Making a pacman game on STM32H750B-DK

The goal of the project is to implement pacman game using C programing language. I'll be using a template project[1] that allready has implemented some basic functionalities and additional drivers.

[1] https://github.com/LAPSyLAB/ORLab-STM32H7/tree/main/STM32H750B-DK_BSP_C_Basic

Video of a pacman game gameplay on STM32H750B-DK

Parts list

- STM32H750B-DK Discovery kit with STM32H750XB MCU

Pacman game

Pac-Man is a classic arcade game released in 1980 by Namco. Player controls Pac-Man, a yellow circular character, navigating through a maze while eating dots and avoiding four ghosts. There are four energizer dots, located in the corners of the maze, which temporarily turn the ghosts blue, allowing Pac-Man to eat them for extra points. The goal of the game is to get as many points as you can, before the ghosts gets you.

You can read more about the game and it's mechanich on https://pacman.holenet.info/.

Implementation

A classic pacman map has a grid of 31 rows and 28 columns and looks like this.

```
static uint8_t grid_data_init[GRID_ROWS][GRID_COLS] = {
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
        {0,2,2,2,2,2,2,2,2,2,2,2,2,0,0,2,2,2,2,2,2,2,2,2,2,2,2,0},
        {0,2,0,0,0,0,2,0,0,0,0,0,2,0,0,2,0,0,0,0,0,2,0,0,0,0,2,0},
        {0,3,0,0,0,0,2,0,0,0,0,0,2,0,0,2,0,0,0,0,0,2,0,0,0,0,3,0},
        {0,2,0,0,0,0,2,0,0,0,0,0,2,0,0,2,0,0,0,0,0,2,0,0,0,0,2,0},
        {0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0},
        {0,2,0,0,0,0,2,0,0,2,0,0,0,0,0,0,0,0,2,0,0,2,0,0,0,0,2,0},
        {0,2,0,0,0,0,2,0,0,2,0,0,0,0,0,0,0,0,2,0,0,2,0,0,0,0,2,0},
        {0,2,2,2,2,2,2,2,0,0,2,2,2,2,0,0,2,2,2,2,0,0,2,2,2,2,2,0},
        {0,0,0,0,0,0,2,0,0,0,0,0,1,0,0,1,0,0,0,0,0,2,0,0,0,0,0,0},
        {0,0,0,0,0,0,2,0,0,0,0,0,1,0,0,1,0,0,0,0,0,2,0,0,0,0,0,0},
        {0,0,0,0,0,0,2,0,0,1,1,1,1,1,1,1,1,1,1,0,0,2,0,0,0,0,0,0},
        {0,0,0,0,0,0,2,0,0,1,0,0,0,0,0,0,0,0,1,0,0,2,0,0,0,0,0,0},
        {0,0,0,0,0,0,2,0,0,1,0,0,0,0,0,0,0,0,1,0,0,2,0,0,0,0,0,0},
        {1,1,1,1,1,1,2,1,1,1,0,0,0,0,0,0,0,0,1,1,1,2,1,1,1,1,1,1},
        {0,0,0,0,0,0,2,0,0,1,0,0,0,0,0,0,0,0,1,0,0,2,0,0,0,0,0,0},
        {0,0,0,0,0,0,2,0,0,1,0,0,0,0,0,0,0,0,1,0,0,2,0,0,0,0,0,0},
        {0,0,0,0,0,0,2,0,0,1,1,1,1,1,1,1,1,1,1,0,0,2,0,0,0,0,0,0},
        {0,0,0,0,0,0,2,0,0,1,0,0,0,0,0,0,0,0,1,0,0,2,0,0,0,0,0,0},
        {0,0,0,0,0,0,2,0,0,1,0,0,0,0,0,0,0,0,1,0,0,2,0,0,0,0,0,0},
        {0,2,2,2,2,2,2,2,2,2,2,2,2,0,0,2,2,2,2,2,2,2,2,2,2,2,2,0},
        {0,2,0,0,0,0,2,0,0,0,0,0,0,2,0,0,2,0,0,0,0,0,2,0,0,0,0,2,0},
        {0,2,0,0,0,0,2,0,0,0,0,0,0,2,0,0,2,0,0,0,0,0,2,0,0,0,0,2,0},
        {0,3,2,2,0,0,2,2,2,2,2,2,1,1,2,2,2,2,2,2,2,0,0,2,2,3,0},
        {0,0,0,2,0,0,2,0,0,2,0,0,0,0,0,0,0,0,2,0,0,2,0,0,2,0,0,0},
        {0,0,0,2,0,0,2,0,0,2,0,0,0,0,0,0,0,0,2,0,0,2,0,0,2,0,0,0},
        {0,2,2,2,2,2,2,0,0,2,2,2,2,0,0,2,2,2,2,0,0,2,2,2,2,2,2,0},
        {0,2,0,0,0,0,0,0,0,0,0,0,2,0,0,2,0,0,0,0,0,0,0,0,0,0,2,0},
        {0,2,0,0,0,0,0,0,0,0,0,0,2,0,0,2,0,0,0,0,0,0,0,0,0,0,2,0},
        {0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0},
        {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
    };
```

pacman.h delivered with ❤️ by EmGithub view raw

Where 0 represents a wall, 1 represents a path, 2 represents a dot and 3 represents an energizer dot.

The player spawn at row 24, ghosts spawns at row 12, and "fruits" spawn at row 18. All between column 14 and 15.

The logic behind the game is pretty straight forward, we just check in what cell we are in, and move to the next one if it's not a wall. A player needs some extra logic to collect the dots and fruits, and ghost needs to move around on their own.

The game goes on until one of the ghosts colides with a player. In a classic game, you loose a life, but in my implementation you loose the game.

Game Logic

To simplify draw calls so we don't need multiple layers, we draw grid, arrows, "score" text, "high score" text and dots once at the start of the game.

Player

Player logic is pretty straight forward. We just need to check if it can move in a given direction, and update the score. Most other things in the following functions are just translations between pixels and grid. The only special part is the "tunnel" where we just teleport the player on the other side of the map by updating it's position in grid and on screen.

```
void Player_Logic(GameData *data){
    int normalized_x = (data->player.pos_x - 125);
    int normalized_y = (data->player.pos_y - 9);
    int cx = (normalized_x / 8);
    int cy = (normalized_y / 8);

    if ((data->player.dir_x != 0 && data->player.cdir_x == -data->player.dir_x)
        || (data->player.dir_y  != 0 && data->player.cdir_y == -data->player.dir_y)
        || ((data->player.dir_x != 0 || data->player.dir_y != 0) && normalized_x%8 > 5 && normalized_y%8
        data->player.cdir_x = data->player.dir_x;
        data->player.cdir_y = data->player.dir_y;
    }


    int x = cx + data->player.cdir_x;
    int y = cy + data->player.cdir_y;
    UTIL_LCD_FillCircle(data->player.pos_x, data->player.pos_y, 5, COLOR_BG);
    if(x == GRID_COLS){
        data->player.pos_x+=data->player.cdir_x;
    }
    else if(x==-1){
        data->player.pos_x = 27*8+125;
        x = GRID_COLS-1;
    }
    else if (x == GRID_COLS+1) {
        data->player.pos_x = 128;
        x = 0;
    }
    if(data->grid[y][x]>0){
        if (data->grid[y][x] == 2) {
            data->score+=10;
            data->collectables--;
        }else if (data->grid[y][x] == 3) {
            data->collectables--;
            data->score+=50;
            data->timer_start = data->timer;
            GhostDisable(data);
        }
        if (data->max_score < data->score) {
            data->max_score = data->score;
        }
        data->grid[y][x] = 1;
        data->player.pos_x+=data->player.cdir_x;
        data->player.pos_y+=data->player.cdir_y;
    }else if(data->player.cdir_x == 1 && normalized_x%8 < 6){
        data->player.pos_x+=data->player.cdir_x;
    }else if(data->player.cdir_y == 1 && normalized_y%8 < 6) {
        data->player.pos_y+=data->player.cdir_y;
    }
    UTIL_LCD_FillCircle(data->player.pos_x, data->player.pos_y, 5, data->player.color);
}
```

pacman.c delivered with ❤️ by EmGithub view raw

We also need to implement touch controlls for the player. We could connect a physical buttons or a joystick to the MCU, but since it has a touch display we can implenet movement by creating sections in display.

I decided to have up and down on both sides of the screen, so it's easier to play the game.

Even though I set the touchscreen size the same as the screen, we still need to check if the given position is less than the screen.

```
void Player_Touch_Controls(GameData *data){
    if (data->touch_x < 128) {
        if(data->touch_y > 41) {
            if(data->touch_y > 119) {
                if(data->touch_y > 189) {
                    data->player.dir_x = 0;
                    data->player.dir_y=1;
                }else if(data->touch_y < data->screen_y)  {
                    data->player.dir_x=-1;
                    data->player.dir_y=0;
                }else {
                    data->player.dir_x=0;
                    data->player.dir_y=0;
                }
            } else {
                data->player.dir_x=0;
                data->player.dir_y=-1;
            }
        }else {
            data->player.dir_x=0;
            data->player.dir_y=0;
        }
    } else if (data->touch_x > 352 && data->touch_x < data->screen_x) {
        if(data->touch_y > 41) {
            if(data->touch_y > 119) {
                if(data->touch_y > 189) {
                    data->player.dir_x=0;
                    data->player.dir_y=1;
                }else if(data->touch_y < data->screen_y) {
                    data->player.dir_x=1;
                    data->player.dir_y=0;
                }else {
                    data->player.dir_x=0;
                    data->player.dir_y=0;
                }
            } else {
                data->player.dir_x=0;
                data->player.dir_y=-1;
            }
        }else {
            data->player.dir_x=0;
            data->player.dir_y=0;
        }
    } else {
        data->player.dir_x=0;
        data->player.dir_y=0;
    }
}
```

pacman.c delivered with ❤ by EmGithub view raw

Ghost

Ghost logic is pretty much the same as the player logic. The three differences are that it doesn't need to update the score, that it has to redraw dots and that it randomly decides in what directions the ghost will move next at the intersections.

```
void Ghost_Logic(GameData *data, Player *ghost){

    int normalized_x = (ghost->pos_x - 125);
    int normalized_y = (ghost->pos_y - 9);
    int cx = (normalized_x / 8);
    int cy = (normalized_y / 8);

    if (normalized_x%8 > 5 && normalized_y%8 > 5){
        if(ghost->cdir_x != 0){
            if(rand()%2 == 0 || data->grid[cy][cx+ghost->cdir_x]==0){
                if(rand()%2 == 0 && data->grid[cy+1][cx]>0){
                    ghost->cdir_x = 0;
```

```
                    ghost->cdir_y = 1;
                }else if(data->grid[cy-1][cx]>0){
                    ghost->cdir_x = 0;
                    ghost->cdir_y = -1;
                }
            }
        }else{
            if(rand()%2 == 0 || data->grid[cy+ghost->cdir_y][cx]==0){
                if(rand()%2 == 0 && data->grid[cy][cx+1]>0){
                    ghost->cdir_x = 1;
                    ghost->cdir_y = 0;
                }else if(data->grid[cy][cx-1]>0){
                    ghost->cdir_x = -1;
                    ghost->cdir_y = 0;
                }
            }
        }
    }
    int x = cx + ghost->cdir_x;
    int y = cy + ghost->cdir_y;
    Draw_Ghost(ghost->pos_x, ghost->pos_y, COLOR_BG);

    if(ghost->cdir_x<0 || ghost->cdir_y<0) Draw_GridCell(data, cy, cx);
    else Draw_GridCell(data, cy-ghost->cdir_y, cx-ghost->cdir_x);

    if(x == GRID_COLS){
        ghost->pos_x+=ghost->cdir_x;
    }
    else if(x==-1){
        ghost->pos_x = 27*8+125;
        x = GRID_COLS-1;
    }
    else if (x == GRID_COLS+1) {
        ghost->pos_x = 128;
        x = 0;
    }
    if(data->grid[y][x]>0){
        ghost->pos_x+=ghost->cdir_x;
        ghost->pos_y+=ghost->cdir_y;
    }else if(ghost->cdir_x == 1 && normalized_x%8 < 6){
        ghost->pos_x+=ghost->cdir_x;
    }else if(ghost->cdir_y == 1 && normalized_y%8 < 6) {
        ghost->pos_y+=ghost->cdir_y;
    }


    ghost->ghostDraw(data, ghost);
}
```

pacman.c delivered with ❤️ by EmGithub view raw

For ghosts 2, 3 and 4, we need to implenet wait functions. The ghosts waits until player has left 150 dots, 100 dots and 50 dots respectedly, and then get's moved to a ghost starting position so they can continue their jurney.

```
void GhostWait2(GameData *data, Player *ghost){
    if(data->collectables == 150){
        Draw_Ghost(ghost->pos_x, ghost->pos_y, COLOR_BG);
        ghost->pos_x = 240;
        ghost->pos_y = 104;
        data->Ghost2 = Ghost_Logic;
    }else Ghost_Logic(data, ghost);
}
```

pacman.c delivered with ❤️ by EmGithub view raw

When the energizer is not active, the ghost is just checking if the player is colliding with it and it sets game over.

```
void Ghost_Enable(GameData *data, Player *ghost){
    if (abs(data->player.pos_x - ghost->pos_x)<8
        && abs(data->player.pos_y - ghost->pos_y)<8){
        data->refresh = Game_Over;
```

```
    }
    Draw_Ghost(ghost->pos_x, ghost->pos_y, ghost->color);

}
```

pacman.c delivered with ❤ by EmGithub view raw

But when the energizer is active a gray color is drawn, and the score is adjusted acordingly if the player eats the ghost.

The energizer is active for around 9 seconds, and then normal functionality of a ghost is resumed.

```
void Ghost_Disable(GameData *data, Player *ghost){
    if (data->timer - data->timer_start > GHOST_IMMUNITY){
            GhostEnable(data);
    }
    if (abs(data->player.pos_x - ghost->pos_x)<10
        && abs(data->player.pos_y - ghost->pos_y)<10){
        data->score+=ghost_points[data->ghost];
        data->ghost++;
        if(data->ghost == 4){
            data->score+=ghost_points[data->ghost];
            data->ghost = 0;
        }
        ResetGhost(ghost);
    }
    Draw_Ghost(ghost->pos_x, ghost->pos_y, UTIL_LCD_COLOR_LIGHTGRAY);
}
```

pacman.c delivered with ❤ by EmGithub view raw

Fruit

Fruits spawns twice per level, once at 174 dots left and second time at 74 dots left and despawn after around 9 seconds.

```
    }else if(data->collectables==174 || data->collectables == 74){
        timer_start = data->timer;
        Fruit=fruits[data->fruit];
    }
```

pacman.c delivered with ❤ by EmGithub view raw

The only thing that fruit does, is checking if the player is at it's position and increases the score if it is, drawing itself and despawning if it's not collected in time.

```
void Draw_Fruit_1(GameData *data){
    if(data->timer - timer_start > FRUIT_DESPAWN){
        Fruit_Clear();
        return;
    }
    if(data->player.pos_x > 236 && data->player.pos_x < 244
       && data->player.pos_y > 148 && data->player.pos_y < 156){
        data->score+=10;
        Fruit_Clear();
        return;
    }

    UTIL_LCD_DrawHLine(237, 150, 4, UTIL_LCD_COLOR_RED);
    UTIL_LCD_DrawHLine(236, 151, 6, UTIL_LCD_COLOR_RED);
    UTIL_LCD_DrawHLine(236, 152, 6, UTIL_LCD_COLOR_RED);
    UTIL_LCD_DrawHLine(237, 153, 4, UTIL_LCD_COLOR_RED);
}
```

pacman.c delivered with ❤ by EmGithub view raw

This function only clears fruit from the screen and sets the fruit function to empty function so the fruit is not active anymore.

```
void Fruit_Clear(){
    UTIL_LCD_DrawHLine(236, 148, 8, COLOR_BG);
    UTIL_LCD_DrawHLine(236, 149, 8, COLOR_BG);
    UTIL_LCD_DrawHLine(236, 150, 8, COLOR_BG);
    UTIL_LCD_DrawHLine(236, 151, 8, COLOR_BG);
    UTIL_LCD_DrawHLine(236, 152, 8, COLOR_BG);
    UTIL_LCD_DrawHLine(236, 153, 8, COLOR_BG);
    UTIL_LCD_DrawHLine(236, 154, 8, COLOR_BG);
    UTIL_LCD_DrawHLine(236, 155, 8, COLOR_BG);
    Fruit = Draw_Fruit;
}
```

pacman.c delivered with ❤ by EmGithub view raw

Comment

The game would run smoother if CPU cache was enabled, but it was causing unexpected bugs so I had to turn it off.

The screen is pretty small, so creating animations for ghosts and a player was not "wroth it" since it wouldn't be that noticable, but the performance would be slithly worse.

There is also a bug that a player will change direction on a last button press, even if the press happened a while ago, so instead of trying to fix it I decided it will be a feature so you can pre-move your next move.

Things for STM32

Generating font

I wanted a font with arrow keys, so I look online and found https://github.com/zst-embedded/STM32-LCD_Font_Generator project by zst-embedded, that generates font with python. The code didn't work since there was a lot of changes in the libraries used and I had to change a few things to make it work.

More specificly this function of PIL library that was deprecated:

```
        w, h = font.getsize(ch)
```

stm32-font.py delivered with ❤ by EmGithub view raw

to this

```
        mask = font.getmask(ch)
        width, _ = mask.size
```

stm32-font.py delivered with ❤ by EmGithub view raw

and while I was at it, I also added a few more functionalities, like character offset and generating source files (.c) instead of just header files (.h).

Case for the MCU

Since standalone STM32H750B-DK is not the most comfortable thing to hold in hands so I also designed a cases for easier useage of it. The case can be easily opened from the back to access board connectors that are on the back, all other connectors that are on the side are also accessible.

Before printing the case needs to be scaled to match 3D printer tolerances by up to 1%, since it's designed 1:1 with the CAD files that I found https://www.st.com/en/evaluation-tools/stm32h750b-dk.html#cad-resources. Otherwise you will need a lot of sanding if you decide not to do so.

You will need 4 screws. I'm using Ø2.5 10 mm with a head hight of 2.5mm that I had laying around.

The case was designed in blender and you can find it on https://makerworld.com/en/models/517051.

Case - front side

Case - back side

Case - opened