Tutorial 9: Simon explained

http://www.codemii.com/2008/12/30/tutorial-9-simon-explained/

By now you should have the skills needed to start coding or modifying small games and utilities. In this Wii programming tutorial we will go through the Simon game and give a brief explanation of what things do. In this tutorial we focus on how to use the code we know to achieve certain the things that we are after.

Lets begin by firstly downloading the simon source code and opening up main.c. This version of Simon is intended to run on gcube. You can load the dol file with gcube and have a little play around.

Simon is a basic game; you have 4 colours and the computer generates a sequence of colours which you need to press correctly to win. The computer shows you one colour and when you press the correct colour the computer then shows you the next colour and so on.

A quick summary of this Simon game would be:

```
Show current sequence colour
        If sequence_counter is less than level_counter then increment sequence_counter
        If sequence_counter equals level_counter then wait for user input

Wait for user input
        If colour correct, increment user_counter
                If user_count equals level_counter then increment level_counter and show new
sequence
        If colour incorrect, return to main menu
        If user_counter equals sequence_length, game won
```

As usual we start off with our standard includes and also our graphical files. These graphic files were made with a program called bmp2gc which converts .bmp images to .h files which can be used with the gamecube/Wii. I would recommend you stick with using JPEGs instead of using bmp2gc.For the purpose of understanding the Simon game as it was coded, I've left most things as it was.

```
#include <gccore.h>
…
#include "main.h"

// Graphics
#include "simon.h"
…
```

Next up we define how long we want the sequence to be, in this case 15. So the user will have to get the colours right until they reach 15 colours in a row.

```
#define SEQ_LENGTH 15
```

Our standard video variables

```
// 2D Video Globals
static u32 *xfb;
static GXRModeObj *rmode;
```

Now we declare our global variables. Firstly we define an array of characters that is 15 elements long. The next variable defined is the different colours; there are 4 colours. Red is 0, Green is 1, Blue is 2 and Yellow is 3.

Next up we have the sequence counters, the first seqnum is used when we are animating simon to show the user and seqlevel stores the ‘level’ the user is on. If the user has a sequence to play back of 210, they are up to level 3. If the sequence is up to 2102, they are up to level 4, etc.

Userseqnum stores where the user is up to in the sequence. The next booleans are to be used when we allow the user to press a button - userinput, if the user entered the correct input – correctinput and if the user is in the game – ingame.

```
// Vars
char seq [SEQ_LENGTH];
char color [4] = "0123";
int seqnum, seqlevel, userseqnum;
bool userinput, correctinput, ingame;
```

We then have the usual initialise function.

```
void Initialise() {
		…
}
```

We have a SetScreen function which just flushes everything we wrote to the framebuffer to the screen. Note that in this Simon game things aren’t done as correctly as they should be. Generally you should clear the screen and then write to the framebuffer and display it but since there aren’t any cursors or moving objects it’s not really needed.

```
void SetScreen() {
		VIDEO_SetNextFramebuffer(xfb);
		VIDEO_Flush();
		VIDEO_WaitVSync();
}
```

Now we’ll jump right down the bottom to the main function which is quite short. We call initialise, initialise the controller and call another function which handles restarting our gamecube but as we aren’t running this on a gamecube it doesn’t really matter.

Below that we have our drawpic function which has a similar concept as display_jpeg in tutorial 5. drawpic takes the x position (93), y position (0), the images height (48), the images width (120) and the actual picture to display (simontext) which is the title screen image. After that we call our game function and return 0 if the user ever decides to quit.

```
int main () {

        Initialise();
        PAD_Init();
        SYS_SetResetCallback (reset_cb);

        drawpic(93, 0, 48, 120, simontext);

        game();

        return 0;
}
```

Before exploring the game function we need to animate the title screen and initialise our variables. In the first run of the game function all our ingame variable is false so it skips all the code in game() and runs animate_simon().

```
void game() {
        u32 button = 0;

        while (1) {
                while (ingame == true) {
                        …
        …

        // First run or Game over? Then go back to menu
        animate_simon();
        restart_game();
}
```

Animate simon makes the lights spin on Simon and it acts as a menu screen waiting for the user to press A to begin. So we firstly need to know what colour light is currently lit, a variable which we can use to make the spinning go faster or slower and if this menu screen is running.

```
// Animate the "spinning" lights on simon
void animate_simon() {
        int count = 0;
        int multipler = 0;
        bool running = true;
        u32 button = 0;
```

We display our starting text which says “Press A to start” and begin our while loop. We use our light counter (count) and if it’s 0 we show the red light, if it’s 1 we should the green light, etc. We then increment this counter and when it reaches a count of 3 it will be reset to -1 which then is incremented immediately to 0.

```
        drawpic(96, 410, 27, 120, starttext);

        while (running == true) {
                if (count == 0) {
                        drawpic(60, 60, 336, 192, simonred);
                }
                if (count == 1) {
                        drawpic(60, 60, 336, 192, simongreen);
                }
                if (count == 2) {
                        drawpic(60, 60, 336, 192, simonyellow);
                }
                if (count == 3) {
                        drawpic(60, 60, 336, 192, simonblue);
                        count = -1;
                }

                count++;
```

Now when we are animating Simon we need a pause between each light and we can do this using the usleep function. Usleep sleeps for x amount of microseconds. Now when we are sleeping it’s basically just pausing our whole program. But what if the user presses the A button when the program is sleeping? Nothing would happen.

We can avoid this in two ways, either use threads or possibly make the sleep times shorter and then query the user input in-between sleeps. I went with the easier second option. You can see below that we sleep in 20 short times. We have our multiplier which can speed up the animation if it’s increased and we query if the A button was pressed.

```
// Sleep a little bit in the for loop so we can grab user input
int i = 0;
for (i = 0; i< 20; i++) {
        usleep(25000 - (multipler * 100));
        PAD_ScanPads();
        button = PAD_ButtonsDown(0);
        if (button == PAD_BUTTON_A) {
                running = false;
        }
```

After that we have the controller input code which works for either joystick, if it's pushed up the animation speeds up and if pushed down the animation decreases.

```
        // Go faster or slower according to user input
        else if ((PAD_StickY(0) > 18 || PAD_SubStickY(0) > 18) && multipler <= 245) {
                multipler++;
        }
        else if ((PAD_StickY(0) < -18 || PAD_SubStickY(0) < -18) && multipler >= 1) {
                multipler--;
        }
}
```

We then have our normal handling reboot/poweroff code which isn't important and then we display what we have in our framebuffer with SetScreen().

```
        // Handle reboot/poweroff
                if (event == 1) {
                        printf("POWER!\n");
                        reload();
                } else if (event == 2) {
                        printf("RESET!\n");
                        reload();
                }
                if (button == PAD_BUTTON_START)
                        reload();

                //printf("multipler = %i \n",multipler);

                SetScreen();
        }
}
```

So lets say that the user now wants to play, they press A and then running variable turns false which causes us to end the animate_simon function and move on to the restart_game function.

The restart game function just initialises all of our variables, sets the userinput to false as we will firstly show the user the sequence, set the correctinput to true and when they get the colour wrong we'll set it to false and they are in the game so we ingame to true.

As you might remember we don’t clear the screen so and we don’t want to display the Press A to start text anymore so what we show is a blank black image and then show a Simon image which doesn’t have any lights showing.

```
// Reset the variables and text
void restart_game() {
        seqnum = 0;
        seqlevel = 0;
        userseqnum = 0;
        userinput = false;
        correctinput = true;
        ingame = true;

        drawpic(96, 410, 28, 120, blanktext);
        drawpic(60, 60, 336, 192, simon);
```

Now we need to make the sequence that the user has to copy, it’s not something that we want to hardcode so we need to randomise it instead. We can use the current time to provide us with a ‘key’ and we’ll use the rand feature to generate us a random number from 0 to 3 which is our colours (rand() % 4).

Our sequences needs to be a of type char because we can’t have a sequence like 01230102321 because it starts with a 0 and it would be hard to say give me the second digit of this number without converting this number to a char. So we add ‘0’ to our rand() % 4 so that it will convert the 0 to 3 number into type char. We’ll then generate 15 of these numbers and add the number to the sequence.

```
        // Randomise the sequence
        srand((unsigned)time(0));

        int q;
        for(q=0; q<SEQ_LENGTH; q++){
                char random_integer = '0' + rand() % 4;
                seq[q] = random_integer;
                //printf("%c \n", random_integer);
        }
        printf("Sequence is %s \n",seq);
}
```

Now we can explore the game function. As the variable userinput is still false we can skip this part and go onto the part that shows the user which colour the first level of our sequence is. In our example we will use the sequence below:

Sequence	**0**	**3**	**1**	**1**	**2**	**3**	**2**	**0**	**2**	**1**	**0**	**1**	**2**	**1**	**3**
Level	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Seqlevel	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**

Seqlevel would equal 0 and seqnum (our counter) would be 0 as well. We now need a way to increase the speed of the animation of Simon when the level increases. We use usleep again and this time use seqlevel as the multipler. As you recall the seq variable has our sequence stored. As seq is of type char we can request the x position of this variable like seq[0] would be 0, seq[1] would be 3, seq[2] would be 1, etc.

In our first run we would sleep, compare seq[0] with colour[0] which are both 0 and that would make the if statement true and then we would display the Simon red light. After that's done we check to see if our counter is equal to the level the user is on and if it is we set userinput to true and reset the seqnum to -1 which will be incremented back to 0. We will sleep again for a while, draw the blank Simon image and increment seqnum so we can show the next colour to the user if needed.

```
// If we are playing the sequence to the user
if (userinput == FALSE) {
	usleep(1000000 - (seqlevel * 50000)); // Lower time every incrementation

	if (seq[seqnum] == color[0]) {
		drawpic(60, 60, 336, 192, simonred);
	}
	if (seq[seqnum] == color[1]) {
		drawpic(60, 60, 336, 192, simongreen);
	}
	if (seq[seqnum] == color[2]) {
		drawpic(60, 60, 336, 192, simonblue);
	}
	if (seq[seqnum] == color[3]) {
		drawpic(60, 60, 336, 192, simonyellow);
	}

	if (seqnum == seqlevel) {
		userinput = TRUE;
		seqnum = -1;
	}
	// Sleep a while and then show blank simon
	usleep(1000000 - (seqlevel * 50000));
	drawpic(60, 60, 336, 192, simon);
	seqnum++;
}
```

So now that we've shown the sequence to the user, all we need to do is check for which buttons the user presses, Y, X, B, or A. So say the user presses the A button which would be correct for the first level (userseqnum equals 0). Colour[0] is 0 and seq[userseqnum] is 0 so then correctinput stays as true.

```
while (ingame == true) {
        PAD_ScanPads();
        button = PAD_ButtonsDown(0);

        // If we are waiting for user input
        if (userinput == TRUE) {
                if (button == PAD_BUTTON_Y) {
                        drawpic(60, 60, 336, 192, simonred);
                        if (color[0] != seq[userseqnum]) {
                                correctinput = false;
                        }
                }
                if (button == PAD_BUTTON_X) {
                        drawpic(60, 60, 336, 192, simongreen);
                        if (color[1] != seq[userseqnum]) {
                                correctinput = false;
                        }
                }
                if (button == PAD_BUTTON_B) {
                        drawpic(60, 60, 336, 192, simonblue);
                        if (color[2] != seq[userseqnum]) {
                                correctinput = false;
                        }
                }
                if (button == PAD_BUTTON_A) {
                        drawpic(60, 60, 336, 192, simonyellow);
                        if (color[3] != seq[userseqnum]) {
                                correctinput = false;
                        }
                }
```

So now we detect if any button of the A, B, X, Y buttons were pressed. We check that correctinput still equals true which in this case it does. If the userseqnumber (0) is less than the seqlevel (0) then it would increase the userseqnum. In this case it's not true so we skip that and go to the next if statement which is true. We reset the userseqnum, increment the seqlevel (to 1) and set userinput to false as we would like to show the user the new sequence.

As seqlevel doesn’t equal SEQ_LENGTH (15) we’ll skip that part for now.

```
                    if (button == PAD_BUTTON_A || button == PAD_BUTTON_B || button ==
PAD_BUTTON_X || button == PAD_BUTTON_Y) {

                              usleep(500000);
                              drawpic(60,60, 336, 192, simon);
                              //printf("userseq %i seqlevel %i \n",userseqnum, seqlevel);

                              // If they entered the correct sequence
                              if (correctinput == TRUE) {
                                        // Increment the users seq number
                                        if (userseqnum < seqlevel) {
                                                  userseqnum++;
                                        }

                              // If user has matched the seq level, increase the seq level and show the
new sequence to the user
                                        else if (userseqnum >= seqlevel) {
                                                  userseqnum = 0;
                                                  seqlevel++;
                                                  userinput = false;

                                                  // Wait a little while
                                                  usleep(500000);

                                                  // If user has won
                                                  if (seqlevel == SEQ_LENGTH) {
                                                            …
                                                  }
                                        }
                              }
```

We show the new sequence to the user and then wait for their input again. This time around the if statement (userseqnum < seqlevel) is true so we increment userseqnum and wait for the user’s next input if they were correct.

We'll assume that one time that they were incorrect, we need to find what the correct colour was and show that to the user. So we compare seq[userseqnum] with 0, 1, 2, 3 until we find the right colour and then display that colour light. We then display the "You Lost" text, pause for a while and then show the blank simon and set ingame to false so we go back to the menu screen.

```
// Incorrect sequence
else if (correctinput == FALSE) {
        //printf("Sorry wrong key.... \n");
        usleep(1000000);

        //printf("Right key was %c \n",seq[userseqnum]);
        //printf("You lost! \n");

        if (seq[userseqnum] == '0') {
                drawpic(60, 60, 336, 192, simonred);
        }
        if (seq[userseqnum] == '1') {
                drawpic(60, 60, 336, 192, simongreen);
        }
        if (seq[userseqnum] == '2') {
                drawpic(60, 60, 336, 192, simonblue);
        }
        if (seq[userseqnum] == '3') {
                drawpic(60, 60, 336, 192, simonyellow);
        }

        drawpic(96, 410, 27, 120, youlost);

        usleep(15000000);
        SetScreen();

        drawpic(60, 60, 336, 192, simon);
        drawpic(96, 410, 28, 118, blanktext);

        usleep(500000);
        SetScreen();

        ingame = false;
}
```

Lets assume that the user has won the game by getting all the sequence correct. We check this by comparing seqlevel to SEQ_LENGTH (15) and if equal they have won. We display the “You Won” text and display the winning square by comparing the last level in the sequence with 0, 1, 2, and 3. We have a for loop to display the winning square, sleep a bit and then display the blank Simon image a few times so it’s shown as a blinking colour light. We pause for a while and then show the blank simon and set ingame to false so we go back to the menu screen.

```
// If user has won
if (seqlevel == SEQ_LENGTH) {
	//printf("You win! \n");
	drawpic(96, 410, 27, 120, youwin);

	// Blink the winning square
	int a;
	for (a = 0; a < 10; a++) {
		if (seq[SEQ_LENGTH-1] == '0') {
			drawpic(60, 60, 336, 192, simonred);
		}
		if (seq[SEQ_LENGTH-1] == '1') {
			drawpic(60, 60, 336, 192, simongreen);
		}
		if (seq[SEQ_LENGTH-1] == '2') {
			drawpic(60, 60, 336, 192, simonblue);
		}
		if (seq[SEQ_LENGTH-1] == '3') {
			drawpic(60, 60, 336, 192, simonyellow);
		}
		usleep(100000);
		SetScreen();

		drawpic(60, 60, 336, 192, simon);
		usleep(100000);
		SetScreen();
	}

	drawpic(96, 410, 28, 120, blanktext);
	usleep(500000);
	SetScreen();

	//printf("Press A to play again \n");
	ingame = false;
}
```

And there we have it, the Simon game explained in detail. All together it looks like a lot but there’s a few repeated if statements, functions, etc. It’s about programming in small components; you might start off with making the colour light up when you press a key. Then you might move on to comparing the key pressed to a certain colour and then move on to comparing the key pressed to a certain position in a character array, etc.

I hoped this tutorial has helped you understand the Simon game and how things were done. There are plenty of ways that this Simon game can be improved so feel free to modify the code and show us how you’ve improved the game. I didn’t expect this tutorial to be this long but what can you do. If you have any questions or comments feel free to post them.

PDF by Pvt Ryan (http://www.ninet.org),
Changes: Spelling corrections & Formatting.