Human Interface Computer Game Design

Stress Room

Final Report

Subject	Human Interface Computer Game Design
Professor	Professor Chang Ha Lee
Team Name	**Stress Room**
Member (Student Number)	Ga Young Kim(20150035) Mi Ji Park(20150207) Soo Hyun Park(20152583) Kyeo Rye Lee(20120431)

INDEX

1. Title

Stress Reliever Game

2. Development Background

Stress is a widespread issue. Most people get stressed by many things. So we suggest a game which can relieve their stresses by destroying some things in virtual space.

3. Overall Design

- This game is modeled on the real situation in the room.
- This game contains mainly a character, the things that the character can throw away or break such as computers, desks, chairs, glasses.
- The character can move and swing his arm with the weapons.
- There are several stages and the higher stage is more difficult. Each stages have the limit time.
- As the character finishes the stage, he can get some weapons which improve his strength. One can break or throw away something easily when he or she has strong weapons.
- The goal of the game is to finish the last stage and escape the room.
- If the limit time is over, then the game will be over.

3.1 Storyline

You, under a lot of stress, are in a strange room when you open your eyes.
To get out of the room, you have to load up everything in the room and destroy all the things blocking the door.
The more you pass through the room, the more you get your weapon and will also have more power. And obstacles do not break easily.
If you pass through rooms of all levels, you can go outside.

3.2 Character

- **Player**
  The user who plays the game himself is a character.
- **Weapon**
  What players use to break obstacles
  Passing through levels gives you more powerful weapons.

- **Obstacle**
  The player has to break through the room.
  the higher the level, the harder it will get.

3.3 Interface

- You are able to move **a single character** with first-person-view.
- There is a **time bar** for each stages and you have to escape each rooms within a given time. The time bar is provided on the upper side of the screen.
- There are two kinds of **objects** - anti objects and special objects. The first things are the obstacles you have to break, and the other things are the ones you will interact with. Something like a door to open or the items to get and so on.
- There are interactive **keyboard buttons** listed below.
  (1) Move : '**Up(W)**', '**Down(S)**', '**Left(A)**', '**Right(D)**'
  (2) Attack : '**Space**' to attack anti objects. Your action is depend on your current weapon. *(For example, when you have a club, this key will make your character swing a club. With a shotgun, you will shoot a single shot forward.)*
  (3) Interface Example : See **Figure 1** below.

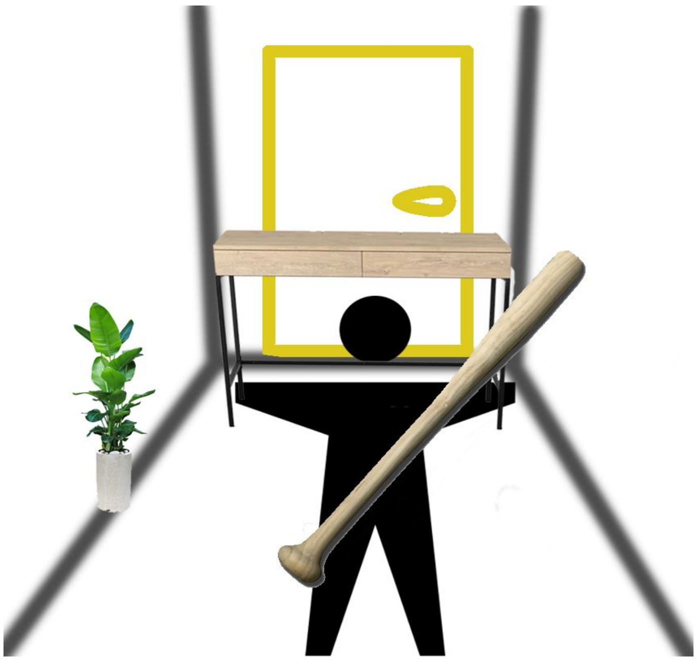


Figure 1. Interface Example

4. Role of each member

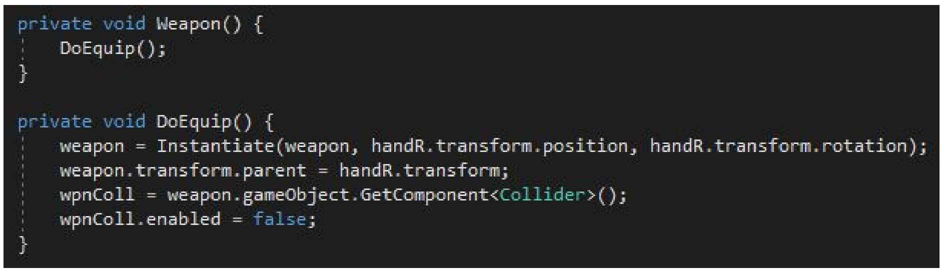
- ➢ Ga Young Kim(20150035)
  - ○ Implementing an Obstacle Collision
- ➢ Mi Ji Park(20150207)
  - ○ Implementing Scene & Obstacle Collision
- ➢ Soo Hyun Park(20152583)
  - ○ Implementing an Obstacle Collision & Camera View Point
- ➢ Kyeo Rye Lee(20120431)
  - ○ Implementing Character Movement & Intro

5. Script Description

➢ Main Character Control Script

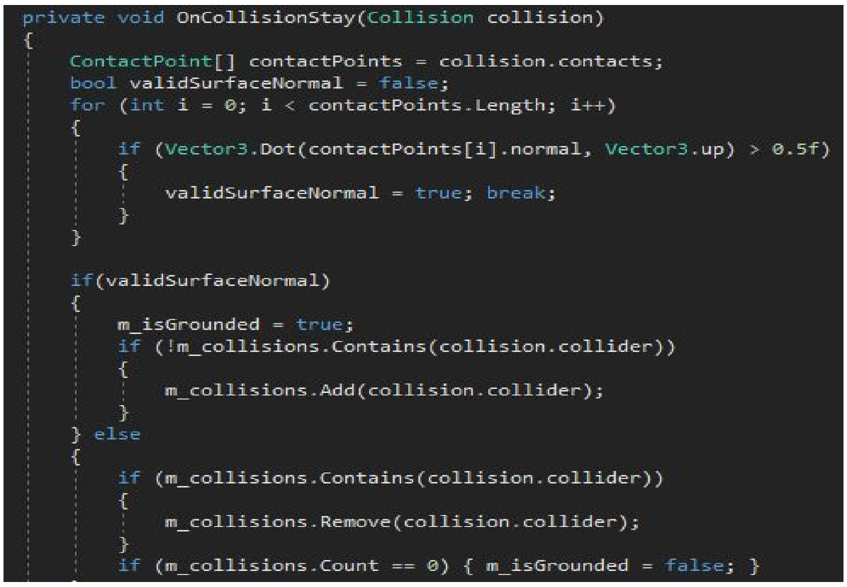
- ○ Script File Name : Simple Character Control

(1) **Weapon()** : This script is about character's weapon and executed by DoEquip() method. Character object is consist of human body hierarchy like upper body, right shoulder, right arm and so on. And weapon object is attached to character's right hand and becomes character's child component.

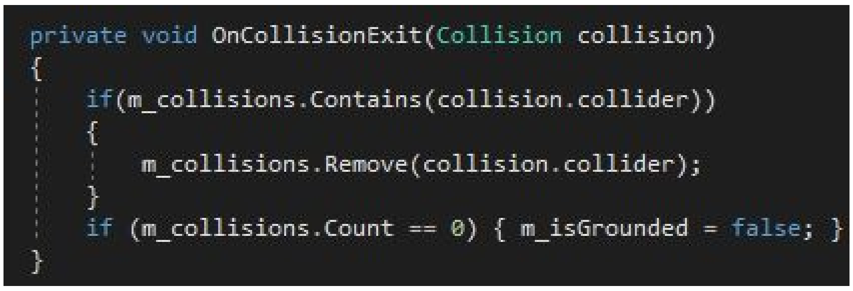


```
private void Weapon() {
    DoEquip();
}

private void DoEquip() {
    weapon = Instantiate(weapon, handR.transform.position, handR.transform.rotation);
    weapon.transform.parent = handR.transform;
    wpnColl = weapon.gameObject.GetComponent<Collider>();
    wpnColl.enabled = false;
}
```

(2) **OnCollisionStay() & OnCollisionExit()** : This script is for character's collision. OnCollisionStay() and OnCollisionExit() methods detect the information whether the other objects touched the character or not.

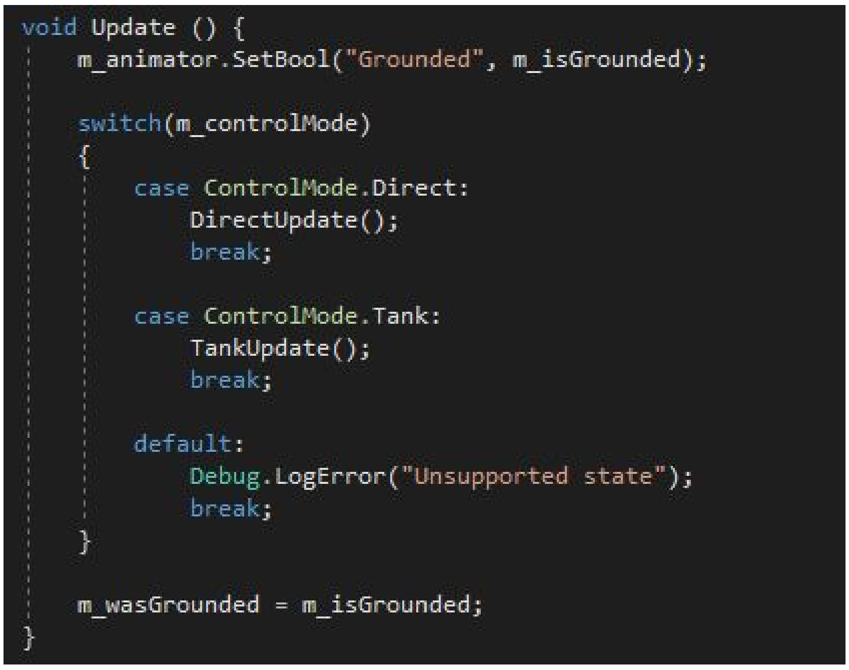


```
private void OnCollisionStay(Collision collision)
{
    ContactPoint[] contactPoints = collision.contacts;
    bool validSurfaceNormal = false;
    for (int i = 0; i < contactPoints.Length; i++)
    {
        if (Vector3.Dot(contactPoints[i].normal, Vector3.up) > 0.5f)
        {
            validSurfaceNormal = true; break;
        }
    }

    if(validSurfaceNormal)
    {
        m_isGrounded = true;
        if (!m_collisions.Contains(collision.collider))
        {
            m_collisions.Add(collision.collider);
        }
    } else
    {
        if (m_collisions.Contains(collision.collider))
        {
            m_collisions.Remove(collision.collider);
        }
        if (m_collisions.Count == 0) { m_isGrounded = false; }
```

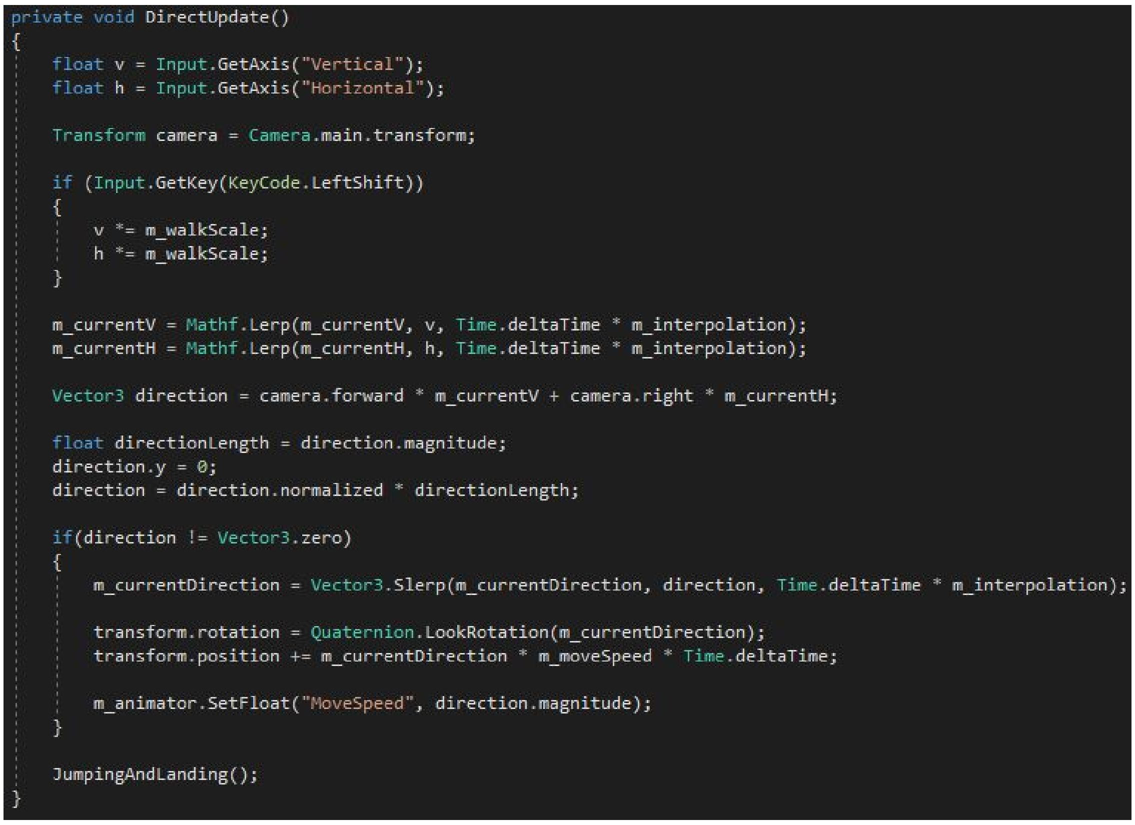


```
private void OnCollisionExit(Collision collision)
{
    if(m_collisions.Contains(collision.collider))
    {
        m_collisions.Remove(collision.collider);
    }
    if (m_collisions.Count == 0) { m_isGrounded = false; }
}
```

(3) **Update()** : This script is about updating character’s movement per frame. There are two modes for updating but we use only ‘Direct’ mode, so DirectUpdate() method will be used only. DirectUpdate() method changes character’s direction and rotation according to player’s key inputs. We set ‘s’ and ‘w’ to Vertical input and ‘a’ and ‘d’ to Horizontal input. Vertical inputs rotate character anti-clockwise or clockwise each, and Horizontal inputs move character forward and backward. And this script also updates the state whether the character is grounded or not.



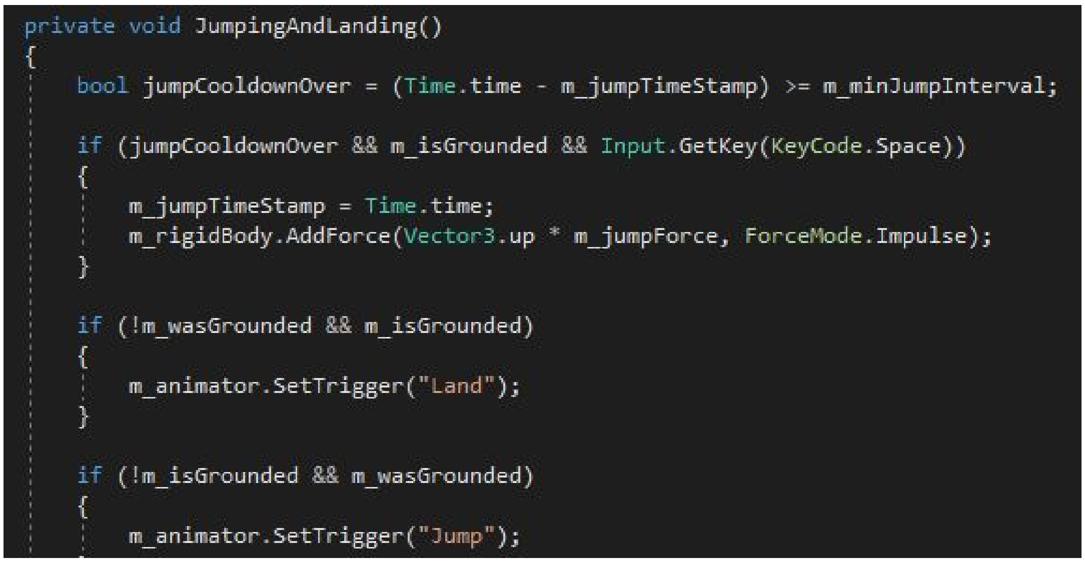
```
void Update () {
    m_animator.SetBool("Grounded", m_isGrounded);

    switch(m_controlMode)
    {
        case ControlMode.Direct:
            DirectUpdate();
            break;

        case ControlMode.Tank:
            TankUpdate();
            break;

        default:
            Debug.LogError("Unsupported state");
            break;
    }

    m_wasGrounded = m_isGrounded;
}
```



```
private void DirectUpdate()
{
    float v = Input.GetAxis("Vertical");
    float h = Input.GetAxis("Horizontal");

    Transform camera = Camera.main.transform;

    if (Input.GetKey(KeyCode.LeftShift))
    {
        v *= m_walkScale;
        h *= m_walkScale;
    }

    m_currentV = Mathf.Lerp(m_currentV, v, Time.deltaTime * m_interpolation);
    m_currentH = Mathf.Lerp(m_currentH, h, Time.deltaTime * m_interpolation);

    Vector3 direction = camera.forward * m_currentV + camera.right * m_currentH;

    float directionLength = direction.magnitude;
    direction.y = 0;
    direction = direction.normalized * directionLength;

    if(direction != Vector3.zero)
    {
        m_currentDirection = Vector3.Slerp(m_currentDirection, direction, Time.deltaTime * m_interpolation);

        transform.rotation = Quaternion.LookRotation(m_currentDirection);
        transform.position += m_currentDirection * m_moveSpeed * Time.deltaTime;

        m_animator.SetFloat("MoveSpeed", direction.magnitude);
    }

    JumpingAndLanding();
}
```

(4) JumpingAndLanding()

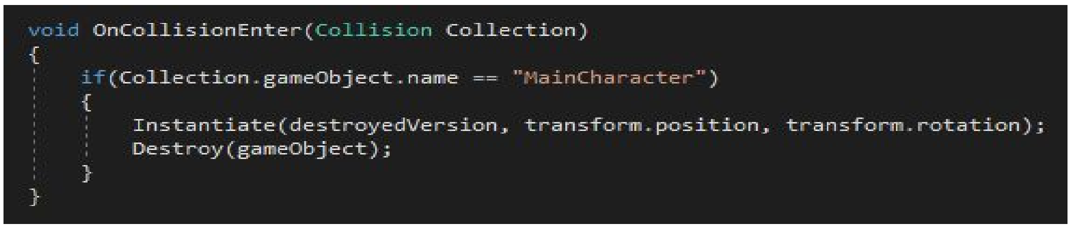
Change the location, status and behavior of game characters before rendering each frame. When the character is on the ground, meets the 'JumpCooldownOver' condition and the user presses the space key, the character jumps. When a character jumps, the rigidBody moves by adding a force of m_jumpForce in the y-axis direction(Vector3.up). When the character is grounded and was not grounded, then landing animation. When the character is not grounded and was grounded, then jump animation.



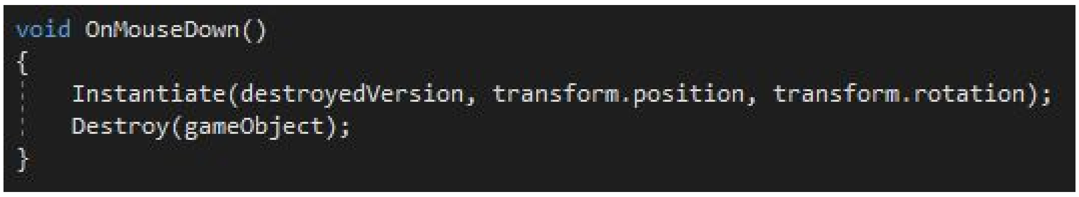
```
private void JumpingAndLanding()
{
    bool jumpCooldownOver = (Time.time - m_jumpTimeStamp) >= m_minJumpInterval;

    if (jumpCooldownOver && m_isGrounded && Input.GetKey(KeyCode.Space))
    {
        m_jumpTimeStamp = Time.time;
        m_rigidBody.AddForce(Vector3.up * m_jumpForce, ForceMode.Impulse);
    }

    if (!m_wasGrounded && m_isGrounded)
    {
        m_animator.SetTrigger("Land");
    }

    if (!m_isGrounded && m_wasGrounded)
    {
        m_animator.SetTrigger("Jump");
```

➢ Obstacle Collision Script

- Script File Name : <u>Destructible</u>
- What a Script does : It is a script that is applied to obstacles. When a weapon hits an obstacle, the obstacle is converted to an object that you have previously specified.



```
void OnCollisionEnter(Collision Collection)
{
    if(Collection.gameObject.name == "MainCharacter")
    {
        Instantiate(destroyedVersion, transform.position, transform.rotation);
        Destroy(gameObject);
    }
}
```

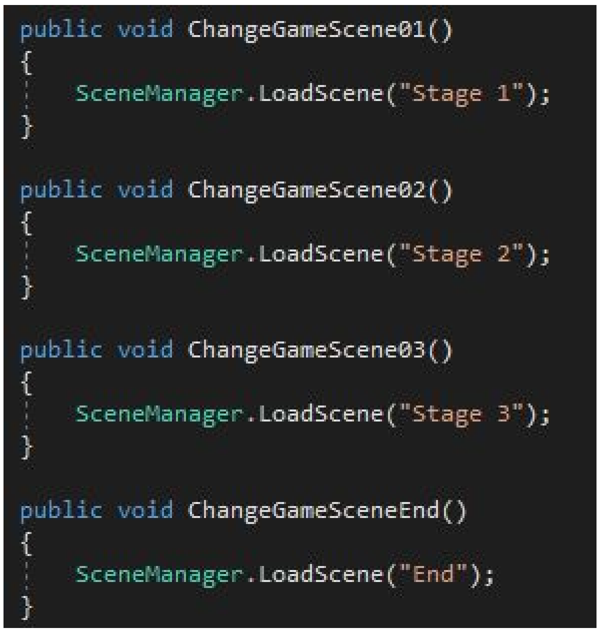


```
void OnMouseDown()
{
    Instantiate(destroyedVersion, transform.position, transform.rotation);
    Destroy(gameObject);
}
```

- If the collider of the character with the weapon overlaps with the collider of the obstacle, a collision occurs.
- Original obstacle and converted object are both applied gravity.

➢ Scene Change Script

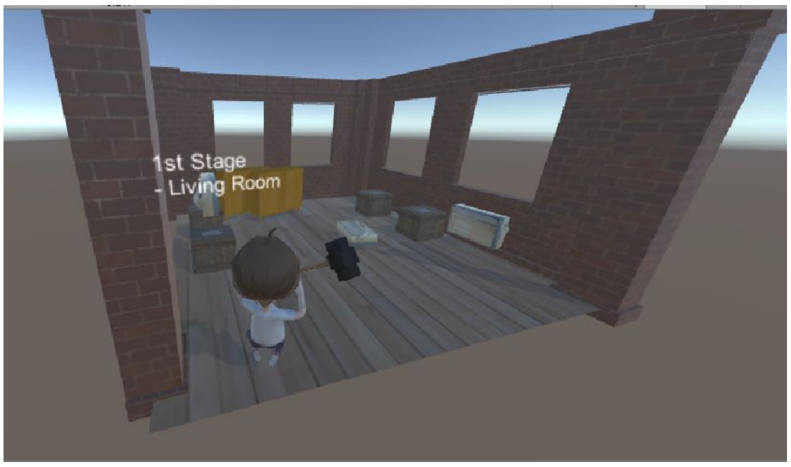
- Script File Name : <u>Start Script</u>
- What a Script does : It is a script that converts the scenes required for the game.



```
public void ChangeGameScene01()
{
    SceneManager.LoadScene("Stage 1");
}

public void ChangeGameScene02()
{
    SceneManager.LoadScene("Stage 2");
}

public void ChangeGameScene03()
{
    SceneManager.LoadScene("Stage 3");
}

public void ChangeGameSceneEnd()
{
    SceneManager.LoadScene("End");
}
```

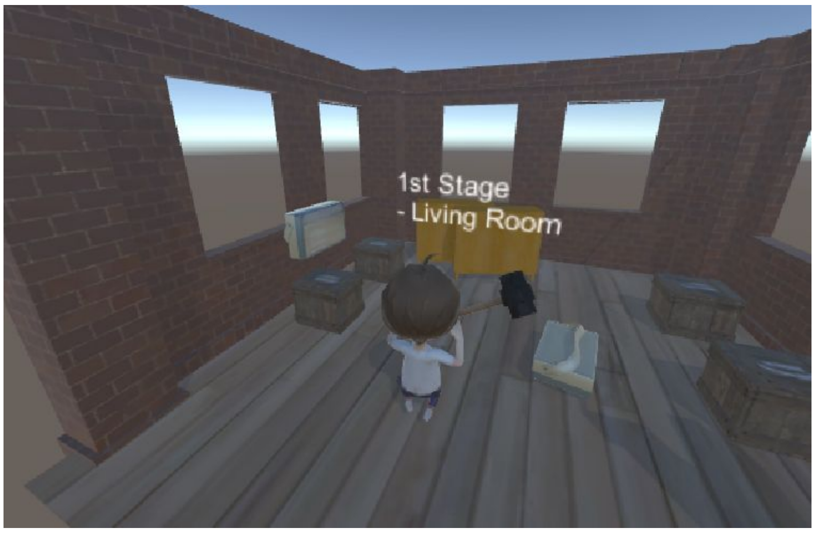
6. Animation

- ➢ Character movement

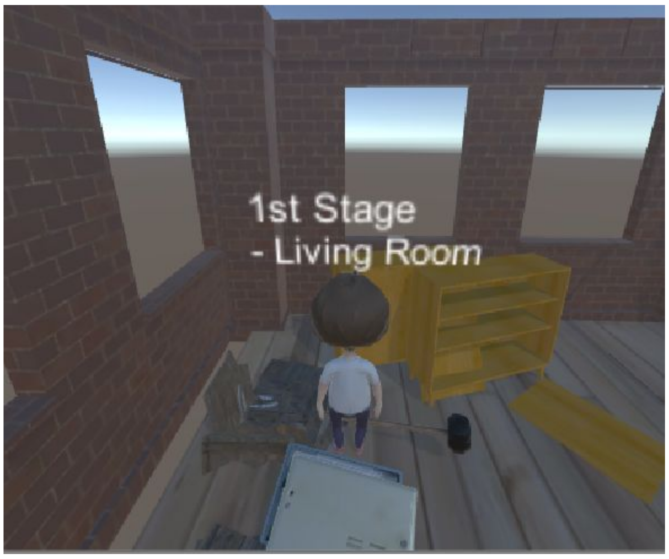


When we start game, this is the position of the character on screen.

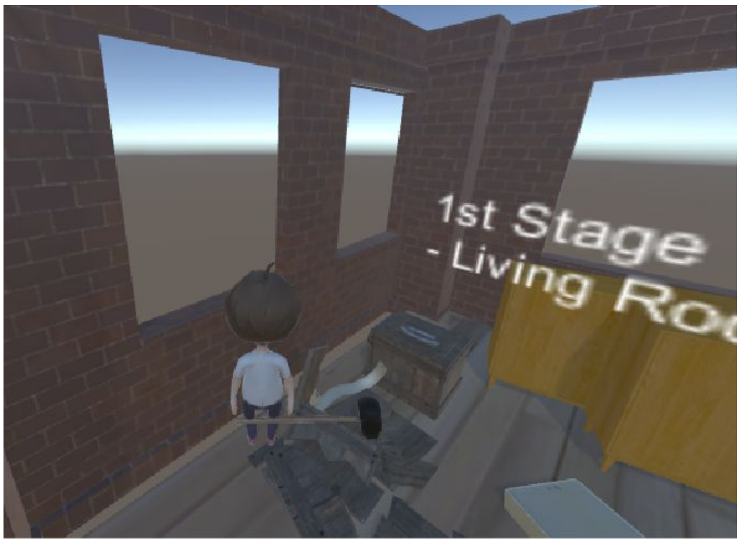
- When we move right (press ‘D’)
    - You can see that the character has moved to the right.



- When we move right (press ‘W’)
    - You can see that the character has moved to the forward.



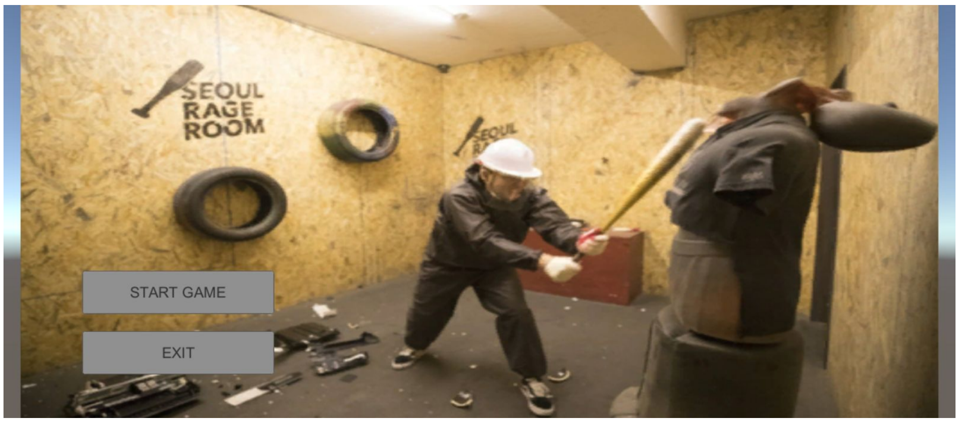
- When we move right (press ‘A’’)
    - You can see that the character has moved to the left.



7. Actual implementation screens and descriptions

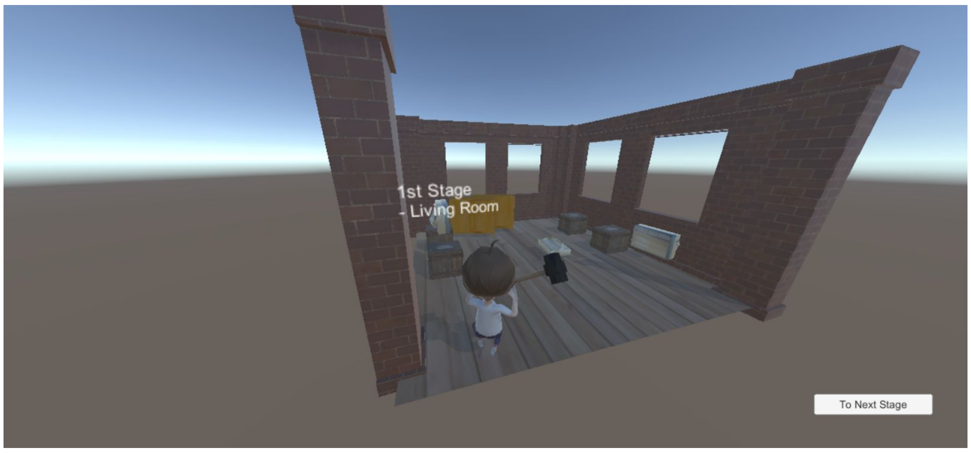
➢ Actual implementation Screens

➢ Intro

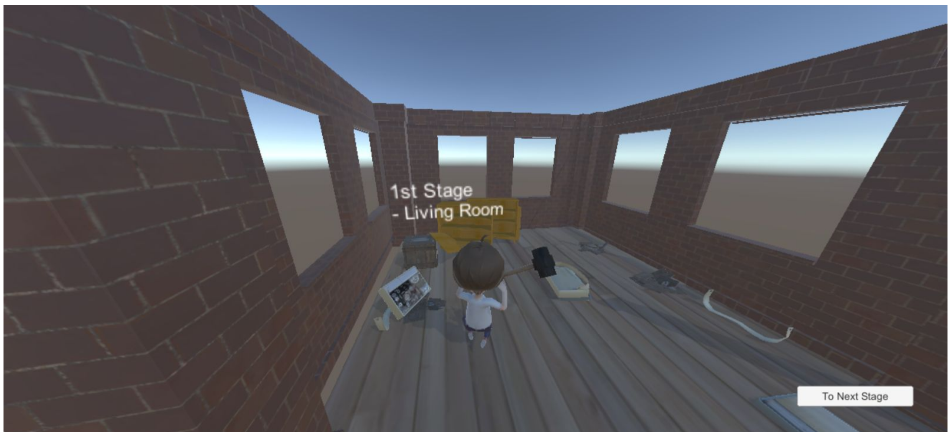


- It is the start screen. Press the Start Game button to go to Stage 1.

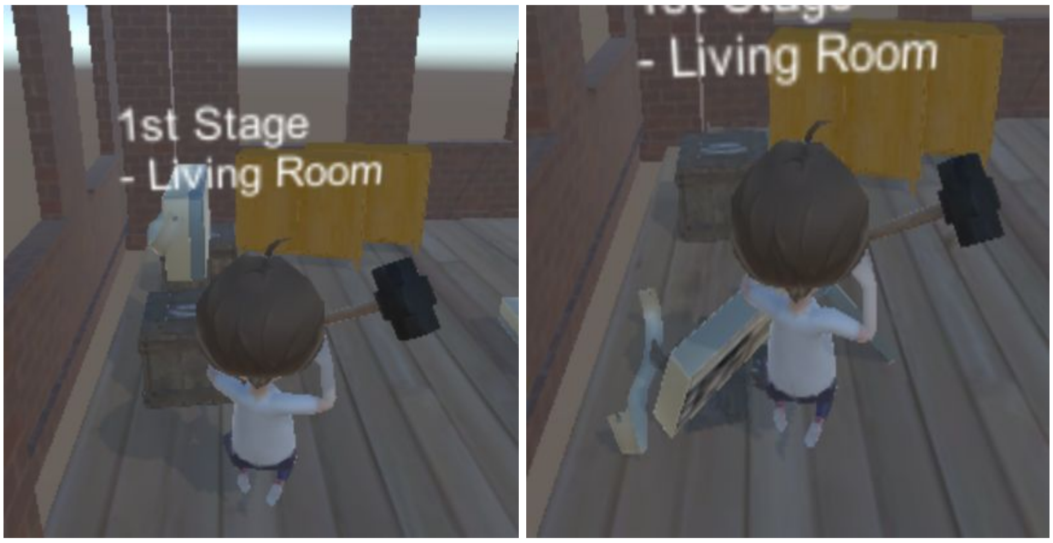
➢ Stage 1



- It is the first start screen of Stage 1. Stage 1 is a room with a living room theme, and items such as a radio and a cabinet are arranged for the living room. The user can operate the character using the keys of A, W, and D. When you reach the obstacle with the weapon, the obstacle is broken.



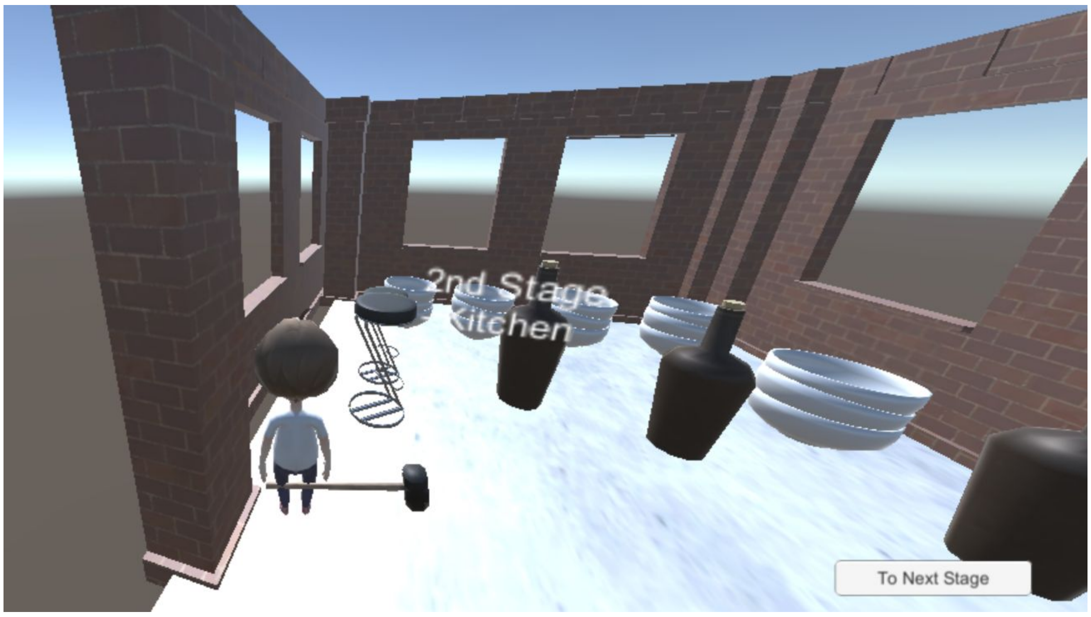
- On this screen, you can see that all obstacles have been broken normally. After breaking all obstacles, go to Step 2 through the To Next Stage button.



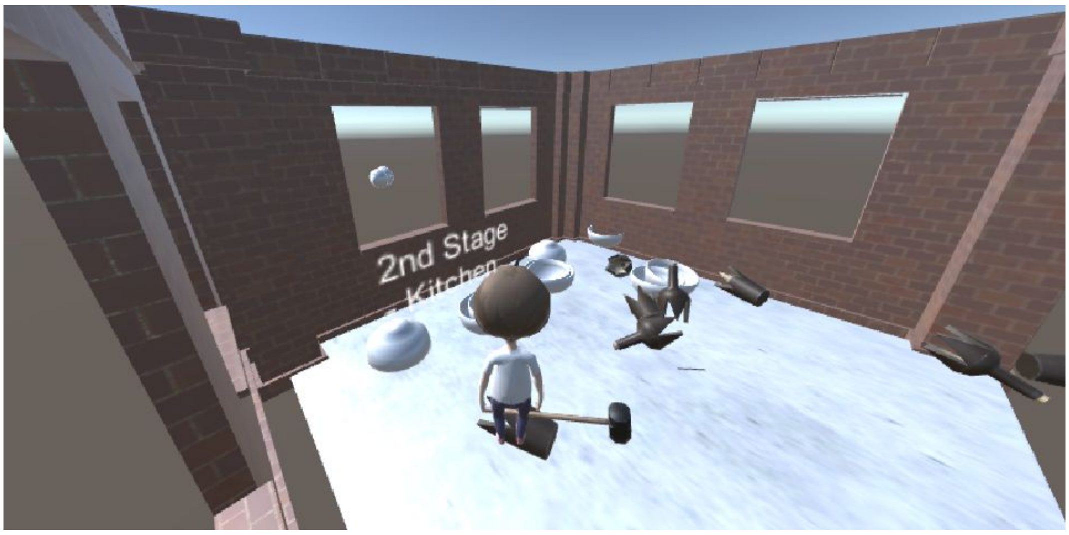
[Left: Before crash, Right: After crash]

- We will show you the details of the crash scene. At first, the obstacle is not broken. After approaching the obstacle, you can see the obstacle is broken

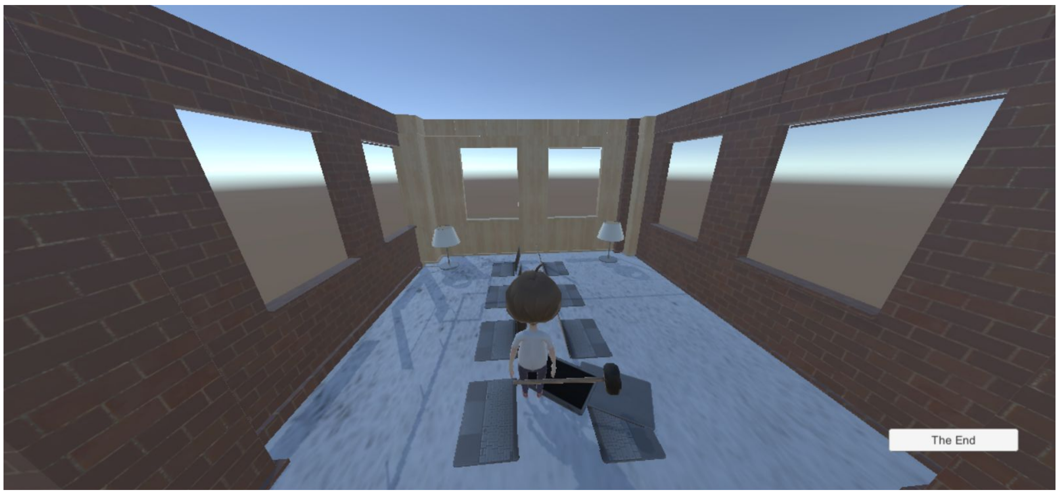
➢ Stage 2



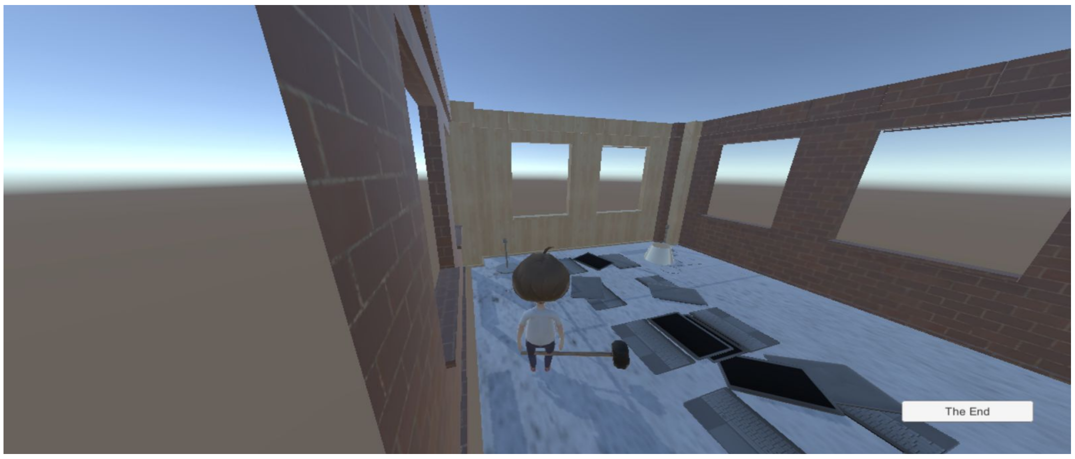
- At the start of stage 2, the character moves to the second level room. Stage 2 is a room with a kitchen theme. Accordingly, there are utensils such as bowls, sake bottles, and users can break this.



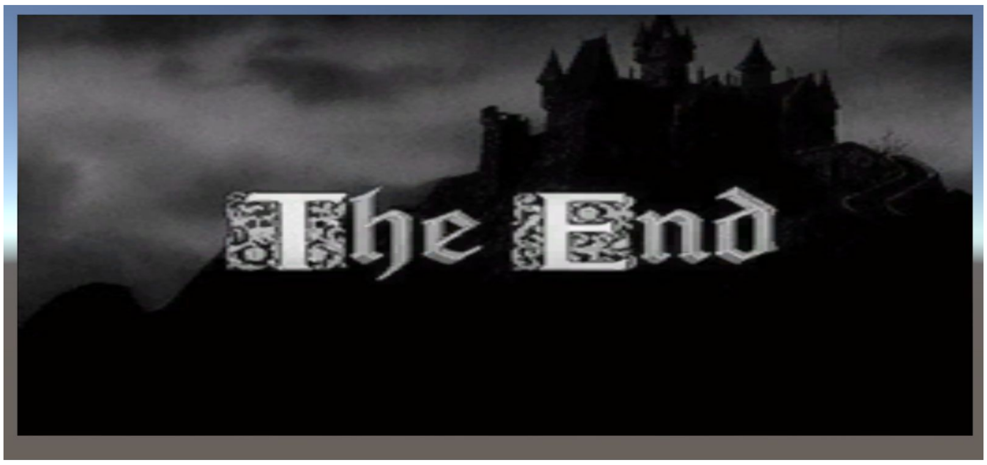
- You can see that all obstacles have been broken. Similarly, click the To Next Stage button to move to Stage 3.



- Stage 3 is a PC room concept. Stage 3 ends when you break the notebook and the light in here.



- This Screen shows all obstacles on Stage 3 are broken. Click the End button to see the ending screen.



- The ending screen comes out and the game ends.

➢ Project Conclusion

Most people are stressed from a lot of work. So we proposed a game to relieve stress by destroying things in virtual space.

The user can use a weapon to destroy objects in each virtual space at stages 1 , 2 and 3 and end the game.

The game is played from the first person perspective and animation is performed according to the movements of the game character. When a character's weapons and obstacles come into contact, a collision function is performed and obstacles are destroyed.

Users can relieve stress through the collision and animation features we implement when an avatar is blowing things.