SHOOT MONSTER

This game uses NUI (natural user interface) that consist of an interactive gun and the software displaying graphics on the screen. The aim was to build entertaining NUI using microbits and Processing software. Processing is responsible for a graphical interface, whereas one microbit is implemented into the gun, and the other passes data from the gun to the computer. The game is also playable only with a gun.

Interaction Overview

Figure 1

The interaction starts with plugging one microbit for communication into the computer, starting running the program in processing, and turning on the gun prototype, which is powered by batteries and starts by pressing the bottom button on its embedded microbit.

The next steps are pictured in figure 1. The first step after the gun is turned on is to calibrate the compass by tilting the prototype until the screen is all filled with red dots. Step 2 is pressing the left button to start a new game. This button can be pressed any time to reset the game if needed. Step 3, the game has begun, and the player has 30 seconds to shoot as many monsters as possible (for each monster earns 10 points). Step 4, by pointing a gun in the left-right and up-down direction, the player aims. An aiming cross is displayed on the screen, and also the monster appears on the microbit's screen

when it is aimed. Step 5, by pressing the right button, the player shoots. The gun has five bullets, and its current status can be seen in the bottom right corner. When the player runs out of ammo, microbit displays a cross (step 6). Shaking the gun reloads it (step 7). At the end of the game, the score is displayed on the screen and on the gun and the game can be repeated by pressing the left button. The game is possible to play without the screen since the monsters appear on the microbit controller. Figure 2 shows a detailed picture of the screen during the game.

Figure 2

Parts List

Hardware

- 2x micro:bit
- Micro: bit's screen, two buttons and accelerometer
- HP computer screen (optional)
- USB Cable
- Batteries

Software

- Processing 3.5.4
- https://makecode.microbit.org/#editor

Other

- Carton box for creating the interactive gun prototype

Process

This project aimed to create a natural user interface. That means to make interaction to feel natural and easy to use. As mentioned, the game's whole setup consists of two microbit controllers, where one is embedded in the gun prototype. The second receives information in the form of Strings from the prototype and passes them to the processing application that reacts accordingly.

The whole game can work only from the gun prototype, which displays the monster when correctly aimed. After turning on the gun prototype, the program runs compass calibration because it uses the compass position to orient horizontally. The compass returns numbers from 0 to 359 according to the direction the gun is facing. The pitch() function is used for vertical orientation, and it returns numbers from -180 to 180 according to its pitch up or down (vertically). When the left button is pressed to start the game, it determines the playing space for the game; its centre is always determined by the direction where the gun is facing at the moment when the button was pressed. The playing space is then 50 degrees right and left and 60 degrees up and down.

The game always generates 20 monsters with a random position in the determined playing space. To generate random placing in X-axes(left/right) and Y(up/down) axes, it uses mathematical conversions to the current playing space (Appendix 1). There generated 3 Arrays, all containing 20 elements. One keeps the information about the position in X and the other about Y of the monster's position. The third one holds the status, whether the monster is alive or not. Indexes of these arrays represent individual monsters. Pressing the left button also starts the 30-second countdown, which is the time for one game.

The whole game setup is then sent through radio communication to the second microbit. The strings containing information about monsters' placement are too long to be sent all at once; thus, while loops were used to send those strings in smaller pieces (Appendix 2). Single strings are connected again to the one string after all of them are received in the microbit connected to the computer, and it then passes them to the processing (Appendix 3).

When the game is running, the forever() function is used to constantly update and send the position of the direction where the gun is pointing. It also goes through else-if statements and displays the monster on the microbit's screen when the monster is aimed. The for loop implementation for this task was tested, but it caused some lagging; therefore, the else-if statement solution remained.

The right button indicates a shooting. When it is pressed, it checks whether any monster was hit; if so, it changes its alive status and sends the information to the processing as an index of the monster that was hit. It also decreases the number of bullets. When the player runs out of bullets and presses the right button, it displays a cross to indicate to reload. Reloading is done by shake gesture—the event for shaking rewrites the number of bullets back to 5 and sends the information about reloading.

After 30 seconds from pressing the start button, the game is stopped by the if statement that is in the forever() function. It compares the current time with the time that was set at the beginning. The score is then displayed on the microbit controller and sends information about the end of the hame to processing, which also shows the score on the screen.

When the processing receives the information, it responds by displaying corresponding animations. In the beginning, the processing prepares all the graphics and other properties. When the start button is pressed, processing receives information about the gun's position and monster placing. It then must convert these values into appropriate ranges. For X, it is in the range from 0 to 100 and for Y to -60 and 60. These values must be further converted to the number corresponding to pixels on the game screen. This is done by using the map() function. It also constantly receives real-time information about the direction the gun is pointing. These values must also be converted in the same way as mentioned. There appears dilatation in receiving these data; easing was used to suppress this effect (Appendix 4). Using averaging of incoming values was also considered, but it did not serve its purpose since the most actual position is needed. The compass data are too easily influenced by other magnetic fields, which is causing slight jittering. Processing further responds with proper graphics according to the received data.

Appendix

```
// Function used to determin monster's x position
function monsterPositionX(rangeL: number) {
    let position
    position = rangeL + Math.floor(Math.random()*100)
    if (position > 359) {
        position = position - 359
    }
    return position
}

// Function used to determin monster's y position
function monsterPositionY() {
    // This is the lowest angle vertically
    let lowerAngle = -60
    // The span vertically is 120 degrees from -60 to 60
    let span = 120
    let randomY = lowerAngle + Math.floor(span*Math.random())
    return randomY
}
```

Appendix 1 – rangeL represents the leftmost side of the game, which is centre (centre determined by gun position when the left button is pressed to start the game) minus 50 degrees.

```
// Radio communication can send only strings 19 char long, so this is used to cut the string into smaller parts
radio.sendString("?"+ gameRangeL + ";")
while (lenX > 0) {
    pause(100)
    radio.sendString("?" + posXStr.substr(lenXindex,length))
    lenX -= 16
    if (lenX < 16) {
        lenXindex += 16
        length = lenX+1
    }
    else {
        lenXindex += 16
    }

}
radio.sendString("?" + ";")
length = 16
while (lenY > 0) {
    pause(100)
    radio.sendString("?" + posYStr.substr(lenYindex,length))
    lenY -= 16
    if (lenY < 16) {
        lenYindex += 16
        length = lenY+1
    }
    else {
        lenYindex += 16
    }
}
radio.sendString("@")
```

Appendix 2 – splitting strings into smaller parts for radio communication.

```
// Used to receive information from the gun's Mbit  and pass it to computer
radio.setGroup(42)
let bufString = ""

radio.onReceivedString(function (receivedString: string) {
    if (receivedString.charAt(0) == '?') {
        bufString += receivedString.substr(1).trim()
    }
    else if (receivedString.charAt(0) == '@') {
        serial.writeLine("@" + bufString)
        bufString = ""
    }
    else {
        serial.writeLine(receivedString)
    }
})
```

Appendix 3 – the code inside of the receiving microbit

```
// Easing for aiming cross
x += (posX - x) * easing;
y += (posY - y) * easing;
```

Appendix 4 -Easing for X and Y axes