STM32 Igralna Konzola

Timotej Božič, Nik Jukič

2022

Contents

1 Uvod

Vsa koda projekta je dostopna na github: https://github.com/tibozic/stm32_console
Vsa koda poročila z videi je dostopna na github: https://github.com/tibozic/stm32_console_porocilo

Parts list:

- STM32H750 Discovery kit

Začetni cilj je bil vzpostavitev vgrajenega zaslona na plošči brez kakršnekoli knjižnice in potem napisati neko osnovno knjižnico za uporabo grafičnih elementov (risanje črt, kvadratov, trikotnikov).
Ker nama po kakšnem tednu raziskovanja, branja dokumentacije in testiranja ni uspelo narediti konkretnega napredka, sva se odločila, da raje uporabiva knjižnico za vzpostavitev zaslona in risanje elementov, midva pa se bova bolj osredotočila na končen produkt, ki naj bi bil osnovna igralna konzola z nekaj enostavnimi igrami.
Ker ima plošča, ki jo uporabljava na voljo zaslon z možnostjo detekcije dotika, sva so odločila da uporabiva to namesto "joystick", saj v tem primeru, za implementacijo nebi potrebovala nobenih dodatnih delov.

Knjižnico, ki sva jo uporabila je TouchGFX, ki je opitimiziran za uporabo z napravami STM32. Knjižnica je priložena tudi z TouchGFX Designer, ki je orodje za grafično oblikovanje in deluje na principu "What You See Is What You Get" in omogoča tudi "Drag & Drop" elementov.

2 Lastnosti TouchGFX

2.1 TouchGFX Designer

TouchGFX Designer je zelo uporaben za osnovne in statične strani, za kaj kompleksnejšega pa je potrebna dodatna koda.

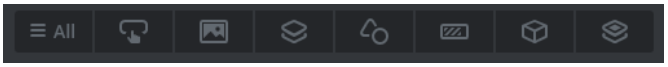

Figure 1: Menu za dodajanje elementov

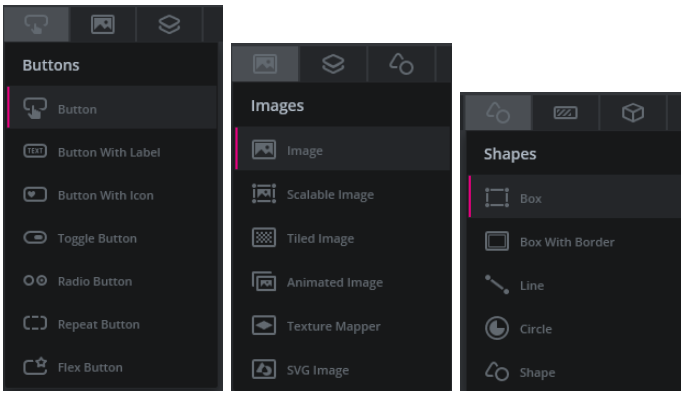

Figure 2: Gumbi, slike in oblike

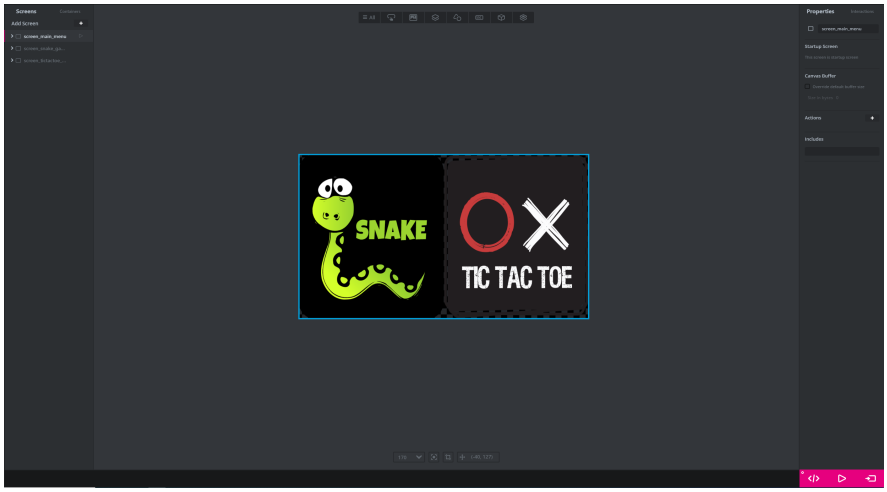

Figure 3: TouchGFX Designer celoten zaslon

Ko zaključimo z oblikovanjem zaslona, pritisnemo gumb "simulate" v spodnjem desnem delu zaslona. To lahko naredimo le če nismo uporabili nobenih zunanjih spremenljivk (zunanjih gumbov, LED, itd.). Ko to naredimo se bo naš projekt zgradil in ga lahko sedaj preizkusimo. V primeru uporabe zunanjih spremenljivk, pa moramo naš projekt zgraditi in naložiti na ploščico, ker nam v tem primeru moznost simulacije ni na voljo. To storimo tako, da pritisnemo gumb generate v spodnjem desnem kotu. Program nato generira kodo za naš dizajn. Projekt lahko sedaj odpremo v CubeIDE, kjer lahko spreminjamo kodo ali pa naložimo program na ploščico.

Ko je projekt generiran, se ustvarijo dve glavni mapi:

- generated
- gui

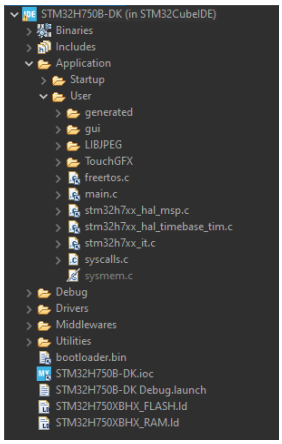

Figure 4: Generirane datoteke

V mapi generated so definirani vsi elementi, ki smo jih ustvarili v TouchGFX Designer, slike ki smo jih dodali ter "Base" datoteke za vse poglede, ki smo jih ustvarili. V tej mapi ne smemo spreminjati nobenih datotek, saj bodo naše spremembe povožene vsakič ko spremenimo kaj v Designer-ju.

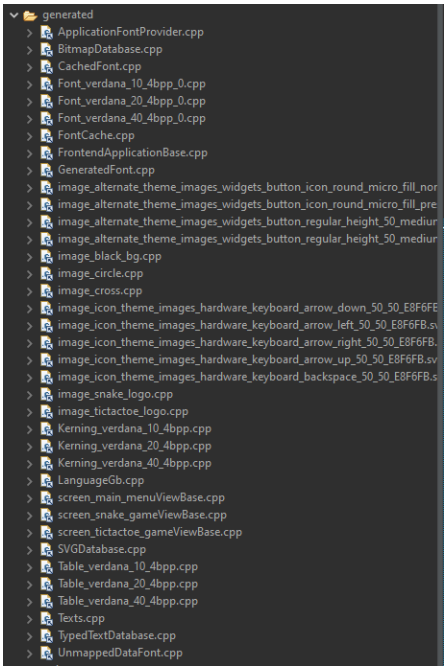

Figure 5: Mapa generated

V mapi gui pa so datoteke za vsak pogled, ki smo ga ustvarili. Te datoteke so namenjene našemu urejanju.

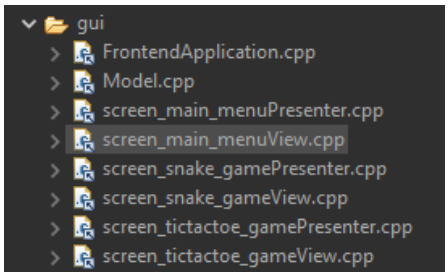

Figure 6: Mapa gui

2.2 TouchGFX Library

Ko je projekt zgeneriran je uporaba knjižnice dokaj preprosta. Na objekte kličemo funkcije kot so .setXY(), .setVisible(), ali pa inicializiramo nove objekte s funkcijami kot so Box().
Ko nek objekt posodobimo ga moraš označiti za "redraw" s funkcijo .invalidate().
Glavna slabost te knjižnice je, da morajo biti vsi elementi na ekranu statično definirani, to nama je povzročalo precej težav pri Snake game-u, saj sva morala maksimalno dolžino kače določiti statično, in za vsak njen del na začetku programa statično rezervirati prostor. V primeru, da bi želela imeti kačo čez cel ekran bi potrebovala okoli 250 tisoč delov rezervirati statično na začetku programa. To je zelo problematično, saj ponavadi vgrajene naprave nimajo na razpolago veliko pomnilnega prostora, ta pa bi bil zaseden skozi celoten življenski cikel programa. Ta problem sva rešila tako, da sva povečala širino kače, kar je zmanjšalo število potrebnih delov na manj kot 5000, kar še vedno ni idealno, je pa izvedljivo.

3 Main Menu

Izoblikovala sva preprost meni, preko katerega lahko igralec z dotikom izbira med vsemi igrami, ki so na voljo.

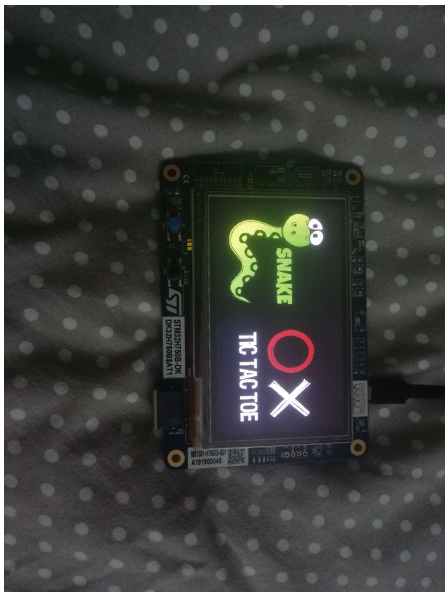

Figure 7: Main menu

4 Igre

Pri izboru iger, sva se omejila na tiste, ki bi bile relativno preproste za implementacijo, in med katerimi je prisotna raznolikost v samem igranju. Na koncu sva si izbrala tri "retro" igre in skušala ustvariti čim bolj zvesto rekreacijo.

- Snake Game
- Tic Tac Toe

4.1 Snake Game

V tej igri nadzorujemo premikanje kače po 480x272 ploščic velikem prostoru. Po prostoru se naključno generira košček hrane. Kačo premikamo s pritiski na smerne gumbe.

Cilj igre je s kačo pojesti čim več hrane brez da se kača zaleti sama vase ali v robove ekrana.
Video: https://github.com/tibozic/stm32_console_porocilo/tree/master

4.1.1 Koda igre

Inicializacija spremenljivk.

```
bool game_started = false;
short snake_direction = SNAKE_DOWN;
int snake_length = 1;

snake_piece *head = NULL;
snake_piece *tail = NULL;

touchgfx::Box snake_pixels[MAX_SNAKE_PIECES];
```

Inicializacija začetnega stanja igre, kar vključuje:

- začetna pozicija in orientacija kače
- skritje gumbov in oken prejšnih instanc

```
void screen_snake_gameView::game_snake_start()
{
  // first hide the button
  btn_snake_start.setVisible(false);
  btn_snake_start.invalidate();

  btn_back.setVisible(false);
  btn_back.invalidate();

  snake_head.setVisible(false);
  snake_head.invalidate();

  lbl_game_over.setVisible(false);
  lbl_game_over.invalidate();

  snake_pixels[0] = Box();

  if( head == NULL )
    head = (snake_piece*)malloc(sizeof(snake_piece));

  if( head == NULL ) {
    error();
    return;
  }

  head->pixel = &snake_pixels[0];

  head->pixel->setPosition(10, 20, 10, 10);
  head->pixel->setColor(touchgfx::Color::getColorFromRGB(0, 255, 50));
  add(*head->pixel);
  head->pixel->getParent()->invalidate();

  head->next = NULL;
  head->prev = NULL;
  head->old_x = 0;
  head->old_y = 0;

  tail = head;

  snake_direction = SNAKE_DOWN;

  snake_length = 1;

  game_started = true;

  // game_snake_loop();
}
```

Vecina kode se nahaja v glavni funkciji, ki je v bistvu glavna zanka celotne igre.

Pregledamo če je glava kače izven igralnega polja

```
if( head->pixel->getX() > SCREEN_WIDTH || head->pixel->getX() < 0 ||
    head->pixel->getY() > SCREEN_HEIGHT || head->pixel->getY() < 0)
{
    game_started = false;
}
```

Pregledamo če se je kača zaletela sama vase

```
snake_piece *snake_part = head->next;
while( snake_part != NULL )
{
    if( snake_part->pixel->getX() == head->pixel->getX() && snake_part->pixel->
    getY() == head->pixel->getY() ) {
        game_started = false;
        break;
    }

    snake_part = snake_part->next;
}
```

Pregledamo, če je kača pojedla hrano

```
snake_part = head;

while( snake_part != NULL )
{
    if( snake_part->pixel->getX() == food.getX() && snake_part->pixel->getY() ==
    food.getY() ) {
      // ...
    }
    snake_part = snake_part->next;
}
```

V primeru, da je kača pojedla hrano, jo moramo podaljšati (dodati nov del)

```
snake_piece *new_piece = (snake_piece*)malloc(sizeof(snake_piece));

if( new_piece == NULL ) {
    error();
    return;
}

snake_pixels[snake_length-1] = Box();
new_piece->pixel = &snake_pixels[snake_length-1];
new_piece->pixel->setPosition(0, 0, PIXEL_WIDTH, PIXEL_HEIGHT);
new_piece->pixel->setColor(touchgfx::Color::getColorFromRGB(255, 130, 0));
new_piece->pixel->setVisible(true);
add(*new_piece->pixel);
new_piece->pixel->getParent()->invalidate();

tail->next = new_piece;
new_piece->prev = tail;
new_piece->next = NULL;
new_piece->old_x = 0;
new_piece->old_y = 0;
tail = new_piece;
```

Če je kača pojedla hrano moramo tudi generirati nov kos hrane.

```
int food_new_x = pseudo_random(tick) % SCREEN_WIDTH;
int food_new_y = pseudo_random2(tick) % SCREEN_HEIGHT;

food_new_x = food_new_x - (food_new_x % 10);
food_new_y = food_new_y - (food_new_y % 10);

food.setXY(food_new_x, food_new_y);
food.invalidate();

Unicode::snprintf(lbl_score_valBuffer, LBL_SCORE_VAL_SIZE, "%d", snake_length);
lbl_score_val.invalidate();
```

Premik kače za eno polje

```
if( snake_direction == SNAKE_RIGHT )
{
    head->old_x = head->pixel->getX();
    head->old_y = head->pixel->getY();
    head->pixel->setXY(head->old_x+(SNAKE_MOVE), head->old_y);
    head->pixel->getParent()->invalidate();

    snake_piece *piece = head->next;
    while( piece != NULL )
    {
        piece->old_x = piece->pixel->getX();
        piece->old_y = piece->pixel->getY();
        piece->pixel->setXY(piece->prev->old_x, piece->prev->old_y);
        piece->pixel->getParent()->invalidate();
        piece = piece->next;
    }
}
else if( snake_direction == SNAKE_LEFT )
{
    head->old_x = head->pixel->getX();
    head->old_y = head->pixel->getY();
    head->pixel->setXY(head->old_x-(SNAKE_MOVE), head->old_y);
    head->pixel->getParent()->invalidate();

    snake_piece *piece = head->next;
    while( piece != NULL )
    {
        piece->old_x = piece->pixel->getX();
        piece->old_y = piece->pixel->getY();
        piece->pixel->setXY(piece->prev->old_x, piece->prev->old_y);
        piece->pixel->getParent()->invalidate();
        piece = piece->next;
    }
}
else if( snake_direction == SNAKE_UP )
{
    head->old_x = head->pixel->getX();
    head->old_y = head->pixel->getY();
    head->pixel->setXY(head->old_x, head->old_y-(SNAKE_MOVE));
    head->pixel->getParent()->invalidate();

    snake_piece *piece = head->next;
    while( piece != NULL )
    {
        piece->old_x = piece->pixel->getX();
        piece->old_y = piece->pixel->getY();
        piece->pixel->setXY(piece->prev->old_x, piece->prev->old_y);
        piece->pixel->getParent()->invalidate();
        piece = piece->next;
    }
}
else if( snake_direction == SNAKE_DOWN )
{
    head->old_x = head->pixel->getX();
    head->old_y = head->pixel->getY();
    head->pixel->setXY(head->old_x, head->old_y+(SNAKE_MOVE));
    head->pixel->getParent()->invalidate();

    snake_piece *piece = head->next;
    while( piece != NULL )
    {
```

```
        piece->old_x = piece->pixel->getX();
        piece->old_y = piece->pixel->getY();
        piece->pixel->setXY(piece->prev->old_x, piece->prev->old_y);
        // piece->pixel->setXY(head->old_x, head->old_y+(SNAKE_MOVE));
        piece->pixel->getParent()->invalidate();
        piece = piece->next;
    }
}

head->pixel->getParent()->invalidate();
```

Ko igralec pritisne gumb za spremembo smeri kače, se kliče naslednja funkcija (za vsako smer druga), ki shrani smer v globalno spremenljivko.

```
void screen_snake_gameView::change_direction_up() {
  snake_direction = SNAKE_UP;
}
```

Funkciji za generiranje pozicije novega kosa hrane.

```
int screen_snake_gameView::pseudo_random(int tick) {
  return (3*snake_length + SNAKE_SPEED*3*tick + 3*tail->pixel->getX() * 3*tail->
    pixel->getY());
}

int screen_snake_gameView::pseudo_random2(int tick) {
  return (7*snake_length + SNAKE_SPEED*7*tick + 7*tail->pixel->getX() * 7*tail->
    pixel->getY());
}
```

4.2 Tic Tac Toe

To je vsem znana igra za 2 igralca, kjer je cilj biti prvi igralec, ki spravi 3 enake simbole v vrsto. Video: https://github.com/tibozic/stm32_console_porocilo/tree/master

4.2.1 Koda igre

Inicializacija spremenljivk.

```
char board[3][3];
bool turn = true; // true -> player 1, false -> player2
short turn_number = 0;
bool game_over = false;
short result = 0;
```

Ponastavimo stanje igralne plošče, v primeru da to ni prva instanca igre, ki jo igramo. Skrijemo se morebitna okna in gumbe prejšnih instanc.

```
screen_tictactoe_gameView::screen_tictactoe_gameView()
{
    for(short i = 0; i < 3; ++i) {
        for(short j = 0; j < 3; ++j) {
            board[i][j] = 0;
        }
    }


    game_over = false;
    lbl_game_over.setVisible(false);
    lbl_game_over.invalidate();

```

```
    box_background.setVisible(false);
    box_background.invalidate();

    lbl_result.setVisible(false);
    lbl_result.invalidate();

    btn_back.setVisible(false);
    btn_back.invalidate();

    turn_number = 0;

    turn = true;
}
```

Za vsako polje moramo voditi posebej spremenljivke in metodo, ki hranijo podatke o stanju igre in preverjajo če se je igra že zaključila.

```
void screen_tictactoe_gameView::pos1_clicked() {
  if( !game_over && board[0][0] == 0 ) {
    if( turn ) {
      board[0][0] = 'X';
      pos1_cross.setVisible(true);
      pos1_cross.invalidate();
    } else {
      board[0][0] = 'O';
      pos1_circle.setVisible(true);
      pos1_circle.invalidate();
    }

    turn_number++;

    short temp_result = is_game_over(0, 0);

    if( temp_result != 0 ) {
      game_over = true;
      result = temp_result;
      return;
    }

    turn = !turn;
  }
}
```

V tem delu kode pa glede na stanje spremenljivke game_over izrišemo končni ekran.

```
void screen_tictactoe_gameView::handleTickEvent() {
  if( game_over ) {
    btn_back.setVisible(true);
    btn_back.invalidate();

    box_background.setVisible(true);
    box_background.invalidate();

    lbl_game_over.setVisible(true);
    lbl_game_over.invalidate();

    lbl_result.setVisible(true);

    if( result == 1 )
      Unicode::snprintf(lbl_resultBuffer, LBL_RESULT_SIZE, "Player 1 wins");
    else if( result == 2 )
      Unicode::snprintf(lbl_resultBuffer, LBL_RESULT_SIZE, "Player 2 wins");
    else if( result == 3 )
```

```
      Unicode::snprintf(lbl_resultBuffer, LBL_RESULT_SIZE, "Tie");

    lbl_result.invalidate();
  }
  else {
    if( turn_number == 0 ) {
      btn_back.setVisible(true);
      btn_back.invalidate();
    }
    else {
      btn_back.setVisible(false);
      btn_back.invalidate();
    }

    if( turn )
      Unicode::snprintf(lbl_turnBuffer, LBL_TURN_SIZE, "1");
    else
      Unicode::snprintf(lbl_turnBuffer, LBL_TURN_SIZE, "2");

    lbl_turn.invalidate();
  }
}
```

Sama metoda, ki se izvede ob pritisku na polje, da se preveri morebiten konec igre.

```
short screen_tictactoe_gameView::is_game_over(int x, int y) {
  // 0 -> game isn't over
  // 1 -> player 1 wins
  // 2 -> player 2 wins
  // 3 -> tie

  char symbol;

  if(turn)
    symbol = 'X';
  else
    symbol = 'O';

    //check col
    for(short i = 0; i < 3; i++){
        if(board[x][i] != symbol)
            break;
        if(i == 3-1){
            if( turn )
              return 1;
            else
              return 2;
        }
    }

    //check row
    for(int i = 0; i < 3; i++){
        if(board[i][y] != symbol)
            break;
        if(i == 3-1){
            if( turn )
              return 1;
            else
              return 2;
        }
    }

    //check diagonal
```

```
    if(x == y){
        //we're on a diagonal
        for(int i = 0; i < 3; i++){
            if(board[i][i] != symbol)
                break;
            if(i == 3-1){
                if( turn )
                  return 1;
                else
                  return 2;
            }
        }
    }

    //check anti diag
    if(x + y == 3-1){
        for(int i = 0; i < 3; i++){
            if(board[i][(3-1)-i] != symbol)
                break;
            if(i == 3-1){
                if( turn )
                  return 1;
                else
                  return 2;
            }
        }
    }

  if( turn_number == 9 )
    return 3;

  return 0;
}
```

4.3 Space invaders

Cilj igre je odstraniti vse nasprotnike preden pridejo do igralčeve ladje.
Video:https://github.com/tibozic/stm32_console_porocilo/tree/master

4.3.1 Koda igre

Inicializacija spremenljivk

```
touchgfx::ScalableImage enemies[NUM_OF_ENEMIES];
int enemy_move_direction = MOVE_RIGHT;

bool invaders_game_over = false;
int enemies_killed = 0;
```

Inicializacija zacetnega stanja, ki vsebuje:

- Začetna pozicija ladje
- Skritje gumbov prejšnih instanc iger
- Postavitev vseh nasprotnikov

```
screen_space_invadersView::screen_space_invadersView()
{
  // Hide the reference enemy
```

```
  enemy1.setVisible(false);
  enemy1.invalidate();

  // hide the bullet
  bullet.setVisible(false);
  bullet.setXY(-1, -1);
  bullet.invalidate();

  bullet_enemy.setVisible(false);
  bullet_enemy.setXY(-1, -1);
  bullet_enemy.invalidate();

  lbl_game_over.setVisible(false);
  lbl_game_over.invalidate();

  btn_back.setVisible(false);
  btn_back.invalidate();

  lbl_lose.setVisible(false);
  lbl_lose.invalidate();

  lbl_win.setVisible(false);
  lbl_win.invalidate();

  enemies_killed = 0;

  enemy_move_direction = MOVE_RIGHT;

  // Initialize all the enemies
  int current_x = 0;
  int current_y = 0;
  int counter = 0;

  for(int i = 0; i < NUM_OF_ROWS_OF_ENEMIES; ++i) {
    current_x = 0;
    for(int j = 0; j < NUM_OF_ENEMIES_PER_ROW; ++j) {
      enemies[counter].setBitmap(touchgfx::Bitmap(BITMAP_SPACE_INVADERS_ENEMY_ID)
    );
      enemies[counter].setPosition(current_x, current_y, 50, 52);
      enemies[counter].setScalingAlgorithm(touchgfx::ScalableImage::
    NEAREST_NEIGHBOR);
      enemies[counter].setVisible(true);
      add(enemies[counter]);
      enemies[counter].invalidate();

      current_x += enemy1.getWidth() + SPACE_BETWEEN_ENEMIES_X;
      counter++;
    }
    current_y += enemy1.getHeight() + SPACE_BETWEEN_ENEMIES_Y;
  }

  invaders_game_over = false;
}
```

Vsako urino periodo premaknemo metek, če je bil ta izstreljen.

```
if( bullet.isVisible() ) {
    bullet.setY(bullet.getY() - BULLET_MOVE_SPEED);
    bullet.invalidate();

    if( bullet.getY()+bullet.getHeight() < 0 ) {
        bullet.setVisible(false);
        bullet.invalidate();
```

```
    }
}
```

Če je igralec pritisnik gumb moramo tudi premakniti ladjo.

```
if( btn_left.getPressedState() && ship.getX() > 0 ) {
    ship.setX(ship.getX() - SHIP_MOVE_SPEED);
}
else if( btn_right.getPressedState() && ship.getX() + ship.getWidth() <
    SCREEN_WIDTH ) {
    ship.setX(ship.getX() + SHIP_MOVE_SPEED);
}
ship.getParent()->invalidate();
```

Nasprotnik lahko tudi izstreli metek, vendar je lahko na ekranu le en nasprotnikov metek hkrati.

```
if( !bullet_enemy.isVisible() ) {
    for(int i = 0; i < NUM_OF_ENEMIES; ++i) {
        if( !enemies[i].isVisible() )
            continue;

        if( pseudo_random(tick) < 10 && tick % 13 == 0 ) {
            bullet_enemy.setVisible(true);
            bullet_enemy.setXY(enemies[i].getX() + enemies[i].getWidth()/2,
    enemies[i].getY() + enemies[i].getHeight());
            bullet_enemy.invalidate();
            break;
        }
    }
} else {
    bullet_enemy.setY(bullet_enemy.getY() + BULLET_MOVE_SPEED);

    if( bullet_enemy.getY() > SCREEN_HEIGHT ) {
        bullet_enemy.setVisible(false);
    }
    bullet_enemy.invalidate();
}
```

Premaknemo nasprotnike po vnaprej določeni poti ter preverimo če je igre konec.

```
static int pixels_moved = 0;

if( ++tick%10 == 0 ) {
    if( enemy_move_direction == MOVE_RIGHT ) {
        if( enemies[NUM_OF_ENEMIES_PER_ROW-1].getX() + enemies[
    NUM_OF_ENEMIES_PER_ROW].getWidth() < SCREEN_WIDTH ) {
            for(int i = 0; i < NUM_OF_ENEMIES; ++i) {
                enemies[i].setX(enemies[i].getX() + ENEMY_MOVE_SPEED);
                enemies[i].invalidate();
            }
        } else {
            enemy_move_direction = MOVE_DOWN;
        }
    }
    else if( enemy_move_direction == MOVE_LEFT ) {
        if( enemies[0].getX() > 10 ) {
            for(int i = 0; i < NUM_OF_ENEMIES; ++i) {
                enemies[i].setX(enemies[i].getX() - ENEMY_MOVE_SPEED);
                enemies[i].invalidate();
            }
        } else {
            enemy_move_direction = MOVE_DOWN;
        }
    }
```

```
    else if( enemy_move_direction == MOVE_DOWN ) {
        if( pixels_moved < enemies[0].getHeight()/2 ) {
            for(int i = 0; i < NUM_OF_ENEMIES; ++i) {
                enemies[i].setY(enemies[i].getY() + ENEMY_MOVE_SPEED);
                enemies[i].invalidate();
                pixels_moved += ENEMY_MOVE_SPEED;
            }
        }
        else {
            pixels_moved = 0;
            if( enemies[NUM_OF_ENEMIES_PER_ROW-1].getX() + enemies[
    NUM_OF_ENEMIES_PER_ROW].getWidth() < SCREEN_WIDTH ) {
                enemy_move_direction = MOVE_RIGHT;
            }
            else {
                enemy_move_direction = MOVE_LEFT;
            }
        }
    }
}

check_game_over();
```

Če je igralec pritisnik gumb za strel ustvarimo metek.

```
void screen_space_invadersView::fire_bullet()
{
  if( !bullet.isVisible() )
  {
    bullet.setXY(ship.getX() + (ship.getWidth()/2 - bullet.getWidth()/2), ship.
    getY() - bullet.getHeight()/2);
    bullet.setVisible(true);
    bullet.invalidate();
  }
}
```

Pregledamo če je metek zadel katerega od nasprotnikov, če je nasprotnika in metek odstranimo.

```
void screen_space_invadersView::check_bullet_hitbox()
{
  if( bullet.isVisible() ) {
    for(int i = 0; i < NUM_OF_ENEMIES; ++i ) {
      if( !enemies[i].isVisible() )
        continue;

      if( (bullet.getX() < enemies[i].getX()+enemies[i].getWidth()
        && bullet.getX() > enemies[i].getX()
        && bullet.getY() > enemies[i].getY()
        && bullet.getY() < enemies[i].getY() + enemies[i].getHeight())
        || (bullet.getX() + bullet.getWidth() < enemies[i].getX()+enemies[i].
    getWidth()
          && bullet.getX() + bullet.getWidth() > enemies[i].getX()
          && bullet.getY() > enemies[i].getY()
          && bullet.getY() < enemies[i].getY() + enemies[i].getHeight())  )
      {
        // The bullet hit an enemy

        // remove the enemy
        enemies[i].setVisible(false);
        enemies[i].invalidate();

        // remove the bullet
        bullet.setVisible(false);
```

```
        bullet.setXY(-1, -1);
        bullet.invalidate();

        enemies_killed++;
        break;
      }

    }
  }
}
```

Funkcija za pregled konca igre.

```
void screen_space_invadersView::check_game_over()
{
  // check that the player hasn't killed all the enemies
  invaders_game_over = true;
  for(int i = 0; i < NUM_OF_ENEMIES; ++i) {
    if( enemies[i].isVisible()) {
      invaders_game_over = false;
      break;
    }
  }

  // check that the enemy hasn't collided with the player
  for(int i = 0; i < NUM_OF_ENEMIES; ++i) {
    if( !enemies[i].isVisible() )
      continue;

    if( (ship.getX() < enemies[i].getX()+enemies[i].getWidth()
      && ship.getX()+ship.getWidth() > enemies[i].getX()
      && ship.getY() + ship.getHeight() > enemies[i].getY()
      && ship.getY() + ship.getHeight()/2 < enemies[i].getY() + enemies[i].
    getHeight()))
    {
      // Enemy collided with the player
      invaders_game_over = true;
      return;
    }
  }

  // check that the enemy bullet didn't hit the player
  if( (ship.getX() < bullet_enemy.getX()+bullet_enemy.getWidth()
    && ship.getX()+ship.getWidth() > bullet_enemy.getX()
    && ship.getY() + ship.getHeight() > bullet_enemy.getY()
    && ship.getY() + ship.getHeight()/2 < bullet_enemy.getY() + bullet_enemy.
    getHeight()))
  {
    // Enemy collided with the player
    invaders_game_over = true;
    return;
  }

  // check that the enemy hasn't made it past the player
  for(int i = 0; i < NUM_OF_ENEMIES; ++i) {
    if( !enemies[i].isVisible() )
      continue;

    if( enemies[i].getY() > SCREEN_HEIGHT ) {
      // Enemy has gone past the player
      invaders_game_over = true;
      return;
    }
```

```
  }
}
```

Funkcija za generiranje naključnega števila.

```
int screen_space_invadersView::pseudo_random(int tick)
{
  int bullet_x = 7;
  int bullet_y = 7;

  if( bullet.getX() > 0 )
    bullet_x = bullet.getX();

  if( bullet.getX() > 0 )
    bullet_y = bullet.getY();

  return ((7*tick+13*ship.getX()) + (3*bullet_x+3*bullet_y)) % 100;
}
```

5 References