Tech document

This is a Reversi game made by Unity. The below will explain how I made this game. There are 4 classes to make this game.

1. Game_manager: Control the whole game flow and game logic.
2. UI_control: Control the user interface, including two switches, one button, and the result scene.
3. Disk_obj: Stores all information that a disk need and control the animation of the disk.
4. Number: To show the score of both players.

Game_manager

Members

```
int player;
int mode;
public static bool music;
public static bool hint;
public static int winner;
public Number[] nums;
public Number num_prefab;
public Disk_obj[] disks_objs;
public Disk_obj disk_prefab;
Dictionary<int, int> disk_count;
List<Vector2Int>[] valid_pos;
Dictionary<Vector2Int, List<Vector2Int>>[] between_disks;
```

1. Player: The current player of the game.

```
// disk
const int NONE = 0;
const int BLACK = 1;
const int WHITE = 2;
```

2. Mode: The current mode of the game

```
// game mode
const int END = 0;
const int GAME = 1;
```

3. Music: The switch of the music. True means turn on, and vice versa.
4. Hint: The switch of the hint of the valid place to move. True means turn on, and vice versa.

5. Winner: The winner of the game
6. Nums: An array of 4 Numbers. Nums[0] is the first number of player black. Nums[1] is the second number of player black. Nums[2] is the first number of player white. Nums[3] is the second number of player white.
7. Num_prefab: The basic of every Number
8. Disk_objs: An array of 64 Disk_obj. Initialization is done with 60 NONE, 2 BLACK, 2 WHITE.
9. Disk_prefab: The basic of every Disk_obj
10. Disk_count: How many disks the both players have, and how many empty spaces there are
11. Valid_pos: The valid places to move of the current player
12. Between_disks: The disks that should be flipped when the player places disk

Game Flow

```
if (mode == GAME)
{
start_again:
    FindValidPos();
    UpdateScore();

    // detect board clicked
    for (int x = 0; x < 8; x++)
    {
        for (int y = 0; y < 8; y++)
        {
            if (disks_objs[y * 8 + x].clicked)
            {
                ClickBoard(new Vector2Int(x, y));
                disks_objs[y * 8 + x].clicked = false;

                goto start_again;
            }
        }
    }
```

In every game loop, first I will check the valid position of both player and then update the score of both players. When the board is clicked, it will process the position in the ClickBoard function, turn off the signal, and go to the start of the game loop again.

```
// show or hide hint
if (hint)
    ShowHint();
else
    HideHint();
```

After that, it detects the hint switch to decide whether to show the hint.

```
// end game, when there is no valid place for both players
if (valid_pos[player].Count == 0 && valid_pos[player % 2 + 1].Count == 0)
{
    mode = END;
}
```

Finally, it will check if the game is end or not by checking the valid position of both players.

End game

```
else if (mode == END)
{
    // check winner
    if (disk_count[BLACK] > disk_count[WHITE])
    {
        winner = BLACK;
    }
    else if(disk_count[BLACK] < disk_count[WHITE])
    {
        winner = WHITE;
    }
    else
    {
        winner = 3; // tie
    }
}
```

When the game ends, check who is the winner.

Functions

```
void FindValidPos()
{
    valid_pos[WHITE] = new List<Vector2Int>();
    valid_pos[BLACK] = new List<Vector2Int>();
    between_disks[WHITE] = new Dictionary<Vector2Int, List<Vector2Int>>();
    between_disks[BLACK] = new Dictionary<Vector2Int, List<Vector2Int>>();

    for (int y = 0; y < 8; y++)
    {
        for(int x = 0; x < 8; x++)
        {
            int index = y * 8 + x;

            if(disks_objs[index].color != NONE)
            {
                for(int i = 0; i < 8; i++)
                {
                    FindByDir(x, y, disks_objs[index].color, UP + i);
                    //Debug.Log("COLOR: " + disks_objs[index].color + " POS: "
                }
            }
        }
    }
}
```

To find the valid position of both players, I will first clear all the previous valid position and the disks should be flipped, check for 8 direction for each disk, and use FindByDir to process the position of one certain direction.

```
//dir
const int UP = 0;
const int DOWN = 1;
const int LEFT = 2;
const int RIGHT = 3;
const int CROSS_UP_LEFT = 4;
const int CROSS_UP_RIGHT = 5;
const int CROSS_DOWN_LEFT = 6;
const int CROSS_DOWN_RIGHT = 7;
```

```
bool FindByDir(int x, int y, int color, int dir)
```

FindByDir will keep moving along the assigned direction. If the position has opposite color of the player's, it will keep moving. If the position has the same color of the player's, it will stop and won't update the valid position because it means there is no valid position in this direction. If the position is empty, it will also stop to update the valid position and the disks that should be flipped in between.

```
void UpdateScore()
{
    nums[0].num = disk_count[BLACK] / 10;
    nums[1].num = disk_count[BLACK] % 10;
    nums[2].num = disk_count[WHITE] / 10;
    nums[3].num = disk_count[WHITE] % 10;
}
```

UpdateScore simply shows the current scores of the both players on the user interface.

```
void ClickBoard(Vector2Int pos)
{
    // add new disk, when point to a valid place
    if (valid_pos[player].Count != 0 && IsValidPos(player, pos))
    {
        // add disk
        disks_objs[pos.y * 8 + pos.x].color = player;
        disk_count[player]++;
        disk_count[NONE]--;

        List<Vector2Int> between = between_disks[player][pos];
        for (int i = 0; i < between.Count; i++)
        {
            int index = between[i].y * 8 + between[i].x;

            disks_objs[index].GetComponent<Animator>().SetTrigger("flip");
            disks_objs[index].color = player;
            disk_count[player]++;
            disk_count[player % 2 + 1]--;
        }
        // take turn
        player = player % 2 + 1;
        disk_prefab.GetComponent<Animator>().SetTrigger("flip");
```

If the disk (empty or not) is clicked, the ClickBoard is called. It will make sure the clicked position is valid. If it is valid, the player can place a disk and flip the disks in between. When all the flipping is done, the current player changes.

```
// take turn, when there is no valid place for current player
else if (valid_pos[player].Count == 0 && valid_pos[player % 2 + 1].Count != 0)
{
    player = player % 2 + 1;
    disk_prefab.GetComponent<Animator>().SetTrigger("flip");
}
```

When the current player has no valid place to place the disk, and the current player will also change.

```
void AddDisk(int color, int x, int y)
{
    int index = y * 8 + x;

    disks_objs[index] = Instantiate(disk_prefab, ScreenPosition(new Vector2Int(x, y)), Quaternion.identity);
    disks_objs[index].color = color;
    disks_objs[index].disk_pos = new Vector2Int(x, y);
}
```

AddDisk will create a disk at an assign geometry.

```
Vector2 ScreenPosition(Vector2Int grid)
{
    Vector2 center00 = new Vector2(-3.9f, 3.9f);
    float width = 1.12f;
    float height = 1.12f;

    float x = center00.x + grid.x * width;
    float y = center00.y - grid.y * height;

    return new Vector2(x, y);
}
```

ScreenPosition returns the position that I want a disk to be place according to the index of the disk.

```
void ShowHint()
{
    HideHint();

    for (int i = 0; i < valid_pos[player].Count; i++)
    {
        int index = valid_pos[player][i].y * 8 + valid_pos[player][i].x;
        disks_objs[index].GetComponent<Animator>().SetBool("show_hint", true);
    }
}

2 references
void HideHint()
{
    for(int i = 0; i < 64; i++)
    {
        disks_objs[i].GetComponent<Animator>().SetBool("show_hint", false);
    }
}
```

ShowHint will first set every hint invisible to clear the old hint and then show the valid position of current player.

UI_control

```
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    if (gameObject.name == "winner")
    {
        gameObject.transform.position = new Vector3(-100, 0, 0);
    }

    GameObject.Find("background_music").GetComponent<AudioSource>().Stop();
}
```

When game starts, it will turn off the music and set the winner result image far away that it is invisible for the player.

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    if(gameObject.name == "winner" && Game_manager.winner != 0)
    {
        sprite_rederer = gameObject.GetComponent<SpriteRenderer>();
        sprite_rederer.sprite = sprite_texture[Game_manager.winner % 3];
        gameObject.transform.position = new Vector3(0, 0, 0);
    }
}
```

Every time it updates, it will check whether the winner is out. If the result is out, it will show the winner result image.

```
private void OnMouseUp()
{
    if(gameObject.name == "music_button")
    {
        sprite_rederer = gameObject.GetComponent<SpriteRenderer>();

        if (sprite_rederer.sprite == sprite_texture[0])
            sprite_rederer.sprite = sprite_texture[1];
        else
            sprite_rederer.sprite = sprite_texture[0];

        Game_manager.music = !Game_manager.music;
        Debug.Log("music" + Game_manager.music);

        AudioSource bgMusicAudioSource = GameObject.Find("background_music").GetComponent<AudioSource>();
        if (Game_manager.music)
        {
            bgMusicAudioSource.Play();
        }
        else
        {
            bgMusicAudioSource.Stop();
        }
```

When the music switch is clicked, it will change the image as well as the music status. The hint switch is done in the same way.

```
else if(gameObject.name == "new_game_button")
{
    SceneManager.LoadScene("GameScene");
}
```

If the new game button is clicked, it will reload the game scene, which starts the game all over again.

Disk_obj

```
public int color;     // 1 for black, 2 for white
public bool clicked;
public Vector2Int disk_pos;

Unity Message | 0 references
private void Start()
{
    GetComponent<Animator>().SetInteger("color", color);
    clicked = false;
}

Unity Message | 0 references
private void Update()
{
    GetComponent<Animator>().SetInteger("color", color);
}
```

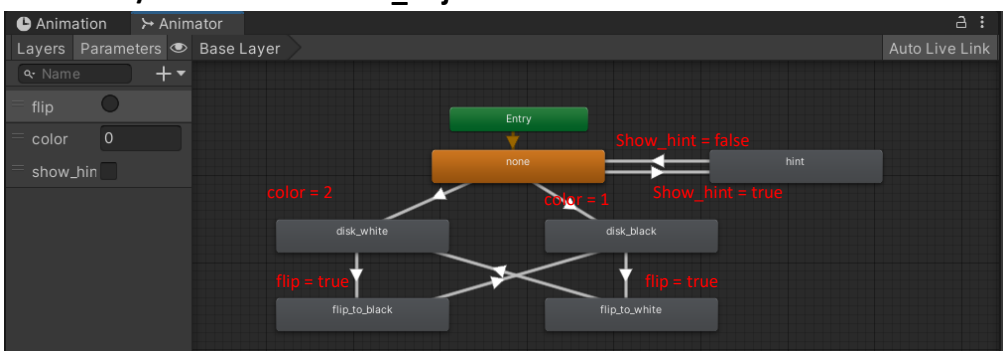
Disk_obj stores the information a disk needs, including color of the disk, whether the disk is clicked, and geometry relative to the board.

In every update, it detects if the color changes.

```
private void OnMouseUp()
{
    clicked = true;
    //Debug.Log("click"
}
```

If a disk object is clicked, it will set “clicked” attribute as true;

Animator / animation of Disk_obj



Number

```
public class Number : MonoBehaviour
{
    SpriteRenderer sprite_rederer;
    public Sprite[] sprite_texture = new Sprite[10];
    public int num;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        num = 0;
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        GetComponent<SpriteRenderer>().sprite = sprite_texture[num];
    }
}
```

What this class does is simply changes the image with the given number.