TD/TP de Programmation – Correction de la Planche 2
Animation de Serpentin articulé

1 Fonctions élémentaires de calcul vectoriel

```
Coord
Coord_FromXY (double x, double y)
{
  Coord p;
  p.x= x;
  p.y= y;
  return p;
}

Coord
Util_MouseCoord (bx_mouse mouse)
{
  return Coord_FromXY (bx_mouse_x (mouse),
                       bx_mouse_y (mouse));
}
```

```
Coord
Coord_Sum (Coord p, Coord q)
{
  return Coord_FromXY (p.x + q.x,
                       p.y + q.y);
}

Coord
Coord_Difference (Coord p, Coord q)
{
  return Coord_FromXY (p.x - q.x,
                       p.y - q.y);
}
```

```
double
Coord_DotProduct (Coord p, Coord q)
{
  return p.x * q.x
    +    p.y * q.y;
}

double Coord_Length2 (Coord p) { return Coord_DotProduct (p, p); }
double Coord_Length  (Coord p) { return sqrt (Coord_Length2 (p)); }
```

```
Coord
Coord_ScaledBy (Coord p, double lambda)
{
  return Coord_FromXY (p.x * lambda,
                       p.y * lambda);
}

# define EPSILON 0.000001
bool Double_IsZero (double value) { return fabs (value) < EPSILON; }

Coord
Coord_Normalized (Coord p)
{
  double length= Coord_Length (p);
  return Double_IsZero (length) ?
    Coord_FromXY (1.0, 0.0) :
    Coord_ScaledBy (p, 1.0/length);
}
```

```
Coord
Coord_MovedAtDistanceFrom (Coord p, double distance, Coord target)
{
  Coord vector= Coord_Difference (p, target);
  Coord unit_vector= Coord_Normalized (vector);
  Coord scaled_vector= Coord_ScaledBy (unit_vector, distance);
  return Coord_Sum (target, scaled_vector);
}
```

2 Test des fonctions élémentaires de calcul vectoriel

```
double
Double_Round (double x)
{
  double f= floor (x);
  return (x-f <= 0.5) ? f : f+1.0;
}
```

```
void
Util_DrawLine (bx_window window, Coord p, Coord q)
{
  bx_draw_line (window,
                Double_Round (p.x), Double_Round (p.y),
                Double_Round (q.x), Double_Round (q.y));
}
```

```
void
Util_DrawCircle (bx_window window, Coord center, double radius, int filled)
{
  bx_draw_circle (window,
                  Double_Round (center.x - radius),
                  Double_Round (center.y - radius),
                  2*radius, 2*radius, filled);
}
```

```
int
MainTest_CoordDemo (void)
{
  bx_init ();
  Coord dim= Coord_FromXY (200, 100);
  bx_window window= bx_create_window ("test", 0, 0, dim.x, dim.y);
  for (;;)  {
    Coord center= Coord_ScaledBy (dim, 0.5);
    bx_mouse mouse= bx_read_mouse (window);
    Coord cursor= Util_MouseCoord (mouse);
    double distance= 50, radius= 5;
    Coord point= Coord_MovedAtDistanceFrom (cursor, distance, center);

    bx_clear_canvas (window, bx_white ());
    bx_set_color (bx_black()); Util_DrawLine   (window, center, cursor);
    bx_set_color (bx_red  ()); Util_DrawCircle (window, center, radius, 1);
    bx_set_color (bx_blue ()); Util_DrawCircle (window, cursor, radius, 1);
    bx_set_color (bx_black()); Util_DrawCircle (window, point , radius, 0);
    bx_show_canvas (window, 10);
  }
  bx_loop ();
  return 0;
}
```

3 Initialisation du serpentin

```
void
Joint_Init (Joint * j, Coord center, double radius)
{
  j->center= center;
  j->radius= radius;
}
```

```
double Double_Lerp (double x0, double alpha, double x1)
{
  return x0 + alpha * (x1-x0);
}

void
Snake_Init (Snake * s, int nb_joints, Coord head_center,
            double head_radius, double tail_radius)
{
  s->nb_joints= (nb_joints <= MAX_JOINTS) ? nb_joints : MAX_JOINTS;
  Joint_Init (& s->joints [0], head_center, head_radius);
  for (int k= 1; k < nb_joints; k++) {
    double lerp= k / (nb_joints - 1.0);
    double radius= Double_Lerp (head_radius, lerp, tail_radius);
    double distance= radius + s->joints[k-1].radius;
    Coord shift=  Coord_FromXY (distance, 0);
    Coord center= Coord_Sum (s->joints [k-1].center, shift);
    Joint_Init (&  s->joints [k], center, radius);
  }
}
```

4 Affichage du serpentin

```
void
Joint_Draw (Joint const * j, bx_window window)
{
  bx_set_color (bx_red  ()); Util_DrawCircle (window, j->center, j->radius, 1);
  bx_set_color (bx_black()); Util_DrawCircle (window, j->center, j->radius, 0);
}
```

```
void
Snake_Draw (Snake const * s, bx_window window)
{
  for (int k= 0; k < s->nb_joints; k++) {
    Joint const * current= & s->joints [k];
    Joint_Draw (current, window);
  }
}
```

```
int
MainTest_SnakeDrawingDemo (void)
{
  bx_init ();
  bx_window window= bx_create_window ("Snake", 50, 50, 800, 800);
  Snake snake;
  Snake_Init (& snake, 50, Coord_FromXY (50,50), 10, 5);
  bx_clear_canvas (window, bx_white ());
  Snake_Draw (& snake, window);
  bx_show_canvas (window, 10);
  bx_loop ();
  return 0;
}
```

5 Déplacement et animation du serpentin

```
void
Joint_MoveTowards (Joint * j, Joint const * target)
{
  double distance = j->radius + target->radius;
  j->center= Coord_MovedAtDistanceFrom (j->center, distance, target->center);
}
```

```
void
Snake_MoveHeadTo (Snake * s, Coord head_center)
{
  s->joints [0].center= head_center;
  for (int k= 1; k < s->nb_joints; k++) {
    Joint * j0= & s->joints [k-0];
    Joint * j1= & s->joints [k-1];
    Joint_MoveTowards (j0, j1);
  }
}
```

```
int
MainTest_SnakeAnimationDemo (void)
{
  bx_init ();
  bx_window window= bx_create_window ("Snake Demo", 50, 50, 800, 800);
  Snake snake;
  Snake_Init (& snake, 50, Coord_FromXY (50,50), 10, 5);
  for (;;)  {
    bx_mouse mouse= bx_read_mouse (window);
    Coord cursor= Util_MouseCoord (mouse);
    Snake_MoveHeadTo (& snake, cursor);
    bx_clear_canvas (window, bx_white ());
    Snake_Draw (& snake, window);
    bx_show_canvas (window, 10);
  }
  bx_loop ();
  return 0;
}
```