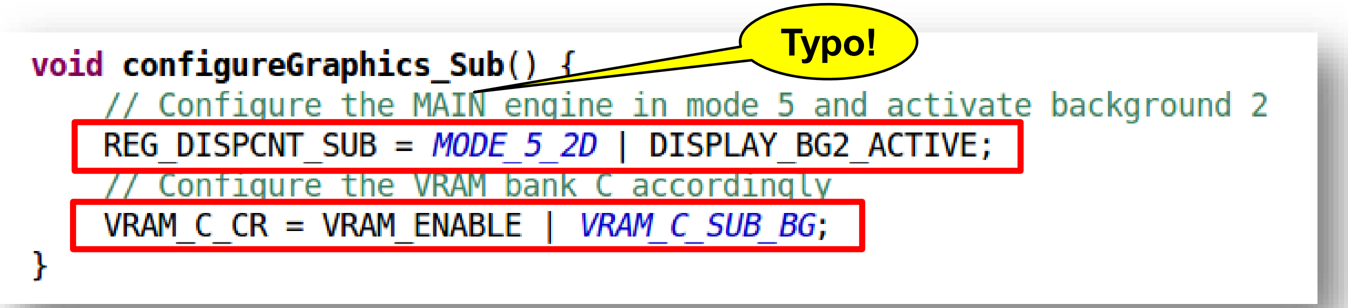
Mid-Term Exam 2014
Code review

Prof. David Atienza Alonso

Systèmes Embarqués Microprogrammés

Exercise 1: Configure SUB engine

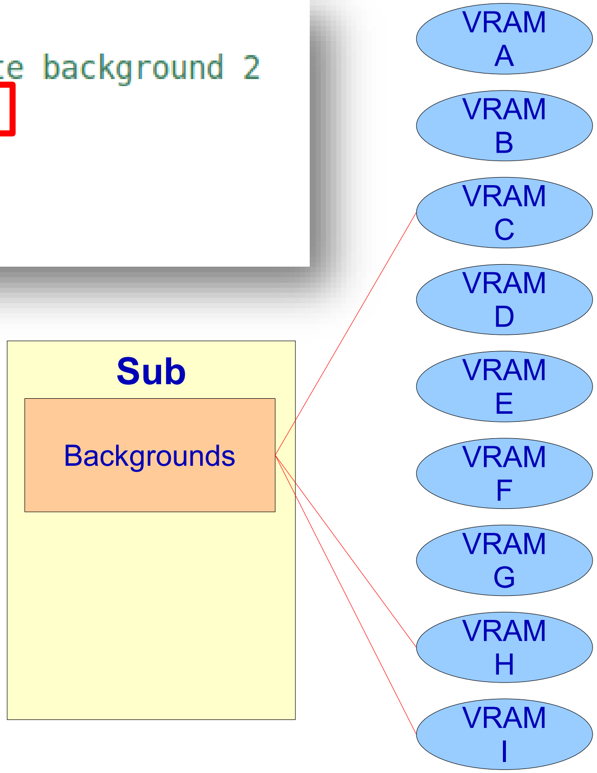
- Complete the function **configureGraphics_Sub()** following the given comments to configure the SUB engine in mode 5 and activate the background BG2.
  1. Configure engine in Mode 5 activating background 2
  2. Enable and configure VRAM bank accordingly



```
void configureGraphics_Sub() {
    // Configure the MAIN engine in mode 5 and activate background 2
    REG_DISPCNT_SUB = MODE_5_2D | DISPLAY_BG2_ACTIVE;
    // Configure the VRAM bank C accordingly
    VRAM_C_CR = VRAM_ENABLE | VRAM_C_SUB_BG;
}
```

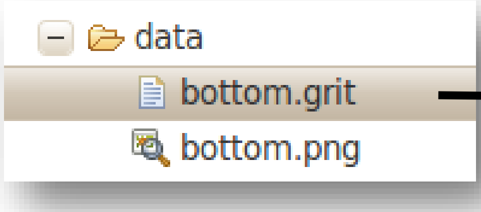
Typo!

Mode	BG0	BG1	BG2	BG3
0	Tiled	Tiled	Tiled	Tiled
1	Tiled	Tiled	Tiled	Rotoscale
2	Tiled	Tiled	Rotoscale	Rotoscale
3	Tiled	Tiled	Tiled	Ext. Rotoscale
4	Tiled	Tiled	Rotoscale	Ext. Rotoscale
5	Tiled	Tiled	Ext. Rotoscale	Ext. Rotoscale



Exercise 1: Transform image with grit

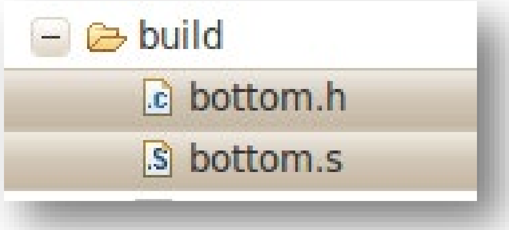
- Download the image **bottom.png** into the "data" folder of the project and create the configuration grit file in order to obtain the bitmap and the corresponding palette (therefore using pixels of 8-bit length)
  1. Copy "bottom.png" to the data folder and create grit file



data
bottom.grit
bottom.png

```
-g
-gb
-gB8
-p
```

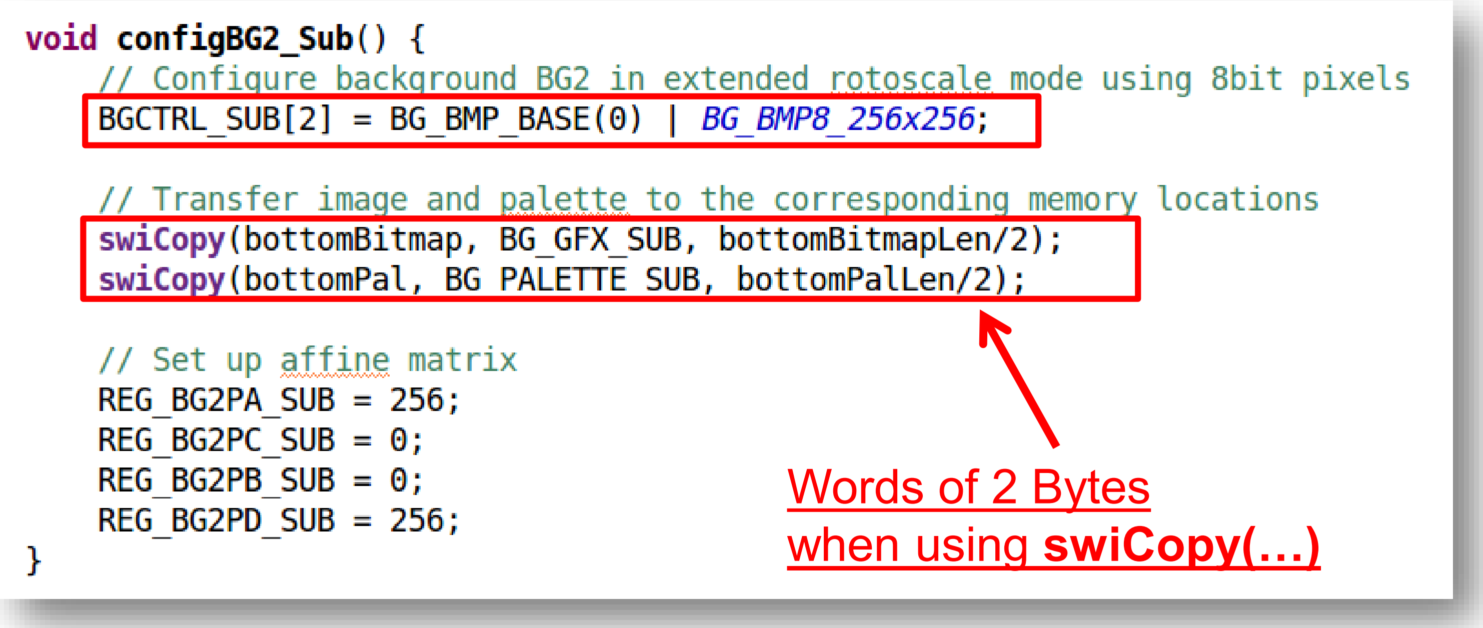
  2. The compilation tool-chain **will generate automatically** the necessary files



build
bottom.h
bottom.s

Exercise 1: Transfer image

- Complete the function **configBG2_Sub()** following the given comments to configure the background correctly and transfer the image information to the corresponding locations in memory.
  1. Configure background in rotoscale mode using the palette (8-bit pixels)
  2. Transfer bitmap and palette to the graphical memory

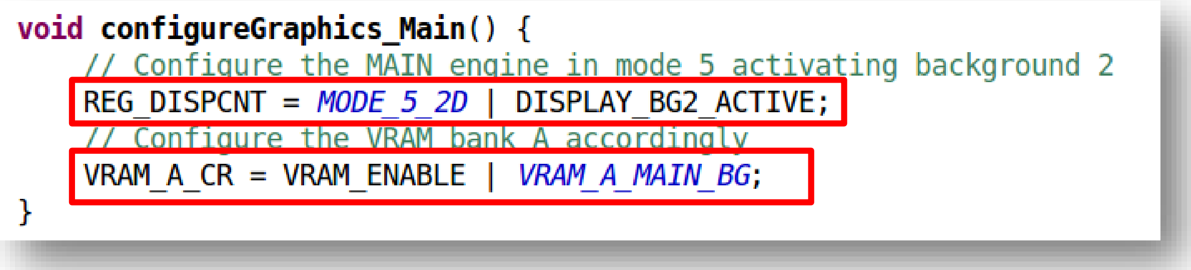


```
void configBG2_Sub() {
    // Configure background BG2 in extended rotoscale mode using 8bit pixels
    BGCTRL_SUB[2] = BG_BMP_BASE(0) | BG_BMP8_256x256;

    // Transfer image and palette to the corresponding memory locations
    swiCopy(bottomBitmap, BG_GFX_SUB, bottomBitmapLen/2);
    swiCopy(bottomPal, BG_PALETTE_SUB, bottomPalLen/2);

    // Set up affine matrix
    REG_BG2PA_SUB = 256;
    REG_BG2PC_SUB = 0;
    REG_BG2PB_SUB = 0;
    REG_BG2PD_SUB = 256;
}
```

Words of 2 Bytes
when using **swiCopy(…)**

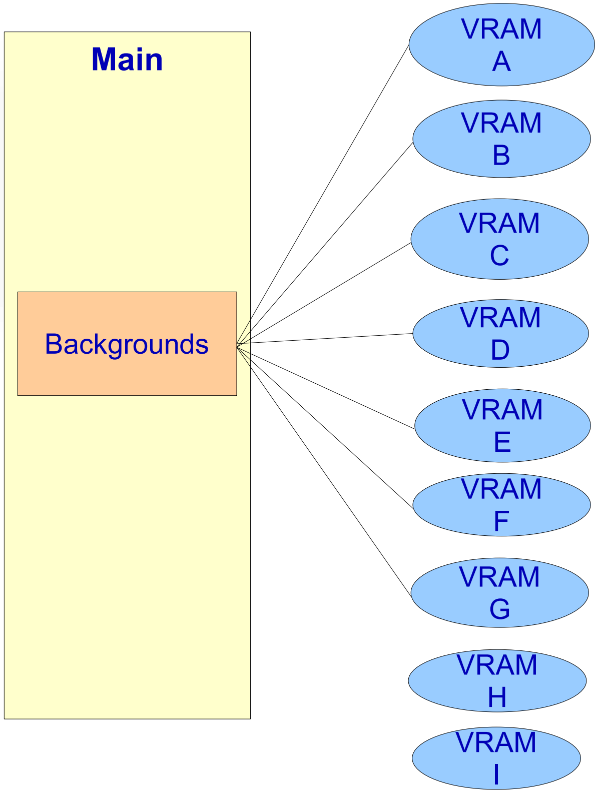
Exercise 2: Configure MAIN Engine

- Complete the function **configureGraphics_Main()** following the given comments in the source code to configure the MAIN engine **in mode 5** activating **background 2**.
  1. Configure Engine in mode 5 activating background 2.
  2. Enable and configure VRAM memory accordingly



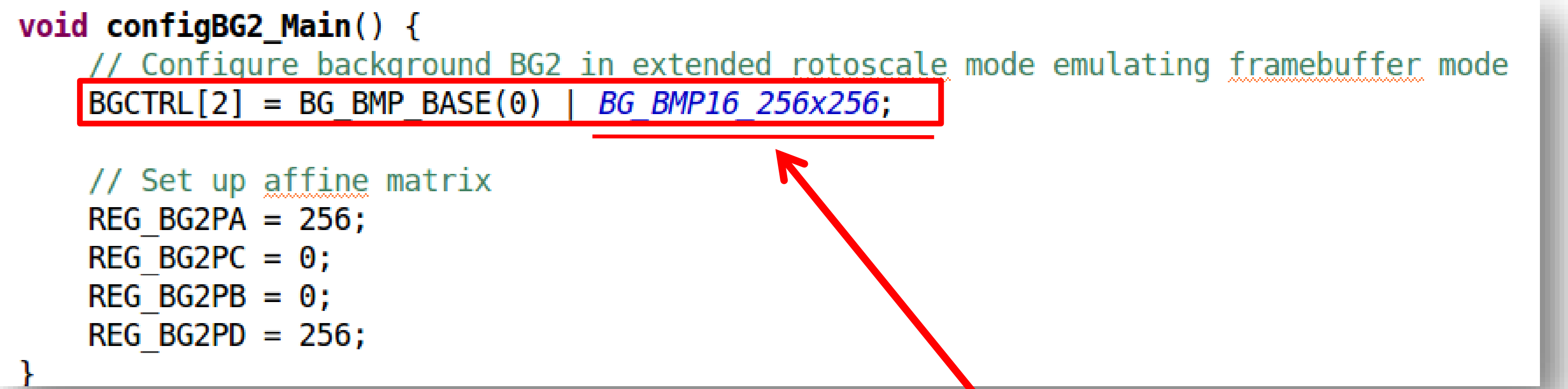
```
void configureGraphics_Main() {
    // Configure the MAIN engine in mode 5 activating background 2
    REG_DISPCNT = MODE_5_2D | DISPLAY_BG2_ACTIVE;
    // Configure the VRAM bank A accordingly
    VRAM_A_CR = VRAM_ENABLE | VRAM_A_MAIN_BG;
}
```

Mode	BG0	BG1	BG2	BG3
0	Tiled/3D	Tiled	Tiled	Tiled
1	Tiled/3D	Tiled	Tiled	Rotoscale
2	Tiled/3D	Tiled	Rotoscale	Rotoscale
3	Tiled/3D	Tiled	Tiled	Ext. Rotoscale
4	Tiled/3D	Tiled	Rotoscale	Ext. Rotoscale
5	Tiled/3D	Tiled	Ext. Rotoscale	Ext. Rotoscale
6	3D	N/A	Large Bitmap	N/A
FrameBuf.	Direct VRAM display as a bitmap			



Exercise 2: Configure Background

- Complete the function **configBG2_Main()** following the given comments in the source code to configure the **background 2** in **extended rotoscale mode** emulating **framebuffer mode**.



```
void configBG2_Main() {
    // Configure background BG2 in extended rotoscale mode emulating framebuffer mode
    BGCTRL[2] = BG_BMP_BASE(0) | BG_BMP16_256x256;

    // Set up affine matrix
    REG_BG2PA = 256;
    REG_BG2PC = 0;
    REG_BG2PB = 0;
    REG_BG2PD = 256;
}
```

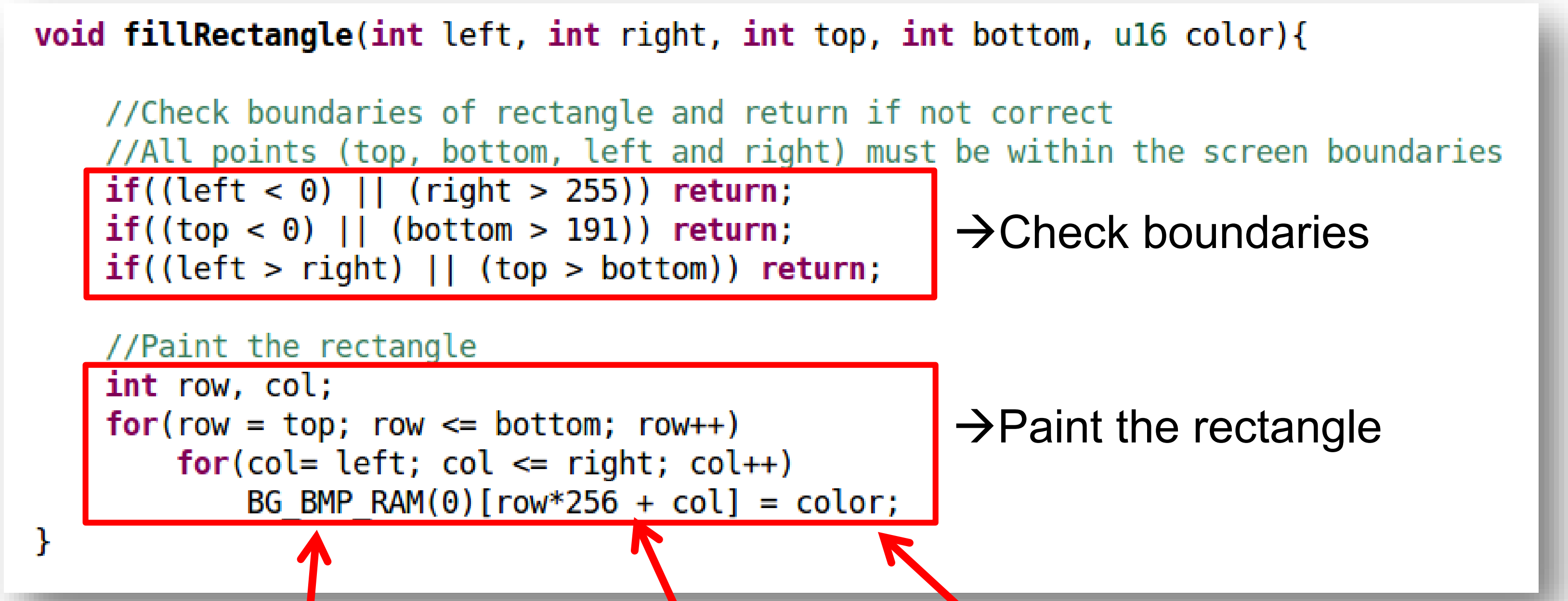
Mode	BG0	BG1	BG2	BG3
0	Tiled/3D	Tiled	Tiled	Tiled
1	Tiled/3D	Tiled	Tiled	Rotoscale
2	Tiled/3D	Tiled	Rotoscale	Rotoscale
3	Tiled/3D	Tiled	Tiled	Ext. Rotoscale
4	Tiled/3D	Tiled	Rotoscale	Ext. Rotoscale
5	Tiled/3D	Tiled	Ext. Rotoscale	Ext. Rotoscale
6	3D	N/A	Large Bitmap	N/A
FrameBuf.	Direct VRAM display as a bitmap			

2 possible pixel depths in rotoscale mode

- 8-bits pixels → Using palette
- **16-bits pixels → <u>Emulating Framebuffer</u>**

Exercise 2: Fill Rectangle

- Complete the function **fillRegion_Main(…)** following the given comments in order to fill one of the four regions with the color given as parameter.



```
void fillRectangle(int left, int right, int top, int bottom, u16 color){

    //Check boundaries of rectangle and return if not correct
    //All points (top, bottom, left and right) must be within the screen boundaries
    if((left < 0) || (right > 255)) return;
    if((top < 0) || (bottom > 191)) return;
    if((left > right) || (top > bottom)) return;

    //Paint the rectangle
    int row, col;
    for(row = top; row <= bottom; row++)
        for(col= left; col <= right; col++)
            BG_BMP_RAM(0)[row*256 + col] = color;
}
```

→Check boundaries

→Paint the rectangle

Pointer to buffer
(256x192 matrix)

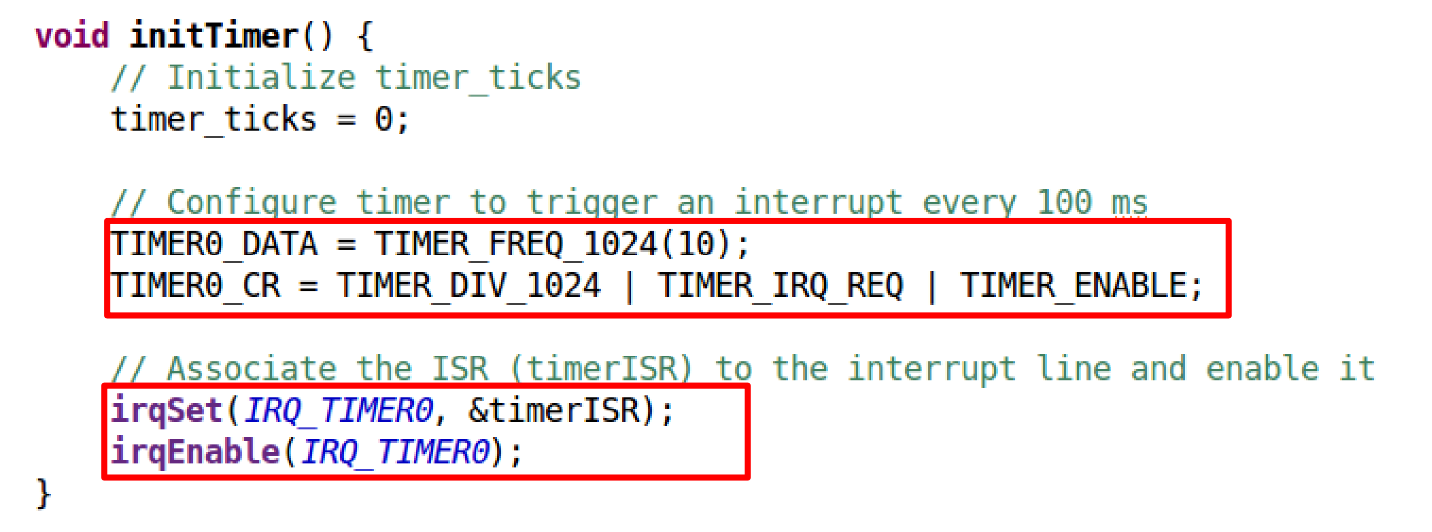
Matrix component

Color given as input parameter

- Set in **configBG2_Main(…)**
- BG_GFX is also valid
- VRAM_A only valid in framebuffer mode!

Exercise 3: Configure Timer

- Complete the function **configureTimer()** following the given comments in order to configure a timer to trigger an interrupt every 100 ms.
  1. Configure the timer to trigger an interrupt at 10 Hz (10 times per second)
  2. Associate the ISR to the interrupt line and enable the interrupt line

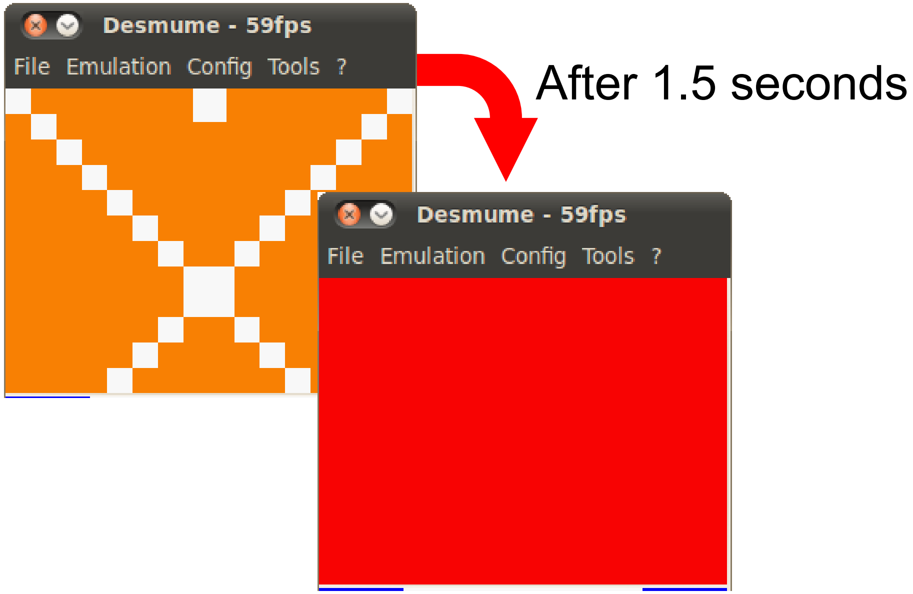


```
void initTimer() {
    // Initialize timer_ticks
    timer_ticks = 0;

    // Configure timer to trigger an interrupt every 100 ms
    TIMER0_DATA = TIMER_FREQ_1024(10);
    TIMER0_CR = TIMER_DIV_1024 | TIMER_IRQ_REQ | TIMER_ENABLE;

    // Associate the ISR (timerISR) to the interrupt line and enable it
    irqSet(IRQ_TIMER0, &timerISR);
    irqEnable(IRQ_TIMER0);
}
```

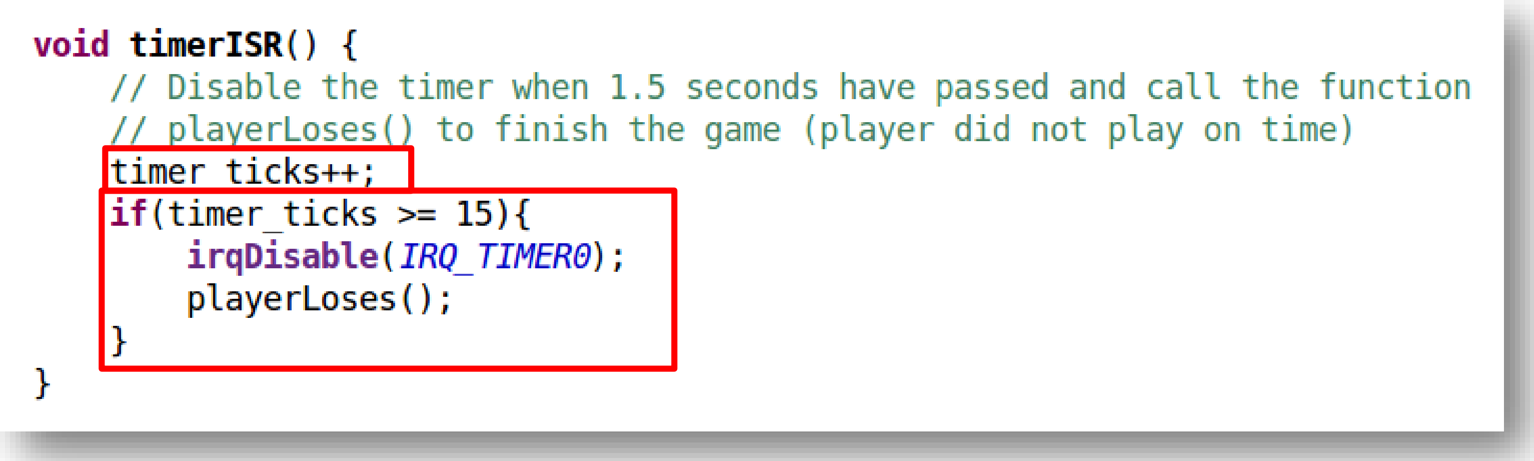
As specified in the text, **irqInit()** must NOT be called!

Exercise 3: Implement Interrupt Service Routine (ISR)

- Complete **timerISR()** so that after 1.5 seconds it <u>disables the timer interrupt</u> and calls the function **playerLoses()**
  1. Increment timer_ticks
  2. After 15 timer_ticks (1.5 seconds), <u>disable the timer interrupt</u> and call the function **playerLoses()**



After 1.5 seconds

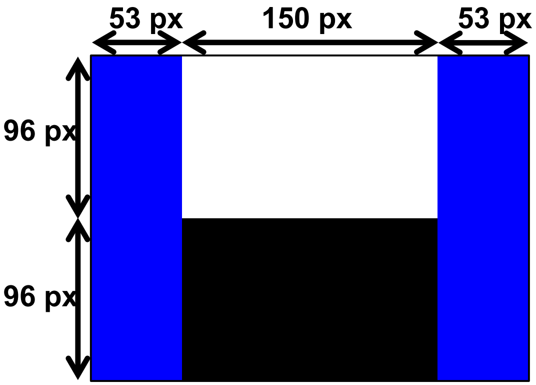


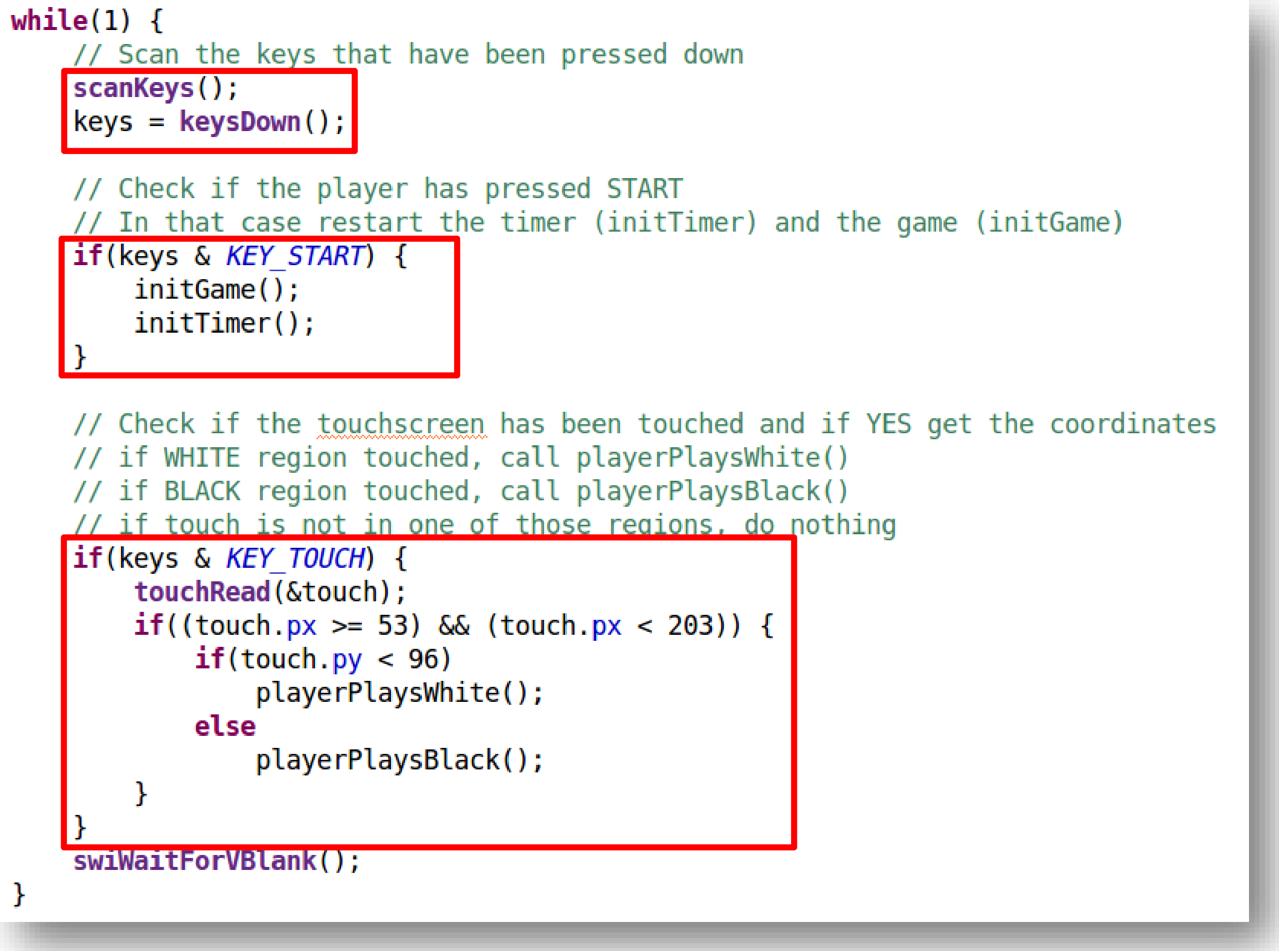
```
void timerISR() {
    // Disable the timer when 1.5 seconds have passed and call the function
    // playerLoses() to finish the game (player did not play on time)
    timer_ticks++;
    if(timer_ticks >= 15){
        irqDisable(IRQ_TIMER0);
        playerLoses();
    }
}
```

Exercise 4: Touchscreen management

- Complete the function **exercise_4()** to read the keys and the touchscreen following the given comments in the source code.

1. Scan the keys that have gone down (from not pressed to pressed)
2. Restart the game if the START key has been pressed
3. Read the touchscreen if it has been touched and do the corresponding action



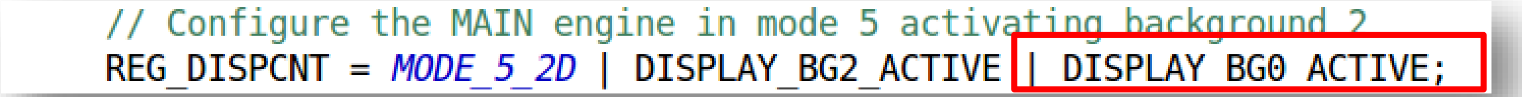


```
while(1) {
    // Scan the keys that have been pressed down
    scanKeys();
    keys = keysDown();

    // Check if the player has pressed START
    // In that case restart the timer (initTimer) and the game (initGame)
    if(keys & KEY_START) {
        initGame();
        initTimer();
    }

    // Check if the touchscreen has been touched and if YES get the coordinates
    // if WHITE region touched, call playerPlaysWhite()
    // if BLACK region touched, call playerPlaysBlack()
    // if touch is not in one of those regions, do nothing
    if(keys & KEY_TOUCH) {
        touchRead(&touch);
        if((touch.px >= 53) && (touch.px < 203)) {
            if(touch.py < 96)
                playerPlaysWhite();
            else
                playerPlaysBlack();
        }
    }
    swiWaitForVBlank();
}
```

©ESL/EPFL

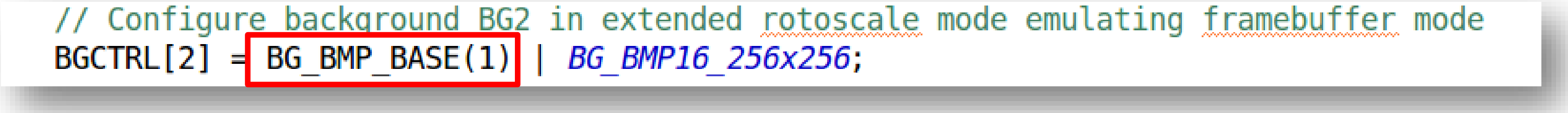
Exercise 5: Activate Background 0 and Change Configuration of Background 2

- Modify the function **configureGraphics_Main()** such that the MAIN engine is configured to use two backgrounds (BG0 and BG2).



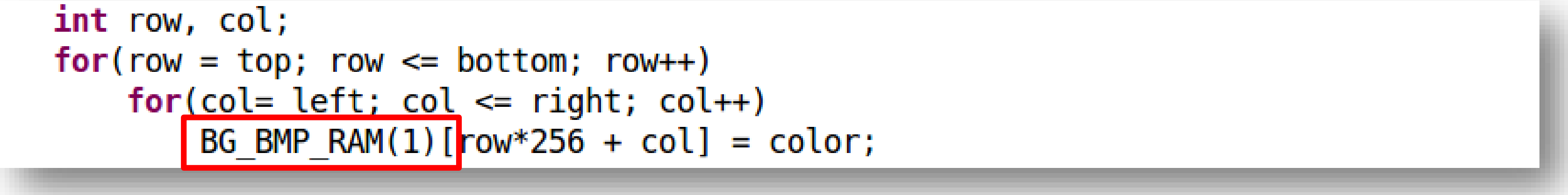
```
    // Configure the MAIN engine in mode 5 activating background 2
    REG_DISPCNT = MODE_5_2D | DISPLAY_BG2_ACTIVE | DISPLAY_BG0_ACTIVE;
```

- Change Background 2 configuration in function **configBG2_Main()** to work in BG_BMP_BASE(1) as specified in the exam sheet.



```
  // Configure background BG2 in extended rotoscale mode emulating framebuffer mode
  BGCTRL[2] = BG_BMP_BASE(1) | BG_BMP16_256x256;
```

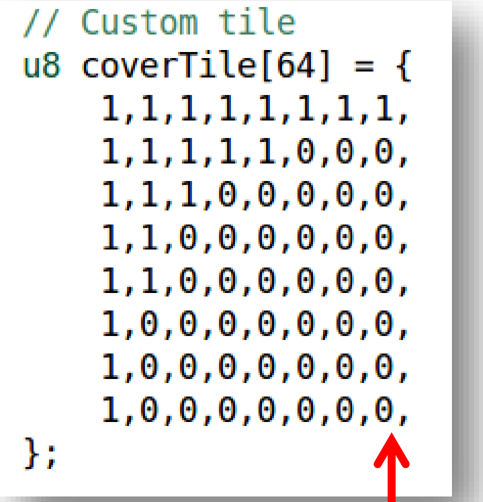
- Change the function **fillRectangle(…)** using the corresponding macro as specified in the exam sheet.



```
    int row, col;
    for(row = top; row <= bottom; row++)
        for(col= left; col <= right; col++)
            BG_BMP_RAM(1)[row*256 + col] = color;
```

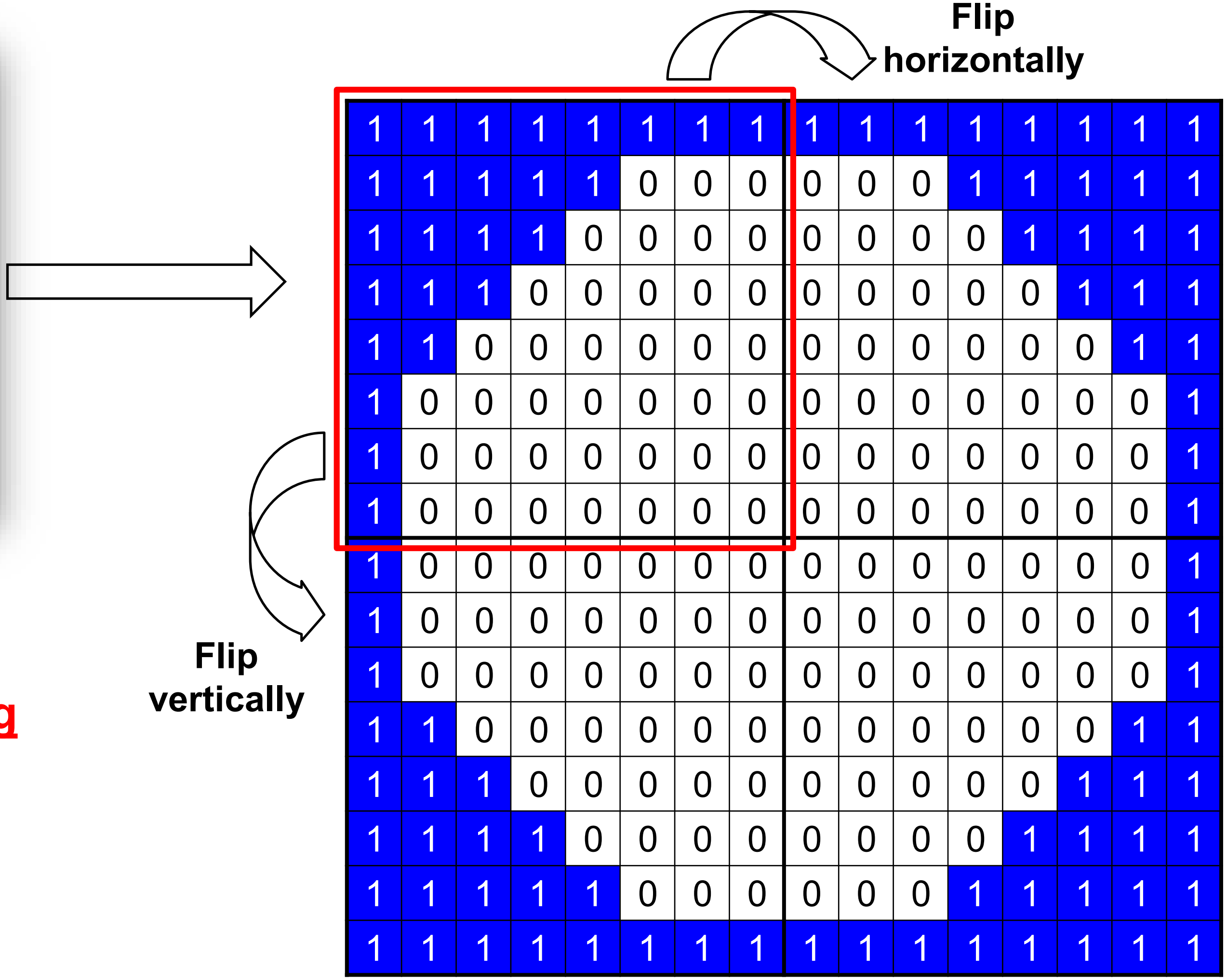
Exercise 5: Tile Declaration

- One single tile is necessary as the background can be constructed by flipping it horizontally and/or vertically vertically.



```
// Custom tile
u8 coverTile[64] = {
    1,1,1,1,1,1,1,1,
    1,1,1,1,1,0,0,0,
    1,1,1,0,0,0,0,0,
    1,1,0,0,0,0,0,0,
    1,1,0,0,0,0,0,0,
    1,0,0,0,0,0,0,0,
    1,0,0,0,0,0,0,0,
    1,0,0,0,0,0,0,0,
};
```

Component 0 of the palette is transparent
<u>If there is something below</u>



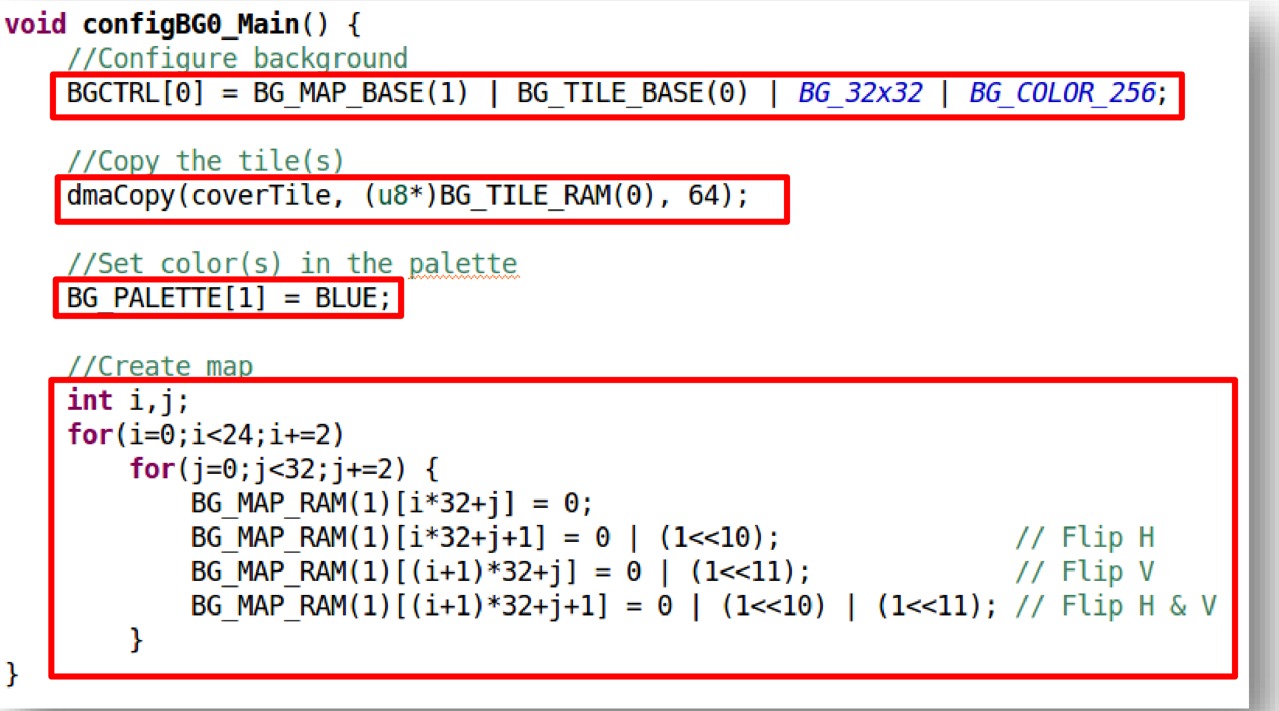
Flip horizontally

Flip vertically

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

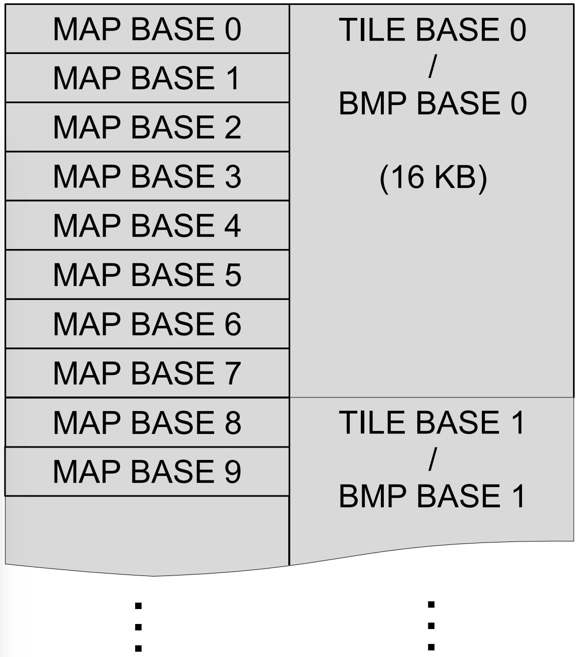
Exercise 5: Configure Background

- Complete the function **configBG0_Main()**
  1. Configure background BG0 in tiled mode using a 32x32 map, tiles with 8bit pixels, the tile base 0 and a map base between 1 and 7 as specified in the exam sheet.
  2. Transfer custom tile to the proper location in memory
  3. Assign color of the used component of the palette
  4. Create map



```c
void configBG0_Main() {
    //Configure background
    BGCTRL[0] = BG_MAP_BASE(1) | BG_TILE_BASE(0) | BG_32x32 | BG_COLOR_256;

    //Copy the tile(s)
    dmaCopy(coverTile, (u8*)BG_TILE_RAM(0), 64);

    //Set color(s) in the palette
    BG_PALETTE[1] = BLUE;

    //Create map
    int i,j;
    for(i=0;i<24;i+=2)
        for(j=0;j<32;j+=2) {
            BG_MAP_RAM(1)[i*32+j] = 0;
            BG_MAP_RAM(1)[i*32+j+1] = 0 | (1<<10);               // Flip H
            BG_MAP_RAM(1)[(i+1)*32+j] = 0 | (1<<11);             // Flip V
            BG_MAP_RAM(1)[(i+1)*32+j+1] = 0 | (1<<10) | (1<<11); // Flip H & V
        }
}
```

VRAM bank



MAP BASE 0	TILE BASE 0 / BMP BASE 0 (16 KB)
MAP BASE 1	
MAP BASE 2	
MAP BASE 3	
MAP BASE 4	
MAP BASE 5	
MAP BASE 6	
MAP BASE 7	
MAP BASE 8	TILE BASE 1 / BMP BASE 1
MAP BASE 9	
⋮	⋮

Exercise 5: Tile Declaration (Version 2)

- Multiples tile, no flipping of the tiles.

```
// With 256-color palette
u8 coverTile00[64] = {
      1,1,1,1,1,1,1,1,
      1,1,1,1,1,0,0,0,
      1,1,1,1,0,0,0,0,
      1,1,1,0,0,0,0,0,
      1,1,0,0,0,0,0,0,
      1,0,0,0,0,0,0,0,
      1,0,0,0,0,0,0,0,
      1,0,0,0,0,0,0,0,
};
```

```
// With 256-color palette
u8 coverTile01[64] = {
      1,1,1,1,1,1,1,1,
      0,0,0,1,1,1,1,1,
      0,0,1,0,1,1,1,1,
      0,0,0,0,0,1,1,1,
      0,0,0,0,0,0,1,1,
      0,0,0,0,0,0,0,1,
      0,0,0,0,0,0,0,1,
      0,0,0,0,0,0,0,1,
};
```

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

```
// With 256-color palette
u8 coverTile10[64] = {
      1,0,0,0,0,0,0,0,
      1,0,0,0,0,0,0,0,
      1,0,0,0,0,0,0,0,
      1,1,0,0,0,0,0,0,
      1,1,1,0,0,0,0,0,
      1,1,1,1,0,0,0,0,
      1,1,1,1,1,0,0,0,
      1,1,1,1,1,1,1,1,
};
```

```
// With 256-color palette
u8 coverTile11[64] = {
      0,0,0,0,0,0,0,1,
      0,0,0,0,0,0,0,1,
      0,0,0,0,0,0,0,1,
      0,0,0,0,0,0,1,1,
      0,0,0,0,0,1,1,1,
      0,0,0,0,1,1,1,1,
      0,0,0,1,1,1,1,1,
      1,1,1,1,1,1,1,1,
};
```

Exercise 5: Configure Background (Vers. 2)

- Complete the function **configBG0_Main()**
  1. Configure background BG0 in tiled mode using a 32x32 map, tiles with 8bit pixels, the tile base 0 and a map base between 1 and 7 as specified in the exam sheet.
  2. Transfer custom tiles to the proper location in memory
  3. Assign color of the used component of the palette
  4. Create map

```
    void configBG0_Main() {
        //Configure background
        BGCTRL[0] = BG_32x32 | BG_COLOR_256 | BG_MAP_BASE(1) | BG_TILE_BASE(0);

        //Copy the full tiles to the corresponding RAM location according to the chosen TILE_BASE
        // If dmaCopy is used, do not forget to cast the destination pointer as a 'byte pointer'
        dmaCopy(Tile00, &BG_TILE_RAM(0)[0], 64);
        dmaCopy(Tile01, &BG_TILE_RAM(0)[32], 64);
        dmaCopy(Tile10, &BG_TILE_RAM(0)[64], 64);
        dmaCopy(Tile11, &BG_TILE_RAM(0)[96], 64);

        //Assign components 254 and 255 as explained in the manual
        BG_PALETTE[1] = BLUE;

        //Set the pointer to the RAM location of the chosen MAP BASE
        int i, j;
        for (i = 0; i < 24; i+=2) {
              for (j = 0; j < 32; j+=2) {
                    BG_MAP_RAM(1)[32 * (i) + j] = 0;
                    BG_MAP_RAM(1)[32 * (i) + j + 1] = 1;
                    BG_MAP_RAM(1)[32 * (i + 1) + j] = 2;
                    BG_MAP_RAM(1)[32 * (i + 1) + j + 1] = 3;
              }
        }
    ;
```