DO178

Développement d'un jeu Pong sur STM32F4 discovery

Robin Rebischung rebisc_r@epita.fr

Francis Visoiu Mistrih visoiu_f@epita.fr

Requirements

System requirements

SR: `Pong` est un jeu développé sur une board `STM32F4 discovery`. Le but du jeu est de simuler un jeu de `ping-pong`, donc avec deux raquettes et une balle, afin de gagner la partie. Le jeu consiste en plusieurs échanges, et lorsque la balle sort du camps d'un joueur, le joueur perd.

High-level requirements

HLR.1: La balle se déplace de raquette en raquete, avec une vitesse et un angle.

```
LLR 1.1: La fonction `game_ball.draw` permet d'afficher la balle à
         l'écran.

LLR 1.2: La fonction `game_ball.update` met à jour la position de la balle
         en fonction de la position courante, la vitesse, l'angle, et les
         limites de l'écran.
```

HLR.2: Au démarrage, la balle est au centre, et avance vers le joueur vert. Le joueur vert est l'utilisateur, et le joueur rouge est l'ennemi.

```
LLR 2.1: L'initialisation du type record `ball` se fait avec une vitesse
         de `10.0` et un angle de `90`. Ainsi, la position est à
         `width'last / 2`, `height'last / 2`. La taille de la balle a un
         rayon de 10.

LLR 2.2: L'ennemi est initialisé de la même manière, et contient une
         couleur rouge, et vert pour l'utilisateur.
```

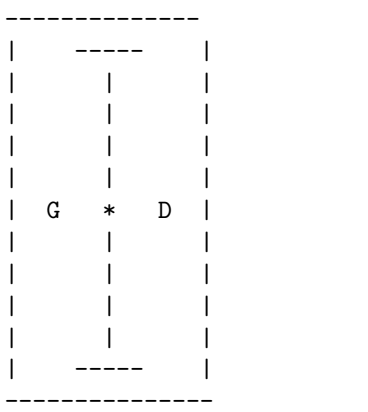
HLR.3: Lorsque la balle touche une raquette, la balle part dans la direction opposée, sur les deux axes, x et y.

```
LLR 3.1: La fonction `core_geometry.intersects` permet de trouver si un
         cercle et un rectangle contiennent une zone commune.

LLR 3.2: Lorsque y'a intersection, l'angle de la balle part à `+45` ou
```

`-45` en fonction de l'angle initial (cos(angle) > 0.0).

HLR.4: Afin de bouger la raquette de l'utilisateur, il faut appuyer sur la partie gauche de l'écran pour aller à gauche, puis la partie droite de l'écran pour aller à droite.



LLR 4.1: Lorsque l'appui est détécté, un déplacement sur l'axe des `x` est éffectué sur la raquette de l'utilisateur. Le déplacement est fait de `10` pixels.

HLR.5: L'utilisateur ne peut pas placer directement la raquette où il souhaite, mais doit utiliser les zones de mouvement afin de bouger la raquette.

LLR 5.1: L'appui sur l'écran permet de déplacer progressivement la raquette, et ne permet pas de choisir une position arbitraire.

HLR.6: A chaque tour, la vitesse de la balle augmente.

LLR 6.1: La fonction `game_ball.update` met à jour la vitesse de la balle.

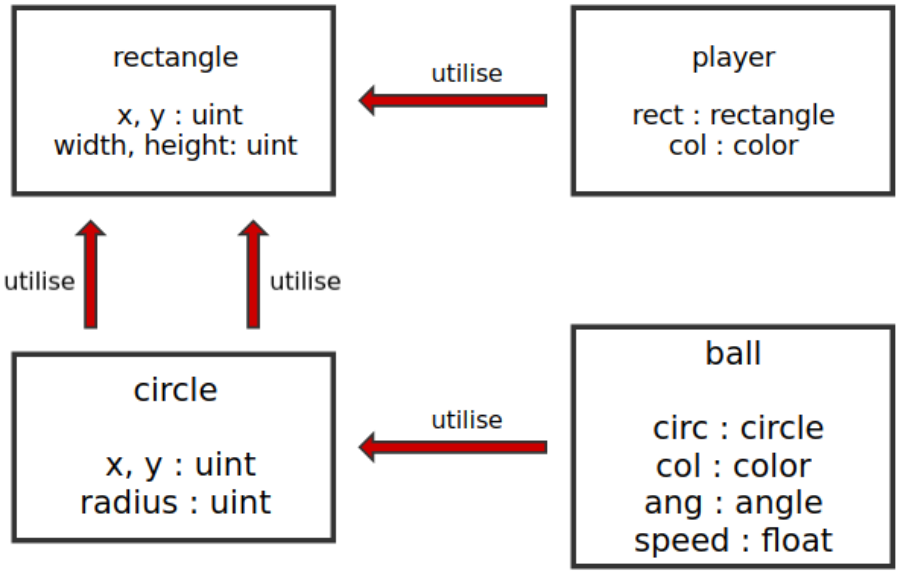


Figure 1: Dépendences

Modules

Core

Core Geometry

- uint
- rectangle
  - "="
  - draw
- circle
  - "="
  - draw

Header

```
with screen_interface; use screen_interface;

package core_geometry is
  subtype uint is natural;
```

```
type rectangle is
  record
    --      w
    -- xy---------
    -- |         |
    -- |         | h
    -- |         |
    -- -----------

    x : uint;
    y : uint;
    w : uint;
    h : uint;
  end record
  with dynamic_predicate => rectangle.w /= 0 and then rectangle.h /= 0;

-- r1: first rectangle.
-- r2: second rectangle.
function "=" (r1, r2 : rectangle) return boolean;

type circle is
  record

    --      ---
    --    /     \
    --    |  xy |
    --    \  |r /
    --      ---

    x : uint;
    y : uint;
    r : uint;
  end record
  with dynamic_predicate => circle.r /= 0;

-- Compare two circles based on their fields.
-- c1: first circle.
-- c2: second circle.
function "=" (c1, c2 : circle) return boolean;

-- Draw a rectangle on the screen.
-- rect: the rectangle.
-- col:  the color for the rectangle to be filled with.
procedure draw(rect : rectangle; col : color);
```

```
  -- Draw a circle on the screen.
  -- circ: the circle.
  -- col:  the color for the circle to be filled with.
  procedure draw(circ : circle; col : color);

  -- Check if a circle and a rectangle intersect.
  -- rect: the rectangle.
  -- circ: the circle.
  function intersects(rect : rectangle; circ : circle) return boolean;

end core_geometry;
```

Implementation

```
with core_math; use core_math;

package body core_geometry is

  function "=" (r1, r2 : rectangle) return boolean is
  begin
    return r1.x = r2.x and then r1.y = r2.y and then r1.w = r2.w
                        and then r1.h = r2.h;
  end "=";

  procedure draw(rect : rectangle; col : color) is
  begin
    for y in rect.y .. rect.y + rect.h loop
      for x in rect.x .. rect.x + rect.w loop
        set_pixel((x, y), col);
      end loop;
    end loop;
  end draw;

  function "=" (c1, c2 : circle) return boolean is
  begin
    return c1.x = c2.x and then c1.y = c2.y and then c1.r = c2.r;
  end "=";

  procedure draw(circ : circle; col : color) is
  begin
    for x in circ.x - circ.r .. circ.x + circ.r loop
      for y in circ.y - circ.r .. circ.y + circ.r loop
        if pow2(x - circ.x) + pow2(y - circ.y) <= pow2(circ.r) then
          set_pixel((x, y), col);
        end if;
```

```
      end loop;
    end loop;
  end draw;

  function intersects(rect : rectangle; circ : circle) return boolean is
    function intersects(circ : circle; x, y : uint) return boolean is
    begin
      return pow2(x - circ.x) + pow2(y - circ.y) <= pow2(circ.r);
    end intersects;
  begin
    for x in rect.x .. rect.x + rect.w loop
      for y in rect.y .. rect.y + rect.h loop
        if intersects(circ, x, y) then
          return true;
        end if;
      end loop;
    end loop;
    return false;
  end intersects;

end core_geometry;
```

Core Math

- pow2
- cos
- sin

Header

```
package core_math is

  type angle is mod 360;

  -- Compute the power of 2.
  -- i: integer to be squared.
  function pow2(i : integer) return integer
    with post => i * i = pow2'result;

  -- Compute the cosinus of an angle.
  -- i: the angle.
  function cos(i : integer) return float
    with post => cos'result <= 1.0 and then cos'result >= -1.0;
```

```
  -- Compute the sinus of an angle.
  -- i: the angle.
  function sin(i : integer) return float
    with post => sin'result <= 1.0 and then sin'result >= -1.0;

end core_math;
```

Implementation

```
package body core_math is

    function pow2(i : integer) return integer is
    begin
      return i * i;
    end pow2;

    function cos(i : integer) return float is
    mem : constant array (0 .. 90) of float := (
      1.0000, 0.9998, 0.9994, 0.9986, 0.9976, 0.9962, 0.9945, 0.9925, 0.9903,
        0.9877, 0.9848, 0.9816, 0.9781, 0.9744, 0.9703, 0.9659, 0.9613, 0.9563,
        0.9511, 0.9455, 0.9397, 0.9336, 0.9272, 0.9205, 0.9135, 0.9063, 0.8988,
        0.8910, 0.8829, 0.8746, 0.8660, 0.8572, 0.8480, 0.8387, 0.8290, 0.8192,
        0.8090, 0.7986, 0.7880, 0.7771, 0.7660, 0.7547, 0.7431, 0.7314, 0.7193,
        0.7071, 0.6947, 0.6820, 0.6691, 0.6561, 0.6428, 0.6293, 0.6157, 0.6018,
        0.5878, 0.5736, 0.5592, 0.5446, 0.5299, 0.5150, 0.5000, 0.4848, 0.4695,
        0.4540, 0.4384, 0.4226, 0.4067, 0.3907, 0.3746, 0.3584, 0.3420, 0.3256,
        0.3090, 0.2924, 0.2756, 0.2588, 0.2419, 0.2250, 0.2079, 0.1908, 0.1736,
        0.1564, 0.1392, 0.1219, 0.1045, 0.0872, 0.0698, 0.0523, 0.0349, 0.0175,
        0.0000
      );
  begin
    if i > 90 then
      return sin(90 - i);
    end if;

    if i < 0 then
      return cos(-i);
    end if;

    return mem(i);
  end cos;

  function sin(i : integer) return float is
    mem : constant array (0 .. 90) of float := (
      0.0000, 0.0175, 0.0349, 0.0523, 0.0698, 0.0872, 0.1045, 0.1219, 0.1392,
```

```
        0.1564, 0.1736, 0.1908, 0.2079, 0.2250, 0.2419, 0.2588, 0.2756, 0.2924,
        0.3090, 0.3256, 0.3420, 0.3584, 0.3746, 0.3907, 0.4067, 0.4226, 0.4384,
        0.4540, 0.4695, 0.4848, 0.5000, 0.5150, 0.5299, 0.5446, 0.5592, 0.5736,
        0.5878, 0.6018, 0.6157, 0.6293, 0.6428, 0.6561, 0.6691, 0.6820, 0.6947,
        0.7071, 0.7193, 0.7314, 0.7431, 0.7547, 0.7660, 0.7771, 0.7880, 0.7986,
        0.8090, 0.8192, 0.8290, 0.8387, 0.8480, 0.8572, 0.8660, 0.8746, 0.8829,
        0.8910, 0.8988, 0.9063, 0.9135, 0.9205, 0.9272, 0.9336, 0.9397, 0.9455,
        0.9511, 0.9563, 0.9613, 0.9659, 0.9703, 0.9744, 0.9781, 0.9816, 0.9848,
        0.9877, 0.9903, 0.9925, 0.9945, 0.9962, 0.9976, 0.9986, 0.9994, 0.9998,
        1.0000);
  begin
    if i > 90 then
      return cos(90 - i);
    end if;

    if i < 0 then
      return -sin(-i);
    end if;

    return mem(i);
  end sin;

end core_math;
```

Core Utils

- sleep
- rand

Header

```
package core_utils is

  -- delay until support.
  -- ms: time to sleep.
  procedure sleep (ms : natural);

  subtype rand_range is natural range 1 .. 100;

  -- pseudo-random (not at all, actually) support.
  function rand return rand_range;

private
```

```
  rand_count : rand_range := 1;

end core_utils;
```

Implementation

```
with Ada.Real_Time; use Ada.Real_Time;

package body core_utils is

  procedure sleep (ms : natural) is
    next_start : time := clock;
    period : constant time_span := milliseconds (ms);
  begin
    next_start := next_start + period;
    delay until next_start;
  end sleep;

  function rand return rand_range is
    mem : constant array (1 .. 100) of rand_range := (
      49, 28, 73, 59, 59, 33, 71, 58, 57, 85, 8, 72, 71, 68, 15, 39, 48, 67,
      17, 79, 5, 58, 6, 70, 78, 34, 3, 32, 54, 35, 35, 10, 96, 12, 18, 68, 27,
      64, 34, 34, 15, 59, 91, 40, 81, 52, 71, 4, 28, 60, 15, 31, 28, 82, 95,
      97, 87, 55, 54, 77, 69, 23, 48, 55, 79, 40, 58, 79, 79, 15, 6, 64, 18,
      35, 65, 5, 77, 39, 83, 75, 38, 71, 94, 22, 19, 24, 74, 88, 14, 33, 21,
      69, 3, 76, 5, 97, 49, 36, 4, 9
    );
  begin
    rand_count := rand_count + 1;
    return mem(mem(rand_count mod 100));
  end rand;

end core_utils;
```

Game

Game Ball

- “=”
- move
- slide_x
- slide_y
- draw

- update

Header

```
with screen_interface; use screen_interface;
with core_geometry; use core_geometry;
with core_math; use core_math;

package game_ball is

  type ball is
    record
      ci : circle;
      c : color;
      a : angle := 0;
      speed : float := 0.0;
    end record;

  -- Compare two balls based on their fields.
  -- b1: first ball.
  -- b2: second ball.
  function "=" (b1, b2 : ball) return boolean;

  -- Move the ball to a specific point.
  -- x: the coordinate of the x axis
  -- y: the coordinate of the y axis
  procedure move(b : in out ball; x, y : uint)
    with post => b.ci.x = x and then b.ci.y = y
    and then b.ci.r = b'old.ci.r;

  -- Slide the ball on the y axis by a delta.
  -- d: distance to slide by.
  procedure slide_y(b : in out ball; d : integer)
    with post => b'old.ci.y + d = b.ci.y
    and then b.ci.r = b'old.ci.r;

  -- Slide the ball on the x axis by a delta.
  -- d: distance to slide by.
  procedure slide_x(b : in out ball; d : integer)
    with post => b'old.ci.x + d = b.ci.x
    and then b.ci.r = b'old.ci.r;

  -- Draw the ball using its own color.
  -- b: the ball.
  procedure draw(b : ball);
```

```
  -- Update the ball position, angle, and speed based on the game rules.
  -- Returns the player that lost.
  -- b: the ball.
  type player_type is (the_enemy, the_player, none);
  function update(b : in out ball) return player_type
    with post => b'old.ci /= b.ci;

end game_ball;
```

Implementation

```
package body game_ball is

  function "=" (b1, b2 : ball) return boolean is
  begin
    -- FIXME: Check color.
    return b1.ci = b2.ci and then b1.a = b2.a;
  end "=";

  procedure move(b : in out ball; x, y : uint) is
  begin
    b.ci.x := x;
    b.ci.y := y;
  end move;

  procedure slide_y(b : in out ball; d : integer) is
  begin
    if b.ci.y + d <= 0 then
      b.ci.y := b.ci.r;
    elsif b.ci.y + d >= height'last then
      b.ci.y := height'last - b.ci.r;
    else
      b.ci.y := b.ci.y + d;
    end if;
  end slide_y;

  procedure slide_x(b : in out ball; d : integer) is
  begin
    if b.ci.x + d < 0 then
      b.ci.x := b.ci.r;
    elsif b.ci.x + d >= width'last then
      b.ci.x := width'last - b.ci.r;
    else
      b.ci.x := b.ci.x + d;
```

```
  end if;
end slide_x;

procedure draw(b : ball) is
begin
  draw(b.ci, b.c);
end draw;

function update(b : in out ball) return player_type is
  procedure check_bounds(b : in out ball; margin : uint) is
  begin
    if b.ci.x - b.ci.r < margin then
      b.ci.x := margin + b.ci.r;
      if sin(integer(b.a)) < 0.0 then
        b.a := b.a + 90;
      else
        b.a := b.a - 90;
      end if;
    elsif b.ci.x + b.ci.r > width'last - margin then
      b.ci.x := width'last - margin - b.ci.r;
      if sin(integer(b.a))  < 0.0 then
        b.a := b.a - 90;
      else
        b.a := b.a + 90;
      end if;
    end if;
  end check_bounds;

  function check_dead(b : ball) return player_type is
  begin
    if b.ci.y + b.ci.r >= height'last then
      return the_player;
    elsif b.ci.y - b.ci.r <= 0 then
      return the_enemy;
    else
      return none;
    end if;
  end check_dead;
  is_dead : player_type;
begin
  -- make the ball faster at each iteration
  b.speed := b.speed + 0.01;

  is_dead := check_dead(b);
  if is_dead /= none then
    return is_dead;
```

```
    end if;
    check_bounds(b, 10);

    slide_x(b, integer(b.speed * cos(integer(b.a))));
    slide_y(b, integer(b.speed * sin(integer(b.a))));

    is_dead := check_dead(b);
    if is_dead /= none then
      return is_dead;
    end if;
    check_bounds(b, 10);

    return none;
  end update;

end game_ball;
```

Game Player

- “=”
- move
- should_slide
- slide_x
- draw
- update_enemy
- update_user

Header

```
with screen_interface; use screen_interface;
with core_geometry; use core_geometry;
with game_ball; use game_ball;

package game_player is

  type player is
    record
      r : rectangle;
      c : color;
    end record;

  -- Compare two players based on their fields.
```

```
-- p1: first player.
-- p2: second player.
function "=" (p1, p2 : player) return boolean;

-- Move a player to a specific point.
-- p: the player.
-- x: coordinates on the x axis.
-- y: coordinates on the y axis.
procedure move(p : in out player; x, y : uint)
  with post => p.r.x = x and then p.r.y = y
  and then p.r.w = p'old.r.w
  and then p.r.h = p'old.r.h;

-- Decide if sliding the player is not going over the bounds.
-- p: the player.
---d: the displacement.
function should_slide(p : player; d : integer) return boolean;

-- Slide the player on the x axis.
-- p: the player.
-- d: delta displacement to be added.
procedure slide_x(p : in out player; d : integer)
  with post => p'old.r.x + d = p.r.x
  and then p.r.w = p'old.r.w
  and then p.r.h = p'old.r.h;

-- Draw a player on the screen with its own color.
-- p: the player.
procedure draw(p : player);

-- Draw a player on the screen.
-- p: the player.
-- c: the color.
procedure draw(p : player; c : color);

-- Update the enemy position based on the ball and the game rules.
-- p: the player (the enemy)
-- b: the main ball of the game.
procedure update_enemy(p : in out player; b : ball)
  with post => p.r.w = p'old.r.w and then p.r.h = p'old.r.h;

-- Update the player position based touch state.
-- p: the player (the user)
-- s: the state
procedure update_user(p : in out player; state : touch_state);
```

```
end game_player;
```

Implementation

```
package body game_player is

  function "=" (p1, p2 : player) return boolean is
  begin
    return p1.r = p2.r;
  end "=";

  procedure move(p : in out player; x, y : uint) is
  begin
    p.r.x := x;
    p.r.y := y;
  end move;

  function should_slide(p : player; d : integer) return boolean is
  begin
    if d >= 0 then
      return p.r.x + p.r.w + d < width'last;
    else
      return p.r.x + d > 0;
    end if;
  end should_slide;

  procedure slide_x(p : in out player; d : integer) is
  begin
    if p.r.x + d < 0 then
      p.r.x := 0;
    elsif p.r.x + p.r.w + d >= width'last then
      p.r.x := width'last - p.r.w;
    else
      p.r.x := p.r.x + d;
    end if;
  end slide_x;

  procedure draw(p : player) is
  begin
    draw(p, p.c);
  end draw;

  procedure draw(p : player; c : color) is
  begin
    draw(p.r, c);
```

```
  end draw;


  procedure update_enemy(p : in out player; b : ball) is
  begin
    if p.r.x /= b.ci.x
    and then b.ci.x - p.r.w > 0
    and then b.ci.x + p.r.w < width'last then
    draw(p, black);
    p.r.x := b.ci.x - p.r.w / 2;
    draw(p);
  end if;
  end update_enemy;

  procedure update_user(p : in out player; state : touch_state) is
  begin
    if state.x > width'last / 2 and should_slide(p, 10) then
      slide_x(p, 10);
    elsif state.x <= width'last / 2 and should_slide(p, -10) then
      slide_x(p, -10);
    end if;
    draw(p);
  end update_user;

end game_player;
```

Tests

Afin de générer les tests, il faut lancer `gnattest`:

```
$ arm-eabi-gnattest -P badass.gpr core/core_geometry.ads core/core_math.ads
                                  core/core_utils.ads game/game_ball.ads
                                  game/game_player.ads
```

Coverage

La couverture de code a été faite à l'aide de `gnatcov`.