An introduction to Mophun

Simon Kågström

ska@bth.se

TEK

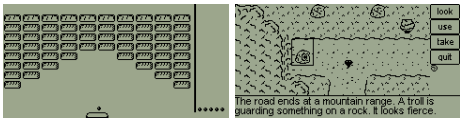
A gentle introduction...

- An example from my own back yard
- The *REX 6000*, the worlds smallest PDA
  - In terms of *size* (around 40 grammes, PCMCIA form factor)
  - In terms of CPU speed (8-bit z80, 4.77 MHz)
  - In terms of memory (8KB code+initialized data size, 4KB stack)
- Programming for it is done using home-made tools
  - *ZCC*, buggy C-compiler (produces inefficient code)
  - Reverse-engineered API (Xircom/Intel never released an official one)

So programming for it should be much fun!

REX 6000, cont.

- You sometimes have to be careful when coding for the REX
  - Frequently run code can easily become a speed constraint
  - Large images: your memory will run out in no-time
  - Using the correct data types is *very* important
  - (Buggy compiler)
- Still, it is possible:



What should we learn today?

- How to build an run a simple Mophun app
  - GCC (plus options)
  - Compiling and linking + special about Makefiles
- An intro to the Mophun API
  - Sprites
  - Tiles and tile maps
  - Input
  - Sound (some)
- Plus examples on how to use the API

What is the Mophun API?

- Platform-independent API for gaming
- Provides graphics, input, sound and communication interfaces
  - Sprite engine
  - Map (tile) engine
  - 3D engine (with newer versions)
  - MIDI and samples for sound
  - Bluetooth, IR and TCP/IP communication
- The API is implemented natively on each device ...
  - (in different versions)
- ... but your application code is run by a virtual machine
  - (Do you get visions of *java-being-slow-on-a-P4*?)

Useful resources

- !!!The Mophun API reference!!!
  - This is available as a help-file under windows
  - You can also browse it with a web-browser
  - **Very important**: Keep this under your pillow at all times!
- Mophun programming guidelines
  - A document on how to start with Mophun
  - Read this before the Pong lab!
- Mophun assembly reference
  - Reference for PIP-assembly programmers
  - Read this if you are interested in how the virtual machine works
- Emulator help etc.

A first, trivial, mophun application

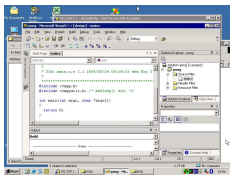
```
#include <vmgp.h> /* Most Mophun stuff */
#include <vmgputil.h> /* msSleep() etc */

int main(int argc, char **argv)
{
  vClearScreen(vRGB(255,0,0));
  vFlipScreen(1);

  while(!vGetButtonData());

  return 1;
}
```

- This application clears the screen to red and waits for a keypress.

- Usually both header-files are included
- The functions used here are outlined later
- **Bottom-line**: Small amounts of code needed for simple things

Using VC++ with Mophun

- You can use VC++ for mophun
  - This is not the only option, the standard UNIX environment works as well
- Mophun programming guidelines have more on VC++
  - But beware: It's not complete!
- And now ... some demonstration!



Using VC++ with Mophun, steps

- **File-¿New Project**
- Select **Makefile project**, enter name
- Select **application settings**
  - Output `NAME.mpn` (was `.exe`)
  - Build command line: `nmake -f makefile`
  - Clean command line: `nmake -f makefile clean`
  - Rebuild command line: `nmake -f makefile rebuild`
- Copy the files into the project directory (if you have them beforehand)
- Right-click **Source Files** in the **Solution** side-bar, **Add-¿Add Existing Item**

Building Mophun applications

- Building mophun applications consist of 4 steps:
  1. Compiling "resources" (explained later)
     `morc res.txt`
  2. Compiling your source code
     `pip-gcc -Wall -O2 -c main.c -o main.o`
  3. Linking the object-files together (`GCC` or `ld`)
     `pip-gcc -mstack=512 -mdata=16384 main.o res.o -o my_game.mpn`
  4. For downloading to your phone: Certifying your game
     - This is done by Synergenix (or through **mocert.student.bth.se** here)
- We'll talk about Makefiles at the end of the lecture

Compiling source code

- Compiling source files is a standard GCC-matter (using `pip-gcc`)
- Useful `pip-gcc`-options
  - `-O2`: Optimize code with level 2
  - `-g`: Produce debugging information (for use with GDB, don't use -O2 with this option)
  - `-Wall`: Present all warnings (use this and *follow its recommendations*!)
  - `-c`: Compile only, do not link
  - `-fvstudio`: Produce error messages that visual studio can parse

```
pip-gcc -Wall -O2 -c main.c -o main.o
```


Linking object files together

- When linking you have to supply some extra info
- `-mstack=512`: Use 512 bytes of stack space
  - Stacks? Refer to the C-lectures!
  - Should be large enough to handle your largest local variable + function nesting

```
int fn(int n) {
     uint8_t buf[128];
     fn(n+1);
}
```

- `-mdata=16384`: Use 16384 (16KB) of heap space.
- This should be large enough to handle all your `vNewPtr`-data (`malloc`)

Resources

- Resource-files specify data used by the game
  - Sprites, tiles, fonts etc.
- The resource-file is a formatted text-file
- From a resource-file (`res.txt`), two files are created:
  - `res.h`: A C header-file with definitions for the resources
  - `res.o`: A object-file with the resource data
- How to use the resources depend on the type (more later)
- To compile resources, the `morc` executable is used
- Templates are available from the course homepage (**http://idenet.bth.se**)

Resources II

- Complete example from the Pong lab

res.txt

```
INFO METAINFO
{
  "Title"              : "Pong"
  "Vendor"             : "Simon Kågström"
  "Copyright info"     : "(c) Simon Kågström 2004"
  "Program version"    : "1.0"
  "Help"               : "Help me!"
}

SECTION DATA
// Tiles and sprites
BG_TILES      TILESET 8 8 FORMAT RGB332 "gfx/tiles.bmp"
BALL_SPRITE   SPRITE      FORMAT RGB332 "gfx/ball.bmp"
PADDLE_SPRITE SPRITE      FORMAT RGB332 "gfx/paddle.bmp"
```

res.h

```
[...]
extern unsigned char BG_TILES[384];
#define BG_TILES_COUNT          6
#define BG_TILES_TILESIZE       64
[...]
extern SPRITE BALL_SPRITE;
#define BALL_SPRITE_WIDTH       8
#define BALL_SPRITE_HEIGHT      8
#define BALL_SPRITE_CENTERX     0
[...]
```


Input handling

- Input handling in mophun is fairly simple
  - *But*: there are device-specific quirks and bugs to look out for!
- Mophun *always* supports a smallest subset of input facilities
  - 4 **direction keys**, **fire** and **select**/**back**
  - These are easily mapped to the numeric keypad and the joysticks on most phones
- You generally want a bitmask of pressed keys
  - `uint32_t vGetButtonData()`
  - `vGetButtonData() & KEY_FIRE`: test if **fire**/5-key pressed

Input handling II

- There is also API functionality for mouse/touchpad-like pointer operation
  - We won't use it here though
- A simple example can be seen below
  - Wait until the fire (5) key is pressed

```
void wait_keys(void)
{
  uint32_t keys;

  do {
    keys = vGetButtonData();
    if (keys & KEY_UP)
      ; /* Do something on up */
    if (keys & KEY_DOWN)
      ; /* Do something else ... */
  } while ( (keys & KEY_FIRE) == 0);
}
```

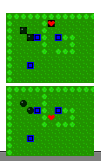

Graphics, introduction

- What do we need for a 2D game really?
  - Sprites
  - A background
  - Big background image
  - Tile-operations
- Double buffering
  - Two buffers: one for *drawing* and one for *displaying*
  - Drawed items are not shown on the screen until the buffers are "flipped"
  - `vFlipScreen(int32_t block)`: Flip the buffers
  - `block` specifies wait for vertical retrace (i.e. until the screen is completely drawn)

Graphics, initialization

- `vSetTransferMode(mode)`: How to copy bitmaps, `MODE_TRANS` (transparent) or `MODE_BLOCK`
  - More in API $\geq$ 1.50
- `vClearScreen(color)`: Clear the screen to `color` (use `vRGB(r,g,b)`)
- `vSetClipWindow(x1,y1,x2,y2)`
- Get the screen size (capabilities)

```
static void get_screen_size(int32_t *p_w, int32_t *p_h) {
  VIDEOCAPS videocaps;

  videocaps.size = sizeof(VIDEOCAPS);
  if(vGetCaps(CAPS_VIDEO, &videocaps)) {
      *p_w = videocaps.width;
      *p_h = videocaps.height;
    }
  else
    vTerminateVMGP(); /* Oh no! */
}
```



Graphics, sprites I

- What is a sprite?
  - We all know that, of course: *"small, human in form, playful, having magical powers"* (WordNet 2.0 August 2003). Aha...

- For computer games it generally denotes a moving object on the screen
- Mophun provides a sprite framework with collision handling etc.
- Probably the part of Mophun you will use the most!
  - (So listen carefully...)

Graphics, sprites II

- Sprites in Mophun are arranged in *sprite slots*
- Slots are used for collisions and as a list for drawing
- You initialize the number of slots with `vSpriteInit(n_sprites)`
- However: the bitmap is *not* tied to the slot
  - Animations (player walking etc.) are handled separately, see **idenet**
- `vSpriteSet()` is used to place sprites and set the bitmaps

```
vSpriteSet(5, &SPRITE_UP, 10, 10);
vSpriteSet(p_player->sprite_slot, p_player->p_sprite,
           p_player->x, p_player->y);
```

- The above example places sprite number 5 on (10,10) and the the player on its x and y position.

Graphics, sprites III

- Call `vUpdateSprite()` to draw sprites to the *back buffer*
- This function draws all sprite slots visible on the screen
  - It is safe to place a sprite outside the screen
- To draw the sprites on the *screen* you must call `vFlipScreen(1)` as well
- `vUpdateSpriteMap()` is used to update *both* the sprites and the tile map (more later)

Graphics, sprites IV

```
#include <vmgp.h>
#include "res.h" /* SPRITE_BITMAP */

int main(int argc, char *argv[])
{
  int16_t y=0;
  int16_t x=0;

  vClearScreen(vRGB(0,0,0));
  vSpriteInit(1);
  while (1) {
    x = (x + 1) & 127;
    /* Place the sprite */
    vSpriteSet(0, &SPRITE_BITMAP,
               x, y);
    /* Update sprites */
    vUpdateSprite();
    /* Draw everything */
    vFlipScreen(1);
  }
}
```

- A complete example!
- We assume that you have a `res.txt` resource file
- This moves a sprite across the screen from left to right
- No way to exit :-)

Graphics, sprites V: collisions

- There are some predefined functions in Mophun for collisions
- `vSpriteCollision(uint8_t slot, uint8_t slotfrom, uint8_t slotto)`: Check for collisions between `slot` and `[slotfrom ... slotto]`.
- The function returns the index of the first collision
- Multiple collisions? We need a loop then...
- Example below from the Pong game (paddle and ball)

```
/* Bounce against paddle? */
if ( vSpriteCollision(p_paddle->sprite_slot,
                      p_ball->sprite_slot, p_ball->sprite_slot) >= 0)
  {
    /* A collision! handle that */
  }
```


Graphics, tiles

- *Tilemaps* are useful in many 2D-based games
- Tiles in mophun are 8 by 8*`n_tiles` BMP images.
- Tiles are used in tilemaps
- A tilemap defines a `NxM` "map" of tiles
- A tilemap is a one-dimensional array of 8-bit values
  - Every value in the array specifies a tile-number (starting at 1, tile 0 is empty)
  - Tilemaps can have attributes
  - (Then) every other value is an attribute value (i.e. every array element takes 16 bits)
  - Attributes: Transparency, animation etc.

Graphics, tile initialization

- Tilemaps must be initialized with `vMapInit(MAP_HEADER*)`
- An example is given below

The map data

```
/* Beware: In version < 1.30,
 * the tilemap size must be
 * as large as the screen area.
 */
#define LEVEL_W 8
#define LEVEL_H 7
static uint8_t level_map[] =
  {
    /* No attributes */
    2, 2, 2, 2, 2, 2, 2, 2,
    2, 1, 2, 1, 1, 1, 1, 2,
    2, 1, 2, 1, 1, 1, 1, 2,
    2, 3, 2, 1, 3, 2, 2, 2,
    2, 1, 2, 1, 1, 1, 2, 2,
    2, 1, 2, 1, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 2, 2, 2,
  };
```

Initialization code

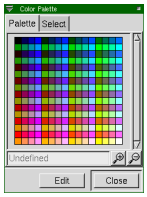
```
static void init_level(MAP_HEADER *p_bgmap)
{
  memset(p_bgmap, 0, sizeof(MAP_HEADER));

  /* Initialise the background map */
  p_bgmap->width = LEVEL_W;
  p_bgmap->height = LEVEL_H;
  p_bgmap->mapoffset = level_map;
  p_bgmap->tiledata = BG_TILES;      /* from res.h */
  p_bgmap->format = BG_TILES_FORMAT; /* ditto */

  /* Setup the background map */
  if ( !vMapInit(p_bgmap) )
    vTerminateVMGP(); /* Failed, terminate! */
}
```


Graphics, tiles, checking against

- You usually need to check the tilemap for the tile at certain coordinates
  - Sokoban game: if a ball is on a hole
  - Pong, lab 2: if the ball bounces against a wall
  - Racing game: what material are we driving on?
- This is simple to do manually (`tilemap[y*MAP_W+x]`, in tilemap coordinates).
- Mophun can also do this for you (`vMapGetTile(x,y)`, also in tilemap coordinates).
- Where should the checking be done? Depends:
  - Racing game: on contact with the material
  - Pong, lab 2: if the ball moves towards the wall

Graphics, misc.

- There are many more functions in the mophun API
- Lookup these in the API documentation
- A note about the palette support:
  - Mophun uses RGB332, i.e. a 255-entry palette with 3 bits for red and green and two for blue
  - Sounds primitive, but is useable.



Sound

- The mophun sound capabilities differs very much between platforms
- Older phones (T610 etc.) have only very basic sound options
  - `vBeep(5000,50)`: Play a beep (think PC-speaker) at 5000 Hz for 50 ms
  - `vPlayResource(BOUNCE, BOUNCE_SIZE, SOUND_TYPE_AMR)`: Play the *AMR* sample bounce
  - See the documentation for more information about these
- Newer phones (API $\geq$ 1.50) have a much more advanced API
  - Since rather few of you have such phones, we'll skip it!

Capabilities

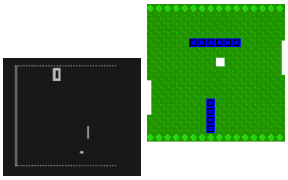
- Different phones support different parts of Mophun
- Capabilities is a way of checking this
- We have already seen the screen-size capabilities, there are many other:
  - Communication
  - Input
  - Sound
  - System (vendor etc.)
- Consult the Mophun documentation for more about this subject

Complete example

```
#include <vmgp.h>
#include <vmgputil.h> /* msSleep */
#include "res.h"

#define LEVEL_W 8
#define LEVEL_H 7
static uint8_t level_map[] = {
  /* No attributes */
  2, 2, 2, 2, 2, 2, 2, 2,
  2, 1, 2, 1, 1, 1, 1, 2,
  2, 1, 2, 1, 1, 1, 1, 2,
  2, 3, 2, 1, 3, 2, 2, 2,
  2, 1, 2, 1, 1, 1, 2, 2,
  2, 1, 1, 1, 2, 2, 2, 2,
  2, 2, 2, 2, 2, 2, 2, 2,
};

int main(int argc, char **argv) {
  MAP_HEADER bgmap;
  int y = 16; /* Ugly... */

  vClearScreen(vRGB(255,0,0));
  vSetClipWindow(0,0,128,206);
  vSpriteInit(1); /* One sprite */
  /* Zero the background map */
  memset(&bgmap, 0, sizeof(MAP_HEADER));
  bgmap.width = LEVEL_W;
  bgmap.height = LEVEL_H;
  bgmap.mapoffset = level_map;
  bgmap.tiledata = BG_TILES;      /* from res.h */
  bgmap.format = BG_TILES_FORMAT; /* ditto */
  vMapInit(&bgmap); /* Use our background map */
  while(1) {
      uint32_t keys = vGetButtonData();

      if (keys & KEY_DOWN)
        y = (y + 1) & 31; /* Keep the sprite 0..31 */
      if (keys & KEY_UP)
        y = (y - 1) & 31;

      vSpriteSet(0, &BALL_SPRITE, 5, y);
      vUpdateSpriteMap();
      msSleep(50);
      vFlipScreen(1);
    }

  vMapDispose();
  vSpriteDispose();
  return 0;
}
```

The Pong labs

- You can download some examples from idenet
  - *sprite animation*, *a complete sokoban game*, *font handling* and a *state machine* example.
- The Pong lab specs contain a description of useful functions for the labs
- Start implementing them on time!!



Makefiles

- *Make* is a tool used for deciding which files to rebuild in a project (used in your projects)
- Used since the beginning of time in UNIX, *extremely* powerful tools
  - You, however, will use the simpler **nmake**
- A Makefile uses *rules* to decide if rebuilding is needed

```
.c.o:
     pip-gcc -c $(CFLAGS) -o $@ $<
```

- This rule states that object-files (`*.o`) depends on C source-files (`*.c`).
- **Example**: `main.o` depends on `main.c`

Makefiles II

- Environment variables can be used in Makefiles:

```
CFLAGS = -Wall -O2    # Set $(CFLAGS)
```

- *Targets* specify files that can be generated by the Makefile (also a rule):

```
OBJS = res.o main.o
my_game.mpn: $(OBJS)
     pip-gcc -mstack=512 -mdata=16384 $(OBJS) -o $@

all: my_game.mpn
```

- The *all*-target is the default target
- Another common target is *clean*

```
rm = del
clean:
     $(rm) *.o *~
```


Makefiles III

- Complete example

```
# Settings
rm = del

CFLAGS = -Wall -g -fvstudio    # Set $(CFLAGS)
OBJS = res.o main.o

# Rules
.c.o:
     pip-gcc -c $(CFLAGS) -o $@ $<

res.o: res.txt
     morc res.txt -o $@

# Targets
my_game.mpn: $(OBJS)
     pip-gcc -mstack=512 -mdata=16384 $(OBJS) -o $@

clean:
     $(rm) *.o *~

all: my_game.mpn
```

- See **http://idenet.bth.se** for template Makefiles

What did we learn today?

- Basic information about the Mophun API
- How to use sprites, tilemaps, etc.
- Some things about resources
- Also how to compile and link mophun apps (introduction to Makefiles)
- What have we *not* learned:
  - Communication API (rumor says: buggy)
  - Sound API (large differences between versions)
  - Text output (see example on **idenet**)
  - plus lots and lots of details

Debugging mophun applications

- Debugging Mophun applications is done with GDB
- PIP-GDB and a graphical frontend (Insight) is installed on the students' computers
- There will be a separate document describing how to work with it
- Unfortunately, VC++ cannot run GDB integrated
  - (GNU/Emacs can of course, also for Mophun)
- Still, you will *need* to use the debugger

Skool Daze!

- C64, 198?)
- Excellent idea, great gameplay and *technically very simple*