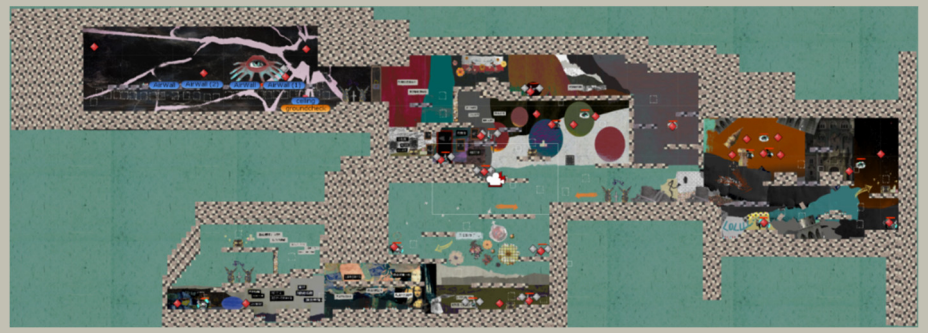
Therefore, in level design, we focus on **utilizing terrain, scene items, and different enemies to provide scenarios where players need to throw or use keys. By combining these scenarios, we challenge players to make decisions about the allocation of the four keys over a certain period of time**.

Level 1 (Tutorial Level)

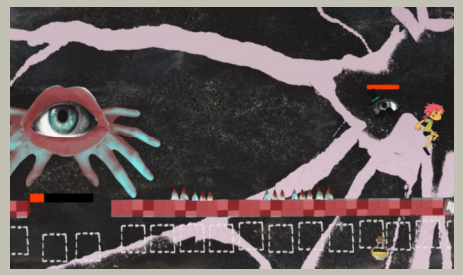
Mainly guides and teaches players the basic gameplay rules such as switching keys, throwing keys to solve puzzles, using keys to pave the way, and throwing keys to deal damage.

We hope that the annoying elements of the game permeate throughout the gameplay but are surmountable. We try not to implement imperiously irksome level components. As illustrated, players need to use 'up' to open the treasure chest, not 'right,' preventing players from being forcibly stuck waiting for their ability to move right to recover.

Level 2

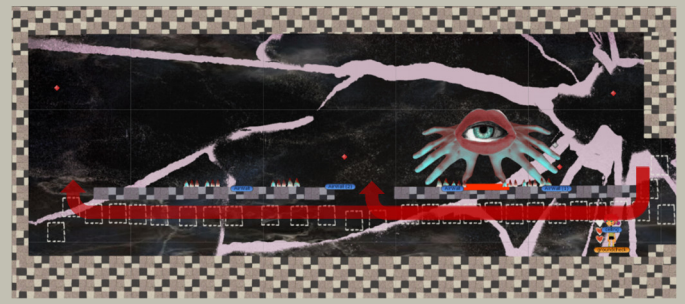


The second level introduces new monsters and props, providing both instruction and assessment of the player's fundamental operational skills. The overall difficulty of the level is not high, but the final Boss room demands a certain level of skill from the player. However, it still allows players to employ circuitous tactics to whittle down the Boss until they achieve victory.



The Boss1 has a ground-smashing ability; the moment it hits the ground, any player standing on the ground will take damage. Therefore, players need to retain the 'up' directional key to respond to this attack.

The Boss room is designed with two levels; the Boss can only move on the second level, while players can move freely between the two levels. Players need to act like whack-a-moles, interspersing through three gaps, using the 'down' button as a step to climb to the second level, and leveraging their speed advantage to kite and wear down the Boss from a distance.



The enemy's second skill's area damage will affect players on the first level, so players on the first level sometimes need to crawl to avoid area damage. Since the explosion damage from the player's keys is also considered area damage, when a key hits the floor of the second level, the resulting explosion damage will also affect the Boss on the second level within that range.

Therefore, we must find a way to avoid this situation, and we adapt by arranging a layer of dashed lines on the first floor. This way, we can solve this problem without increasing the programming workload. At the same time, to increase the pressure on players when they jump to the second level to attack the Boss from a distance after kiting, we have designed monster spawn points at locations A and B. Players not only have to use the physical keys to climb to the second level but also need to be constantly wary of and deal with the minions that may spawn at any time. After repeated testing, we adjusted the spawn frequency of the minions to put players under appropriate pressure.

Level 3

The third level introduces more monsters and items, featuring their more complex combinations, and the level scenarios continuously encourage players to gradually become proficient in throwing keys and using various items.

Iterative Process

At the end of each stage of the game's production, extensive testing was conducted with testers including members of the development team, friends, classmates, and family members. We listened widely to opinions and made improvements on many issues that were fed back, including but not limited to: insufficient guidance for new players, an overly steep difficulty curve, unreasonable movement routes, the presence of unorthodox strategies or emergence of gameplay methods we didn't intend. Simultaneously, without deviating from the main theme, we also accepted many new gameplay features suggested by players. Additionally, we collected many program bugs and promptly corrected these issues.

Here is some of our iterative content:

Time Bar & Spacebar & Key Auto-Recovery Time

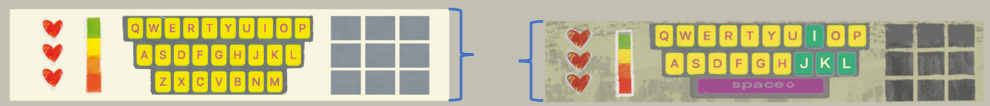
As previously introduced, in the first version of our game, we did not design a space bar mechanism to refresh the key position and cooldown time. This made the game less predictable. Once players mistakenly wasted a required direction key, they could only wait for its refresh. If the player lost the ability to evade enemies, they could easily die as a result.

In the first version, we adjusted the timer duration and the auto-recovery time for the thrown keys multiple times. For instance, as shown in the table, we adjusted from version v1.6's 10/3 to version v1.8's 15/2.5. Both theoretically and from playtest feedback, the game became more forgiving to players' mistakes. The adjusted version better met our expectations for the gaming experience, so it was retained.

Version	Duration of the Time Bar	Spacebar	Auto-Recovery Time
v1.6	10s	×	3s
v1.8	15s	×	2.5s
v2.2	35s	√	10s

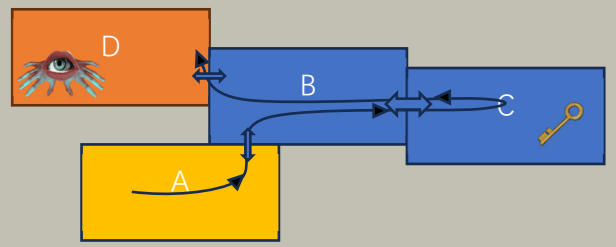
When participating in the Tencent University Game Creation Contest, because our gameplay was too experimental and somewhat overly frustrating, based on the advice of our mentor, we introduced a new mechanism: press the space bar to refresh the key position immediately. To emphasize the importance of this action, we extended the "automatic recovery of lost keys after a period" time. After the adjustment, players became more reliant on using the space bar to refresh the key position to gain controllable keys. This made combat rhythms more player-driven, allowing players to quickly refresh key positions. This meant that key throws, as a form of attack, became more efficient. As a result, we adjusted the monsters' health values again.

This iteration changed the combat rhythm, lowered the game's difficulty, and returned control of the game's pace to the player. However, this change also made the game less "annoying." It's challenging for us to determine if this adjustment fits our game. Therefore, we prefer to see this change as another version of our game rather than an improvement.



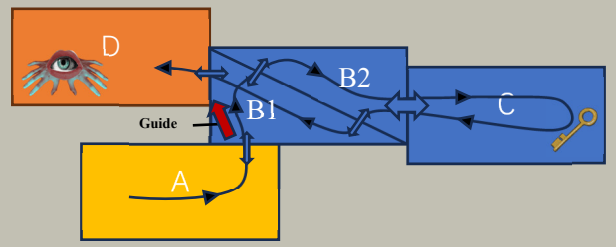
After incorporating the space bar control, in order to fit the space bar into the virtual keyboard and also to reduce the level of "annoyance," we reduced the randomizable positions of the inverted T-shape from the original two lines to one line. Moreover, we once again reduced the height of the keyboard, which means the UI occupies a smaller proportion of the entire screen, allowing players to see more of the game scene.

Movement Path in the Second Level



In side-scrolling games, progression often entails moving right. However, when designing the second level, we intentionally made the movement line counter-intuitive by positioning the Boss room in the top left corner of the map (D), while the key to unlock it was placed on the right side of the map (C).

Players, drawing on past experiences with side-scrolling games, would typically move right first, thus finding the key. By this time, the player would already be at the far right of the level, and they would need to retrace their steps to reach the Boss room. Many testers reported confusion about where to go next, and even if they did find their way back to the Boss room, they often already had the key in their possession. If players always end up with the key unwittingly by the time they reach the locked door, the purpose and guiding role of the key are significantly diminished.

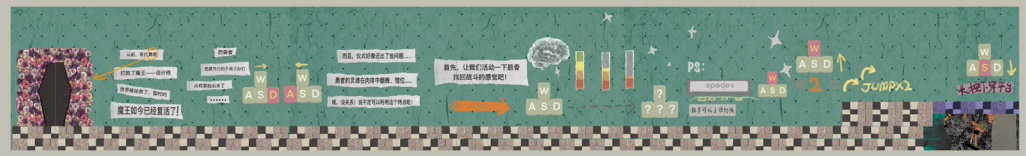


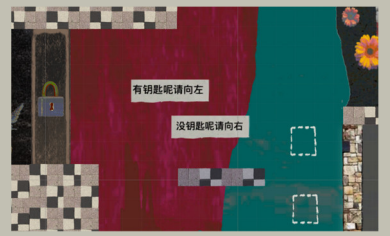
Therefore, we adjusted the movement line, considering players tend to explore unvisited areas. We divided Area B into two sections, B1 and B2. When players reach B1, whether they move left or right, they can reach the other side through an unexplored path after arriving at either C or D.

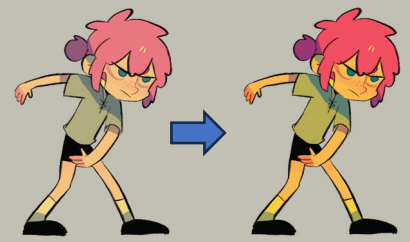
We also added some monsters and treasure chests along the shorter route from B1 to C, guiding players to explore left first. After reaching the Boss room, players are given hints to move right in search of the key, using the task of finding the key to motivate players to explore more areas. This improvement avoids forcing players to repeatedly scour the same area and enhances the key's significance as a task driver.

Collage Art & Fragmented Narration & Beginner's Tutorial

Players, drawing on past experiences with side-scrolling games, would typically move right first, thus finding the key. By this time, the player would already be at the far right of the level, and they would need to retrace their steps to reach the Boss room. Many testers reported confusion about where to go next, and even if they did find their way back to the Boss room, they often already had the key in their possession. If players always end up with the key unwittingly by the time they reach the locked door, the purpose and guiding role of the key are significantly diminished.





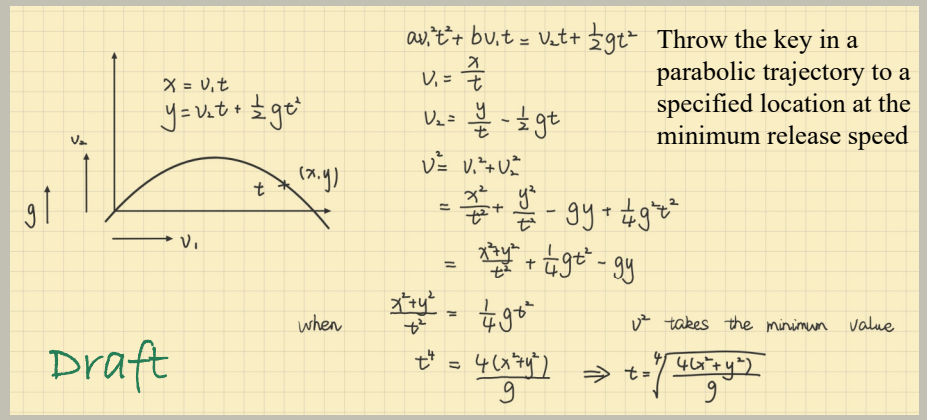


Due to the collage art style, the background is rich in content, making the player character less prominent. Our artists increased the saturation of the character, making the protagonist brighter and more noticeable, so players can more easily observe the character's responses and movements.

Programming Development (the part I handle) (Selected)

Implementation of Key-based Core Gameplay interactions

a. Key Throw



Throw the key in a parabolic trajectory to a specified location at the minimum release speed

```
public void OnToss(Vector3 target, int letterNO)
{
    AudioManager.PlayAudio("投掷出手_1");
    //Find the right key
    key = alphabet[letterNO];

    //Calculating curves based on formulas
    float g = G * GScale;
    Vector2 Self = transform.position + Offset + 0.2f * new Vector3(target.x - transform.position.x, 0, 0);
    float a = Mathf.Pow((target.x - Self.x), 2) + Mathf.Pow((target.y - Self.y), 2);
    float b = Mathf.Pow(g, 2) / 4;
    //float c = g * (target.y - Self.y);
    float t = Mathf.Pow((a / b), 0.25f);
    float Vx = (target.x - Self.x) / t;
    float Vy = (target.y - Self.y) / t + g / 2 * t;
    GameObject Key = Instantiate(key, Self, Quaternion.identity);
    putCtrl(Key, letterNO);
    Key.GetComponent<Rigidbody2D>().velocity = new Vector2(Vx, Vy);
    Key.GetComponent<Rigidbody2D>().angularVelocity = Vx * Rate;
    Key.GetComponent<Explode>().explodeType = 1;
}
```

b. Key Position Switching （selected）

```
public void changeKeys(int n, int[,] array)
{

    player.GetComponent<FinalMovement>().isDragging = false;
    currentKeyGroup = n;

    //If the thrown one is still valid after the switch, it's fine. Otherwise, it needs to be reset before being destroyed
    for (int i = 0; i < 4; i++)
    {
        if (currentInUse[i])
        {

            bool flag = false;
            int j;

            for (j = 0; j < 4; j++)
            {
                if (currentInUse[i] == triggers[array[n, j] - 1]) //still valid
                {
                    caches.Add(j);
                    caches2.Add(i);
                    flag = true;
                }
            }
            if (!flag)//invalid
            {
                //go explosion
                GameObject temp = player.GetComponent<Toss>().takOutCtrl(currentInUse[i].GetComponent<UnderKey>().childNo);
                if (temp)//hasn't exploded, explod
                {
                    temp.GetComponent<Explode>().goExplode();
                }
                //
                currentInUse[i].GetComponent<CanvasGroup>().alpha = 0;
                currentInUse[i].SetActive(false);

            }
        }
    }
    int k = 0;
    for (int i = 0; i < 4; i++)
    {
        if (!caches2.Contains(i))
        {
            bool success = false;
            while (!success)
            {
                if (!caches.Contains(k))
                {
                    currentInUse[i] = triggers[array[n, k] - 1];
                    currentInUse[i].SetActive(true);
                    k++;
                    success = true;
                }
                else
                {
                    k++;
                    currentInUse[i].GetComponent<CanvasGroup>().alpha = 0f;
                }
            }
            currentInUse[i].transform.GetChild(0).gameObject.SetActive(true);
            player.GetComponent<FinalMovement>().draggedKey[currentInUse[i].GetComponent<UnderKey>().childNo] = 0;
        }
        currentInUse[i].GetComponent<UnderKey>().OutCD();
    }
    caches.Clear();
    caches2.Clear();
}
```

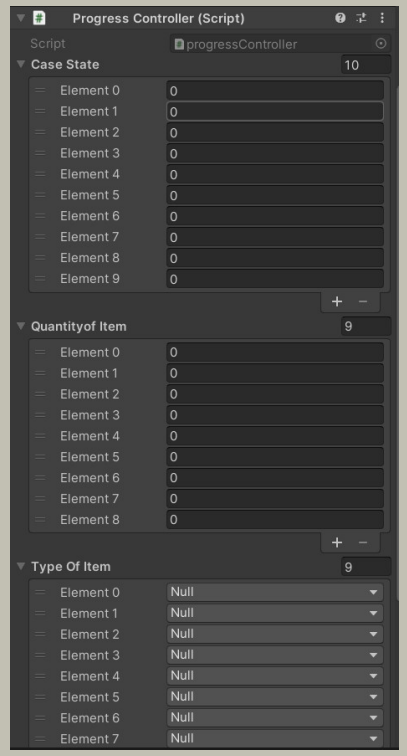
If the thrown one is still valid after the switch, it's fine. Otherwise, it needs to be reset before being destroyed

If the refreshed key position overlaps with the previous one, inherit the key status and cooldown.

Blink in advance to indicate the next key position to be switched, allowing players to prepare ahead of time

Inventory and Save System

The Save System primarily stores information such as save point data, the status of each treasure chest (whether it's been opened), the status of special items acquired, and details about the items within the inventory, among other things. The storage method varies for different items; some items increase by one upon acquisition and are lost after use, while others, once obtained, increase the maximum limit for that item. When the player respawns after death, all such items are refilled to this maximum limit.



The Inventory System, on the other hand, displays the items held, based on the 'items within the inventory' data from the Save System, showing different items in their respective manner in the item bar. The inventory system also allows for the dragging in of items, the use of items by dragging them out, the merging of similar items, and the swapping of item positions, among other functionalities.

The Blower script exposes parameters for debugging various blower attributes (such as wind direction switching sequence, wind force, and attenuation curves) in the editor, ensuring adjustability and fine-tuning.

The Audio Controller script manages all aspects of audio, including volume control, playing, pausing, and stopping. It categorizes audio into two mixers: BGM and SFX, to facilitate separate management.

My Contributions to Legend of Hate

Design Part

On the basis of the 'WASD keys randomly refreshing positions' gameplay proposed by my lead designer colleague, I contemplated and designed gameplay mechanics where throwing a key results in the loss of corresponding abilities, utilizing keys to attack enemies, platform jumping, and puzzle-solving for level progression.

I also drafted some initial settings for this mechanic. Moreover, I conceptualized the primary abilities for the Key-Stealing Creature and the Dashed-Line Creature, deepening and expanding the key-centric gameplay.

As the executor for the programming of items, I collaborated with the lead strategist to discuss and finalize the functionalities of certain items, combining gameplay mechanics and programming feasibility.

In terms of level design, the lead designer was responsible for the initial drafts, and I joined the testers in making revisions and refinements based on their feedback.

Development Part

In the initial phase of programming, I was involved in functional modularization, assigning coding responsibilities, and standardizing programming interfaces.

I also contributed to the construction of some fundamental frameworks, including the save system, inventory system, audio system, and numerical system.

My responsibilities extended to the implementation of key-related core mechanics, such as virtual keyboard display, timed key position swapping, pre-swap blinking alerts, key drag-and-throw/place mechanics, damage calculation for key explosions, and interactions between keys and dashed-line boxes, among others.

Additionally, I handled the realization of interactive scene items, encompassing destructible barriers, interactive dashed-line boxes, ground spikes, treasure chests, save points, blowers, and unlockable doors.

Pertaining to the item segment, I managed the physical properties of props, operational logic, usage functions, UI for function descriptions, and the seamless transition display between the inventory UI and the scene entities.

Insights and Reflections

Through this project, I became acquainted with the process of team collaboration in creating 2D games and learned about specific programming implementations in platformer games, such as coyote time, input buffering, and dynamic gravity adjustments. I also gained an initial understanding and hands-on experience with the programming structure of common game systems, like character statistics.

While our game, to a significant extent, represents a rebellion against the existing mechanics and experiences of acclaimed platformer games, this defiance may not constitute true innovation. It even introduces certain challenges that the original designs were meant to address. However, overall, our design provides an experience that diverges from many other games. Players must execute more precise controls for tasks that would normally require just a simple hotkey, allocate their actionable abilities sensibly, and adopt indirect tactics to perform what are often fluid actions in other games. These aspects bestow 'Legend of Hate' with its unique traits.

More importantly, by analyzing the beloved mechanics of game design, I've enhanced my capacity to deconstruct games. This enables me to draw from a wider range of exemplary works and to approach each step of creation with an increasingly professional game designer's perspective, thereby progressively producing work that meets a higher standard of satisfaction.