Week 3 Diary

📝 Plan:

~~1. Randomly generated mazes.~~ (fixed maps ☑)

~~2. Move of cameras.~~

3. Control of characters. ☑

4. Collision detection tests. ☑

5. Rules design. ☑

🚫 Problem:

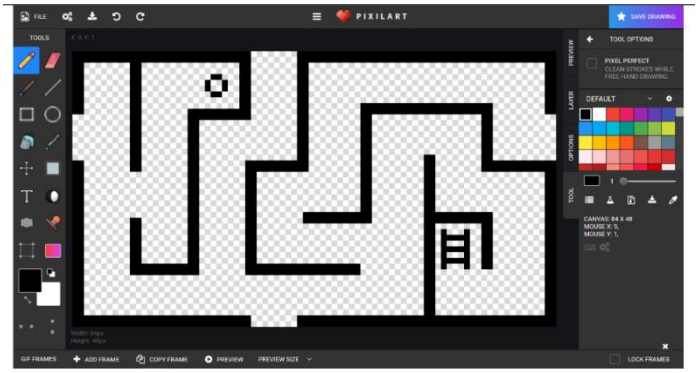
LPC1768 has a 16K RAM, which only supports an array smaller than 130*130. There's no way but to load maps dynamically. But due to the high writing-time-delay and the requirement of high refreshing rates, this cannot be achieved.

I must change my idea from multiple rooms in one layer into one per layer.

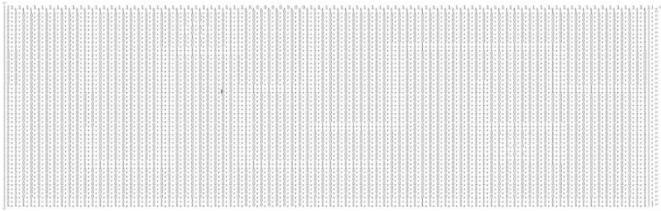
🎁 Outcome:

1. Fixed mazes.

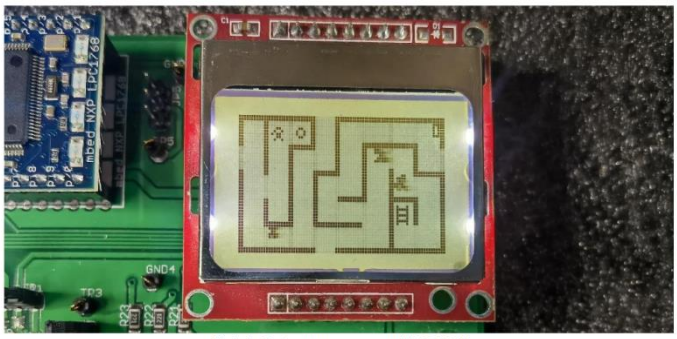
Store as const-variables, only occupy ROM space, Fig.1.



First. Drew pixel-maze



Second, I wrote a program to transfer the image into C++ 84*48 binary-array



Third. Outcome, map on LPC1768

Fig.1 Production of one map

2. Control of characters.

By setting the velocity of x- and y-direction, each frame updates the new position, Fig.2.

```
switch(input->joystick->getDirection())
{
    case LEFT:
        player.vx = -1;
        player.facingLeft = true;
    break;
    case UP_LEFT:
        player.vx = -1;
        player.vy = -1;
        player.facingLeft = true;
    break;
    case DOWN_LEFT:
        player.vx = -1;
        player.vy = 1;
        player.facingLeft = true;
    break;
    case RIGHT:
        player.vx = 1;
        player.facingLeft = false;
    break;
    case UP_RIGHT:
        player.vx = 1;
        player.vy = -1;
        player.facingLeft = false;
    break;
    case DOWN_RIGHT:
```

Fig.2 Control (part)

3. Collision detection*.

Each entity is regarded as a rectangular shape, judging whether they are overlapped, Fig.3.

```
// Collision test between entites and map
void Game::moveWithCollisionTest(Entity* entity, const bool map[HEIGHT][WIDTH])
{
    int x = entity->x;
    int y = entity->y;
    int steps = abs(entity->vx); // how many units (pixels) the entity should move in said direction
    bool collision; // true if colliding

    // Check x-axis
    if (entity->vx > 0) // moving right
    {
        int entityRight = x + entity->width - 1; // Need to check right border of entity, since it is moving right

        while(steps--) // While it still have more movement left
        {
            collision = false;

            // Wrapping
            if (entityRight+1 >= WIDTH)
                entityRight = -1;// wants entityRight = -1, so next check is entityRight 0*/

            for (int i = 0; i < entity->height; ++i) // Loop through all vertical points on the right hand side of the entity (y+i
            {
                if (map[y+i][entityRight+1]) // If moving to the right leads to collision for given y+i
                {
                    // Slope + allows player to climb to top of platform by going right if it hits close to top of wall.
```

Fig.3 Collision detection (part)

4. Others.

Ⅱ Properties of characters, enemies, gun ballets.

Ⅱ Scoring system.

Ⅱ Games start, refresh, and end conditions.

Ⅱ Sprites of characters and enemies.

📌 **Important:**

I adapted an existing game engine, I referred to the general structure of codes, but I modified a lot. In the final version, I will show the differences in code submission and note the reference.