Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)
Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)
Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)
Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)
Embedded Systems Community (/courses/UTAustinX/UT.6.01x/1T2014/e3df91316c544d3e8e21944fde3ed46c/)

When defining the variables used to store the state of the game, we collect the attributes of a virtual object and store/group them together. In C, the **`struct`** allows us to build new data types. In the following example we define a new data type called **STyp** that we will use to define sprites.

VIDEO 15.5 HOW TO CREATE, MOVE, & DRAW SPRITES

Help

0:00 / 9:03 1.0x

JONATHAN VALVANO: In this video, we're going

to show you how to create and manage sprites.

A sprite is a living entity which will exist in our game.

We're going to use the struct to create a data structure to hold our sprite.

So in our data structure, we have attributes,

like the x position, which is where in the world is our sprite, its y

position, what it looks like, and rather than storing

the whole image into the structure, you can

see we're going to just have a pointer to that structure.

But this image pointer will then point to what the sprite looks like,

whether it's a bunker, or an enemy, or a

missile.

And lastly, we have the status of this sprite.

We're going to have a very simple status in this initial demonstration,

where 0 means it's dead, and 1 means it's alive.

And so we're going to create this structure

by defining a new data type which has four fields.

We defined the struct, next, we're going to create a new type.

This is our sprite type.

So a type is like hair, long, short, except in this case,

it's going to create four elements of x, y, pointer, and life.

RAMESH YERRABALLI: The user defines type.

JONATHAN VALVANO: in this initial demo, we're going to create four sprites.

So we're going to create a RAM-based structure of four elements which

we will call the Enemy.

To manage the Enemy, we're going to initialize their position.

So we're going to have four, there's one, there's another one,

there's a third one, and here is a fourth one.

So this is my enemy, my array of sprites.

And I'm going to initialize their position,

and so I'm going to set their x-position along the row.

I'm going to set them so that they occur on the screen in a row,

and they're going to pre-set their y-position to be all in the same spot,

so they'll all be next to each other.

I'm going to set what they look like, they're all

going to look like Enemy30s, and then I'm

going to give them life by setting their life parameter to a 1.

So when I execute this function, these enemies will come to life.

RAMESH YERRABALLI: So Jon, how do I move an enemy

from one position to another position?

JONATHAN VALVANO: Now, I could access the individual locations

by taking the first enemy, taking its parameter,

and I can move it wherever I want.

So if I were to execute this code here, I could set a value here,

I could set a value there, and now Enemy number 0

is now teleported to that location.

But rather, I'm going to use this move function,

and it turns out, if you look very carefully at the BMP image,

each enemy has a border 2 pixels on the left, 2 pixels on the right,

1 pixel on the top, and 1 pixel on the bottom that are black.

So all of these sprites, all these images of these sprites,

have a black border around the edge.

This means, if I move an enemy 2 pixels to the left, 2 pixels to the right,

1 pixel down, or 1 pixel up, the movement of that enemy

will automatically erase where it was.

So in this function, I'm going to take all my enemies, 0 through 3,

and if the enemy is on the screen, if he's not too far to the right,

I'm going to move him to the right.

And when this enemy reaches the x-position of 72,

I will then kill that enemy by making its life equal to 0.

So this function, when I call it, will move all 4 of my enemies to the right

until they get to this threshold, in which case,

then they are going to be killed.

We have one more operation to write, and that

is the one that actually renders the image.

This function is going to be called inside our main loop.

Again, this function should be called at 30 Hz inside the main loop.

And to draw all of the sprites onto a single

image,

I will begin by clearing the buffer.

Clearing the screen buffer, this is the array in RAM.

And then, one by one, if the enemy is alive, if the enemy has life,

it'll draw the image associated with that enemy

at the position of where they exist.

And that's the threshold that we've been using all along.

So this one element here will draw the enemy onto the screen.

Then this one will draw that enemy, and then this one

will draw that one, and that one.

So the four enemies will be drawn into the array which is located in RAM.

It's not until this display function that this RAM buffer is then

punched out to the screen.

Help

RAMESH YERRABALLI: So what we are showing

you is just an example of how to manage the sprites, how to declare them,

how to move them, how to draw them.

So this is just an idea that you can base your own game on.

So let's look at how this idea is put together in a simple main program.

JONATHAN VALVANO: OK.

So here we have a main program, let's load it in, let's compile it.

Oops.

Let's compile it, build.

[DING]

Download.

Debug.

[DING]

So let's step over this main program and see what it does.

Step over, step over.

So again, the Nokia Init will initialize the hardware of the display.

The Init function that we just wrote will then place all of my sprites

into the structure.

So I'll have my four enemies that are

created.

RAMESH YERRABALLI: This will only put them in the screen buffer.

JONATHAN VALVANO: No, this doesn't even put them in the screen buffer.

This puts them into the array of enemies.

The next one is going to take the enemy-- let's step into this one,

step into, step over-- here we cleared the buffer, and now one by one,

I'm going to put the enemies into the screen buffer.

Two, three, four.

Now when I execute this one, we see that the enemies are on the screen.

The four enemies, near the top, all on the same row.

All right, keep stepping.

Step over.

And now, the main loop is going to move the enemies.

Now again, when I move the enemies, the screen

doesn't change, because I just moved their properties.

But when I draw it, they are now moved.

RAMESH YERRABALLI: Yes, they have.

JONATHAN VALVANO: All right, let's move them again.

Oh, let's just hit the Go button, because you

can see that this movement will occur 10 times a second.

So you'll actually be able to see them jump on the screen, so let's push Go.

So now you see the enemies move over, and when

they get to the end of the screen, they fall into the abyss,

their life is set to 0, and there are no more enemies now.

RAMESH YERRABALLI: So the interesting thing as we see here

is, the loop is still running, but there's

nothing to draw because we only draw when their life is not a 0.

JONATHAN VALVANO: Absolutely.

All right.

```
struct State {
  unsigned long x;      // x coordinate
  unsigned long y;      // y coordinate
  const unsigned char *image; // ptr->image
  long life;            // 0=dead, 1=alive
};
typedef struct State STyp;
STyp Enemy[4];
void Init(void){ int i;
  for(i=0;i<4;i++){
    Enemy[i].x = 20*i;
    Enemy[i].y = 10;
    Enemy[i].image = SmallEnemy30PointA;
    Enemy[i].life = 1;
   }
}
void Move(void){ int i;
  for(i=0;i<4;i++){
    if(Enemy[i].x < 72){
      Enemy[i].x += 2; // move to right
    }else{
      Enemy[i].life = 0;
    }
  }
}
void Draw(void){ int i;
  Nokia5110_ClearBuffer();
  for(i=0;i<4;i++){
    if(Enemy[i].life > 0){
     Nokia5110_PrintBMP(Enemy[i].x, Enemy[i].y, Enemy[i].image, 0);
    }
  }
  Nokia5110_DisplayBuffer();      // draw buffer
}
int main(void){
  PLL_Init();                   // set system clock to 80 MHz
```

Help

```
  Nokia5110_Init();
  Nokia5110_ClearBuffer();
  Init();
  Draw();
  while(1){
    Move();
    Draw();
    Delay100ms(2);
  }
}
```

Program 15.3. Example use of structures (see the file sprite.c).

Animation is the illusion of motion. In the above section, we showed you to make the sprite move in space, we simply draw it at different places on the screen. The illusion of life occurs as objects change shape as they move. Program 15.4 shows how to extend the structure adding frames. **Frames** are arrays of images that will be drawn in sequence giving the sprite more life-like features. In the starter project, each of the enemies have two BMP images, called A and B. The two images have the same body but their feet and arms are moving. We will create an array of pointers in the structure, where each element of the array points to one frame of the frame sequence. We will set the images to A and B versions. We add a **FrameCount** variable, which goes 0, 1, 0, 1... and use it when we draw the image. In this example, we assumed all the sprites have two images so we used one count-variable for all sprites.

Help

```
struct State {
  unsigned long x;      // x coordinate
  unsigned long y;      // y coordinate
  const unsigned char *image[2]; // two pointers to images
  long life;            // 0=dead, 1=alive
};
typedef struct State SType;
SType Enemy[4];
void Init(void){ int i;
  for(i=0;i<4;i++){
    Enemy[i].x = 20*i;
    Enemy[i].y = 10;
    Enemy[i].image[0] = SmallEnemy30PointA;
    Enemy[i].image[1] = SmallEnemy30PointB;
    Enemy[i].life = 1;
   }
}
void Move(void){ int i;
  for(i=0;i<4;i++){
    if(Enemy[i].x < 72){
      Enemy[i].x += 2; // move to right
    }else{
      Enemy[i].life = 0;
    }
  }
```

```
}
unsigned long FrameCount=0;
void Draw(void){ int i;
  Nokia5110_ClearBuffer();
  for(i=0;i<4;i++){
    if(Enemy[i].life > 0){
     Nokia5110_PrintBMP(Enemy[i].x, Enemy[i].y, Enemy[i].image[FrameCount], 0);
    }
  }
  Nokia5110_DisplayBuffer();      // draw buffer
  FrameCount = (FrameCount+1)&0x01; // 0,1,0,1,...
}
```

Program 15.4. Adding animation to the sprites.

The entire game action is then rendered at 30 Hz. This rate is the slowest we can run and still have our eyes and brain think the image is continuously moving. This means we run the **game engine**, determine new positions for all the sprites, and then draw a new image on the LCD. This entire loop runs 30 times/sec. We will remove the 100ms delay in Program 15.3, and replace it with a semaphore wait. The semaphore is set in an ISR running at 30 Hz, as shown in Program 15.5.


```
int Flag=0;
void SysTick_Handler(void){  // runs at 30 Hz
  Move();  // replace with your game engine
  Flag = 1;
}
int main(void){
  PLL_Init();                  // set system clock to 80 MHz
  Nokia5110_Init();
  Nokia5110_ClearBuffer();
  Init();
  Draw();
  while(1){
    while(Flag==0){};
    Draw(); // update the LCD
    Flag = 0;
  }
}
```

Program 15.5. Run the game engine in the ISR and draw to LCD in the main.

CHECKPOINT 15.7

For good design we must appropriately place software tasks in either an ISR or in the main. A measure of good design is to have the time to execute an ISR short compared to the time between when the ISR is invoked. In a previous checkpoint

we saw it takes about 1.22ms to draw an image, and we appropriately placed the LCD draw in the main of Program 15.5. Count the number of lines of C that will get executed in the ISR of Program 15.5 and multiply that count by 0.1us. Using this crude estimate, how long will it take to execute the ISR? Is this good design?

Hide Answer

SysTick occurs at 30 Hz, or every 33.33ms. There are about 27 lines (I counted each line in the for-loop four times), meaning it will take about 3us to execute. 3us is very small compared to 33.33ms. This is good. Your game engine may take up to 100us to run, and that will be OK.