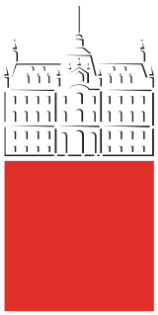
Poročilo 2. domača naloga



Univerza *v Ljubljani*
Fakulteta *za računalništvo*
in informatiko

Izdelal: Matija Baša, 63220014

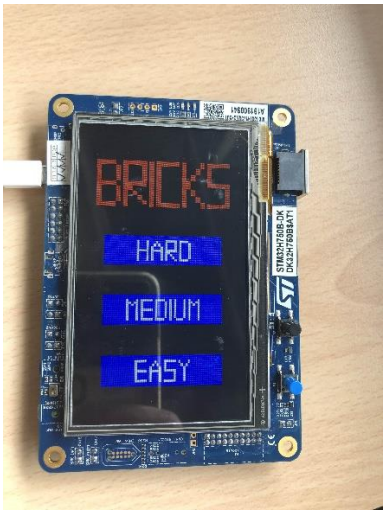
Datum: 3. 8. 2024

Fakulteta: FRI - Univerza v Ljubljani

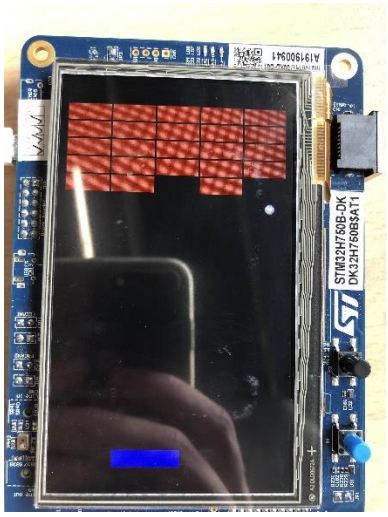
Predmet: Organizacija Računalnikov

Mentor: dr. Robert Rozman

Pri predmetu Organizacija Računalnikov, sem se odločil, da bom izdelal računalniško igrico »bricks« ali »brick breaker«. Igra izgleda tako da igralec z prstom upravlja igralno ploščico z katero mora zadeti žogico, ki se odbije v kocke in jih ob trku uniči. Cilj igre je da igralec uniči vse kocke preden mu žogica pade na tla. Pred začetkom pa si še igralec izbere način težavnosti. Težji način kot izbere več opek bo imel za uničiti in manjša bo igralna ploščica.



Začetni zaslon



Igra

Prikaz pri koncu igre

Za izdelavo igrice sem uporabil mikrokontroler STM32H7. Za programiranje igre pa sem uporabil knjižnico BSP_LCD, ki služi risanju raznih likov na zaslon. Celotno igrico sem sprogramiral v jeziku C.

```
BSP_LCD_Init(0, LCD_ORIENTATION_PORTRAIT);
UTIL_LCD_SetFuncDriver(&LCD_Driver);
```

```
uint32_t ts_status = BSP_TS_Init(0, &hTS);
if(ts_status != BSP_ERROR_NONE) {
  Error_Handler();
}
```

Inicializacija zaslona in dotika na zaslon

Za risanje na zaslon sem uporabljal funkcije UTIL_LCD_FillCircle(), UTIL_LCD_FillRect() in UTIL_LCD_Clear() za počistiti zaslon. Za premikanje igralne ploščice in zaznavanje drugih dotikov na zaslov sem uporabil TS_State.TouchDetected, ki je lahko ali true ali false odvisno ali je uporabnik pritisnih na zaslon. To preverjam v neskončni zanki in vedno ko je TS_State.TouchDetected = true pridobim koordinate dotika z TS_State.TouchY ali TS_State.TouchX. Pri premiku igralca je pomemben le TS_State.TouchY saj se premika le po Y osi. Premike pa naredimo tako, da najprej preverimo ali je trenutni dotik drugačen od prejšnjega, saj če ni nam ni potrebno na novo risati igralca. V primeru, da je drugačen moramo najprej trenutnega igralca izbrisati in nato na novo narisati vendar na drugi koordinati.

```
if (TS_State.TouchDetected) {
    y = TS_State.TouchY;
    if(y != last_y){
        gameStart = true;
        if(y < RECT_HEIGHT / 2){
            y = RECT_HEIGHT / 2;
        }

        if(y > y_size - RECT_HEIGHT / 2){
            y = y_size - RECT_HEIGHT / 2;
        }
        playerx = y;
        UTIL_LCD_SetBackColor(UTIL_LCD_COLOR_BLACK);
        UTIL_LCD_FillRect(20, 0, RECT_WIDTH, y_size, UTIL_LCD_COLOR_BLACK);
        UTIL_LCD_SetTextColor(UTIL_LCD_COLOR_BLUE);
        UTIL_LCD_FillRect(20, y - RECT_HEIGHT / 2, RECT_WIDTH, RECT_HEIGHT, UTIL_LCD_COLOR_BLUE);
    }
    last_y = y;
```

Za prikaz začetnega zaslona uporabljam funkcijo Display_InitialContent(), ki vrne celo število. To število nam služi, da vemo kakšno težavnost je igralec izbral. V tej funkciji imamo neskončno zanko, ki jo prekinemo, ko igralec pritisne na eno izmed težavnosti. Po pritisku se igralcu prikaže odštevanje, ki traja 3 sekunde, da se lahko igralec pripravi na igro. Kocke, igralca in žogo pa nariše funkcija setupScreen(), ki jo kličemo preden se igra začne. Številke, ki se na zaslon izpisujejo za odštevanje sem naredil z risanjem pravokotnikov z funkcijo UTIL_LCD_FillRect()

```
while (1){
  BSP_TS_GetState(0, &TS_State);
  if (TS_State.TouchDetected){
      uint16_t y = TS_State.TouchY;
      uint16_t x = TS_State.TouchX;
      if(y > 272 / 2 - 100 && y < 271 / 2 + 100){
          if(x > 50 && x < 100){
              RECT_HEIGHT = 80;
              gameMode = 1;

              setupScreen();

              UTIL_LCD_FillRect(200, 272/2-20, 10, 40, UTIL_LCD_COLOR_RED);
              UTIL_LCD_FillRect(175, 272/2-10, 10, 30, UTIL_LCD_COLOR_RED);
              UTIL_LCD_FillRect(150, 272/2-20, 10, 40, UTIL_LCD_COLOR_RED);
              UTIL_LCD_FillRect(150, 272/2+10, 50, 10, UTIL_LCD_COLOR_RED);

              HAL_Delay(1000);

              UTIL_LCD_FillRect(150, 272/2-20, 60, 60, UTIL_LCD_COLOR_BLACK);
              UTIL_LCD_FillRect(200, 272/2-20, 10, 40, UTIL_LCD_COLOR_RED);
              UTIL_LCD_FillRect(175, 272/2-20, 10, 40, UTIL_LCD_COLOR_RED);
              UTIL_LCD_FillRect(150, 272/2-20, 10, 40, UTIL_LCD_COLOR_RED);
              UTIL_LCD_FillRect(150, 272/2-20, 25, 10, UTIL_LCD_COLOR_RED);
              UTIL_LCD_FillRect(175, 272/2+10, 25, 10, UTIL_LCD_COLOR_RED);

              HAL_Delay(1000);

              UTIL_LCD_FillRect(150, 272/2-20, 60, 60, UTIL_LCD_COLOR_BLACK);
              UTIL_LCD_FillRect(150, 272/2+10, 60, 10, UTIL_LCD_COLOR_RED);

              HAL_Delay(1000);

              UTIL_LCD_FillRect(150, 272/2-20, 60, 60, UTIL_LCD_COLOR_BLACK);


              return 5;
          }
```

Ko se igra začne vsakih 17 milisekund premaknemo žogico. Žogico premaknemo na način da najprej spremenimo koordinate žogice in jo nato narišemo na teh koordinatah z funkcijo DrawBall() pred tem pa seveda moramo izbrisati žogico pred premikom.

```
void UpdateBallPosition(int rows) {
    //izbrišemo prejšnjo poticijo
    DrawBall(ball_x, ball_y, UTIL_LCD_COLOR_BLACK);

    //posodobimo koordinate žoge
    ball_x += ball_dx;
    ball_y += ball_dy;

    //preverimo nove koordinate
    CheckCollisions(rows);

    //če zoga pade na tla se igra zaključi
    if(ball_x-BALL_RADIUS <= 0){
        gameOver = true;
        return;
    }

    //narišemo žogo na novih koordinatih
    DrawBall(ball_x, ball_y, UTIL_LCD_COLOR_WHITE);
}

void DrawBall(int16_t x, int16_t y, uint32_t color) {
  UTIL_LCD_SetTextColor(color);
  UTIL_LCD_FillCircle(x, y, BALL_RADIUS, color);
}
```

Pri vsakem premiku pa moramo preveriti ali je žogica zadela igralca, steno ali opeko. Obenem pa moramo tudi preveriti ali je igralec igro zmagal ali izgubil. Preveriti moramo ali je igralec zmagal tako da imamo števec (bricksBroken), ki ga povečujemo vsakič ko igralec zadane opeko, v glavni zanki pa nato preverimo ali je bila to zadnja opeka. To naredimo tako da primerjamo števec razbitih opek z številom vseh opek, ki ga izračunamo glede na težavnost igre. Preverjanje ali je žogica zadela igralca ali steno je dokaj preprosto, saj samo spremenimo smer žogice ob primeru trka, tako da negiramo spremenljivko ball_dx ali ball_dy. Nekoliko težje je to pri trku z opekami, saj lahko žogica v opeko trči tudi iz drugih smeri. Ta problem sem rešil tako da sem poleg tega, da preverimo ali je bila opeka zadeta tudi to, it katere smeri smo jo zadela. To naredimo tako da preverimo ali je žogica na desni ali levi strani ali pa zgoraj ali spodaj.

```
for (uint16_t i = 0; i < rows; i++) {
      for (uint16_t j = 0; j < BRICKS_ROWS; j++) {
          if (bricks[i][j]) {

              //ugotovimo, kje se opeka nahaja
              uint16_t brick_x = 458 - BLOCK_WIDTH * (i + 1) - 2 * (i + 1);
              uint16_t brick_y = (2 * (j + 1)) + (BLOCK_HEIGHT * j);

              //premerimo ali se je zgodil trk z njo
              if (ball_x + BALL_RADIUS > brick_x && ball_x - BALL_RADIUS < brick_x + BLOCK_WIDTH &&
                  ball_y + BALL_RADIUS > brick_y && ball_y - BALL_RADIUS < brick_y + BLOCK_HEIGHT) {

                  //preverimo iz katere strani se je trk zgodil
                  if (ball_x < brick_x || ball_x > brick_x + BLOCK_WIDTH) {
                      ball_dx = -ball_dx;
                  }
                  if (ball_y < brick_y || ball_y > brick_y + BLOCK_HEIGHT) {
                      ball_dy = -ball_dy;
                  }

                  //odstranimo opeko
                  bricks[i][j] = 0;
                  bricksBroken++;
                  UTIL_LCD_FillRect(brick_x, brick_y, BLOCK_WIDTH, BLOCK_HEIGHT, UTIL_LCD_COLOR_BLACK);

                  return;
              }
          }
      }
  }
```

Pri nalogi nisem imeli večjih težav, najtežje oziroma največ časa sem porabil pri programiranju logike za razbiranje opek. Z projektom sem zadovoljen in je na koncu izpadel boljše kot sem si ga na začetku zamislil, saj sem dodal nekatere funkcionalnosti, ki jih nisem nameraval ob začetku programiranja.

Link do github projekta: https://github.com/Matija46/STM32h7_bricks/tree/master