A Case Study in OCaml

Implementing a Particle Simulator, OParticles

Prepared for COMP6411,
and presented to Dr. Constantinides

Simon J. Symeonidis
5887887

December 1, 2013

Contents

Listings

::Section 1::

Summary

This is my coursework for SOEN6411, on the OCaml programming language. Inside this part we discus the different powers of the programming language, the flexibilities, and other shortcommings. The application in question is a particle simulator. The particle simulator defines two aspects: the bodies that exist in the simulation, and possible obstructions.

The obstructions are either the bodies themselves, or constraints in the form of walls. The bodies are programmed to have different velocities, acceleration, and gravitational attraction. Calculations are made to find the 3d angle to which they travel when they bounce off such surfaces.

The application was initially intended to bind to *OpenGl* and show the particles bounce off walls. However locating libraries which were not out of date and that actually worked was extremely difficult. Therefore the graphical aspect was scoped out in this project, and simple text output was provided. Even so, the case study was not particularly easy.

::Section 2::

Language Exploration

In this section we show some of the language features providede by OCaml.

2.1 Interface and Implementation

In OCaml, we are able to separate implementation from interfaces. For example, when implement- ing the Particle class, we need to define the modules signatures in one file with *.mli extension, and the specify the implementation in the respective *.ml file. This is very similar to how you would define header files in C/C++ programs. You can refer to Listings 12, and 11 to see the implementations and specifications respectively, of the particle class.

2.2 Modules

Modules act as namespaces. Functions that are defined inside a module definition can be accessed from other parts of the OCaml application. If they are omitted from the module interface, then the methods are not visible and available for local use of the module. The module must have the same name as the file. To access another module, one needs to do this by adding *open ModuleName* at the file of interest. Listing 7 shows a module signature (interface), and 8 shows the implementation of the specification.

2.3 Types

There exists the basic types in OCaml such as integers, floats, booleans, strings, and lists. Here is a small list of possible operations on each type listed above:

- Integers
  - 1 + 1 returns 2
  - 1 - 1 returns 0
  - 5 * 2 returns 10
  - 10 / 3 returns 3 (of type integer)
- Floats *(notice that each operation requires a period after each symbol).*
  - 1.0 +. 2.0 returns 3.0
  - 1.0 /. 3.0 returns 0.33333333333333315
  - 1.0 *. 4.0 returns 4.0

  - 1.0 -. 3.0 returns -2.0

- Booleans
  - true && true returns true
  - true || false returns true

- Lists
  - 1 :: [2; 3; 4];; returns [1; 2; 3; 4]
  - List.hd [1; 2; 3; 4];; returns 1
  - List.tl [1; 2; 3; 4];; returns [2; 3; 4]

2.4 Tuples

Tuples are supported in Ocaml. In order to define a signature for a tuple, we chain the types we want the tuple to contain with the * symbol. For example if we wanted a tuple with the first two types of integer and the last one of type float we would define it as such: *int * int * float*. A practictal example where this has been used in specification is in Listing 13, and its implementation in 14.

2.5 Object Orientation

We are able to define classes in OCaml. To do this, we need to define the interface (mli) file, and then implement it in the implementation file (ml). Let us take the class Particle for this example. We need it to have an interface that requires the implementation of features: mass, velocity, acceleration, group, label, angle-xy, angle-xz, and its coordinates. We also need mutators for these attributes. Listing 11 demonstrates the interface, and Listing 12 demonstrates the implementation of the specification.

It should be noted that if the class needs to call its own methods, then in the implementation file, we need to specify object(self) as opposed to simply object. By doing this, we publish a class pointer within the instance of the class, and can call methods through it (exactly like this). A quick example of how this is used can be viewed in Listing 12, in the *method_start()* where the instance calls its own method via the expression: *self # start_backend()*.

Something to observe is that when defining classes, we need to use the keyword *method* in order to define class methods, as opposed to using *let*.

Another minor note is that method visibility is possible by the reserved word *private*. The order of the keyword 'private' is important, else we get a compilation error. An example of this can be see inside the room class, in listing 4, inside the *start_backend()* method.

2.6 Recursion

OCaml supports recursion. However we need to explicitly specify when this is the case, by adding a rec token after our let token. Theres two techniques to invoke recursion. In a module, recursion is achieved by simply writing the function in the form of: *let rec funcname args = conditional* $\langle true \rangle$ *base-case*$^{+}\langle false \rangle$ $\{$*recursive-case*$\}^{+}$. Listing 6 shows us this, notably in function create list.

To define such behavior in a class instance, we need to define a function inside a function. The inner function is the recursive function - the outer one is simply the interfacing one. Class room (Listing 4) shows such implementation details in the function *start_backend()*.

::Section 3::

Sample Output

Listing 1 gives us some sample output. These are the last few ticks inside the simulation. The values of mass, velocity, etc, are shown inside the square brackets. The coordinates x, y, z are shown inside the parentheses.

```
...
Prtcl
  [label:random group:1]
  [v:1.56707442303 a:0.238954590183 m:0.486273268246]
      (-0.0484577870752, -4028.06566051, 27.8801124281)
Prtcl
  [label:random group:2]
  [v:0.682371147812 a:0.227468348484 m:0.964356786583]
      (23.8516145732, 96.3937503721, 0.30062667424)
Prtcl
  [label:random group:2]
  [v:1.30226236561 a:0.563336235183 m:0.999592035115]
      (76.0842753459, 7.31537329423, 96.1287439091)
Prtcl
  [label:random group:2]
  [v:1.33392129413 a:0.0433985028577 m:0.0619103976662]
      (3791.94409484, 0.0310810268212, 41.5974427329)
Tick # 10000
Prtcl
  [label:random group:2]
  [v:1.03083833615 a:0.619287510227 m:0.729364370812]
      (81.7329550393, 66.0174246037, 38.3580181093)
Prtcl
  [label:random group:0]
  [v:0.339718489261 a:0.8050239962 m:0.380702238651]
      (73.0486679625, 97.9470109765, 60.8922724826)
Prtcl
  [label:random group:1]
  [v:0.560038017839 a:0.698211950748 m:0.55290552373]
      (80.6884587702, 16.345910984, 90.083734737)
Prtcl
  [label:random group:2]
  [v:1.78017150687 a:0.926870430927 m:0.78312921776]
      (77.5797662059, 45.5525293714, 60.5138270725)
Prtcl
  [label:random group:1]
  [v:1.78885795012 a:0.386322010764 m:0.403095970641]
      (1.60668443534, -5845.42755734, 70.0240322108)
Prtcl
  [label:random group:0]
  [v:1.17709744077 a:0.938833426632 m:0.839278626693]
      (1.48047009239, -2941.46462376, 9.97592488942)
Prtcl
  [label:random group:1]
  [v:1.56707442303 a:0.238954590183 m:0.486273268246]
      (1.40982524723, -4028.63936151, 28.8461139563)
Prtcl
  [label:random group:2]
  [v:0.682371147812 a:0.227468348484 m:0.964356786583]
      (23.7850157282, 95.7146369963, 0.105918472598)
Prtcl
  [label:random group:2]
  [v:1.30226236561 a:0.563336235183 m:0.999592035115]
      (77.2280481421, 7.93800565732, 95.9147176169)
Prtcl
  [label:random group:2]
  [v:1.33392129413 a:0.0433985028577 m:0.0619103976662]
      (3791.85964188, -1.3001641484, 42.5950488875)
```

```
Tick # 10001
Prtcl
  [label:random group:2]
  [v:1.03083833615 a:0.619287510227 m:0.729364370812]
      (82.3260761121, 66.8605349572, 38.1323827086)
Prtcl
  [label:random group:0]
  [v:0.339718489261 a:0.8050239962 m:0.380702238651]
      (72.7247827754, 97.8445071352, 60.6205950144)
Prtcl
  [label:random group:1]
  [v:0.560038017839 a:0.698211950748 m:0.55290552373]
      (80.1403394954, 16.2309856081, 89.6687396747)
Prtcl
  [label:random group:2]
  [v:1.78017150687 a:0.926870430927 m:0.78312921776]
      (78.5364542653, 47.0537816989, 61.4289395305)
Prtcl
  [label:random group:1]
  [v:1.78885795012 a:0.386322010764 m:0.403095970641]
      (1.57049763497, -5847.21604924, 68.8084779199)
Prtcl
  [label:random group:0]
  [v:1.17709744077 a:0.938833426632 m:0.839278626693]
      (0.971806122968, -2942.52614123, 9.14579223303)
Prtcl
  [label:random group:1]
  [v:1.56707442303 a:0.238954590183 m:0.486273268246]
      (0.836124245677, -4030.09764455, 29.8121154845)
Prtcl
  [label:random group:2]
  [v:0.682371147812 a:0.227468348484 m:0.964356786583]
      (23.7184168831, 95.0355236205, -0.0887897290429)
Prtcl
  [label:random group:2]
  [v:1.30226236561 a:0.563336235183 m:0.999592035115]
      (78.3718209383, 8.56063802041, 95.7006913247)
Prtcl
  [label:random group:2]
  [v:1.33392129413 a:0.0433985028577 m:0.0619103976662]
      (3791.94409484, 0.031081026821, 43.5926550421)
```

Listing 1: Output of Oparticles

::Section 4::

Code

This section lists all the required code to implement the solution.

```
(* Main entry point! *)

open Room;;
open Coordinate;;

let _ = Random.self_init()
let simulation = new Room.room

let main =
  simulation # init();
  simulation # start();;
```

Listing 2: Main Entry Point

```
(*
 * This is a container of all the particles. This object should define
 * the bounds in which the particles exist.
 *
 * @author Simon Symeonidis
 *)

open Coordinate
open Particle
open ParticleManager
open ParticleBuilder;;

class room :
  object
    val bounds : Coordinate.coordinate
    val mutable particles : Particle.particle list
    val maximum_ticks : int
    val mutable current_tick  : int

    method init  : unit -> unit
    method tick  : unit -> unit
    method start : unit -> unit
    method print : unit -> unit

  end;;
```

Listing 3: Room Interface

```
(*
 * This contains particles, and should take care of the function ticks
 *
```

```
 4  * @author Simon Symeonidis
 5  *)
 6
 7 open Coordinate
 8 open Particle
 9 open ParticleManager
10 open ParticleBuilder;;
11
12 class room =
13   object(self)
14     (* Bounds of the room *)
15     val bounds = (100.0, 100.0, 100.0)
16
17     (* Maximum ticks of the simulation *)
18     val maximum_ticks = 10_000
19
20     (* Current tick in the current room instance *)
21     val mutable current_tick  = 0
22
23     (* The particles that exist in the room, simulation *)
24     val mutable particles : Particle.particle list = [];
25
26     (* Initialize with the list of particles *)
27     method init () = particles <- ParticleBuilder.create_list 10
28
29     (* Start the simulation *)
30     method start () = self # start_backend()
31
32     (* A tick in the simulation, with a single step *)
33     method tick () =
34       current_tick <- current_tick + 1;
35       ParticleManager.tick particles;
36       ParticleManager.collision_check particles bounds;
37       print_endline ("Tick # " ^ string_of_int current_tick);
38
39     (* Print all the information in the room - essentially what happens in
40        the builder *)
41     method print () = ParticleBuilder.print_list particles
42
43     (* Recursive function that prints out all the particles, and makes them
44        interact in the system by a step *)
45     method private start_backend () =
46       let rec tick_tock it =
47         self # tick ();
48         self # print ();
49         if it > 0 then tick_tock (it - 1) else ()
50       in tick_tock maximum_ticks;
51   end;;
```

Listing 4: Room Implementation

```
1 (*
2  * Particle builder, following the builder pattern, and providing functions
3  * that can help us out a little.
4  *
```

```
 5  * @author Simon Symeonidis
 6  *)
 7
 8 open Particle;;
 9
10 module ParticleBuilder : sig
11   val create_particle : unit -> Particle.particle
12   val create_jonny    : unit -> Particle.particle
13   val create_random   : unit -> Particle.particle
14   val create_list     : int  -> Particle.particle list
15
16   val print_list      : Particle.particle list -> unit
17
18 end;;
```

Listing 5: Particle Builder Interface

```
 1 open Physics
 2 open Particle
 3 open Random;;
 4
 5 (*
 6  * Implementation of the particle builder. This provides some static
 7  * creational patterns for creating particles.
 8  *
 9  * @author Simon Symeonidis
10  *)
11 module ParticleBuilder = struct
12   (* Shorthand for a random float bounded by 'f' *)
13   let rfloat f = Random.float f;;
14
15   (* Shorthand for a random int bounded by 'i' *)
16   let rint   i = Random.int i;;
17
18   (* Create a new particle *)
19   let create_particle () = new Particle.particle;;
20
21   (* This is just a function that creates a particle with some predefined and
22      hardcoded values. This is here mainly because I was fiddling around with
23      the language *)
24   let create_jonny () =
25     let n = new Particle.particle in
26       n # set_mass         0.2;
27       n # set_label        "jonny";
28       n # set_velocity     0.3;
29       n # set_acceleration 0.0;
30       n # set_group        0;
31       n # set_angle_xz     0.0;
32       n # set_angle_xy     0.0;
33     n;;
34
35   (* Create a fully randomized particle *)
36   let create_random () =
37     let n = new Particle.particle in
38       n # set_mass         (rfloat 1.0);
```

```
39        n # set_label         "random";
40        n # set_velocity      (rfloat 2.0);
41        n # set_acceleration  (rfloat 1.0);
42        n # set_group         (rint 4);
43        n # set_angle_xz      (rfloat Physics.pi);
44        n # set_angle_xy      (rfloat Physics.pi);
45      n;;
46
47    (* Recursive function to create a list of randomized particles *)
48    let rec create_list amnt =
49      if amnt = 0 then [] else create_random() :: create_list (amnt - 1);;
50
51    (* Recursive function to print out some particles *)
52    let rec print_list plist =
53      if plist = [] then ()
54      else begin
55        print_endline ((List.hd plist) # to_string);
56        print_list (List.tl plist);
57      end;;
58
59  end;; (* module ParticleBuilder *)
```

Listing 6: Particle Builder Implementation

```
1  (*
2   * Simple coordinate helper so that we can move things, and check if particles
3   * have hit a wall / are in some bounds that they are not supposed to
4   * @author Simon Symeonidis
5   *)
6
7  open Coordinate;;
8
9  module CoordinateHelper : sig
10     val hit_wall : Coordinate.coordinate -> Coordinate.coordinate -> bool
11     val fs : Coordinate.coordinate -> float
12     val sc : Coordinate.coordinate -> float
13     val th : Coordinate.coordinate -> float
14
15     val set_fs : 'a -> 'a * 'a * 'a -> 'a * 'a * 'a
16     val set_sc : 'a -> 'a * 'a * 'a -> 'a * 'a * 'a
17     val set_th : 'a -> 'a * 'a * 'a -> 'a * 'a * 'a
18
19     val add_fs : float -> Coordinate.coordinate -> Coordinate.coordinate
20     val add_sc : float -> Coordinate.coordinate -> Coordinate.coordinate
21     val add_th : float -> Coordinate.coordinate -> Coordinate.coordinate
22
23     val to_string : Coordinate.coordinate -> string
24   end;;
```

Listing 7: Coordinate Helper Interface

```
1  (* Small helper to provide common functionalities for Coordinate tuples
```

```
 2  * @author Simon Symeonidis *)
 3
 4 open String
 5 open Coordinate;;
 6
 7 module CoordinateHelper = struct
 8     let hit_wall(px,py,pz) (wx,wy,wz) = true
 9
10     let fs (x,_,_) = x
11     let sc (_,y,_) = y
12     let th (_,_,z) = z
13
14     let set_fs x (_,y,z) = (x,y,z)
15     let set_sc y (x,_,z) = (x,y,z)
16     let set_th z (x,y,_) = (x,y,z)
17
18     let add_fs a (x,y,z) = (a+.x,y,z)
19     let add_sc a (x,y,z) = (x,y+.a,z)
20     let add_th a (x,y,z) = (x,y,z+.a)
21
22     let to_string c = "(" ^ string_of_float (fs c)
23       ^ ", " ^ string_of_float (sc c)
24       ^ ", " ^ string_of_float (th c) ^ ")"
25
26   end;;
```

Listing 8: Coordinate Helper Implementation

```
1 (* Particle Manager *)
2
3 open Coordinate;;
4
5 module ParticleManager : sig
6   val tick : Particle.particle list -> unit
7   val collision_check : Particle.particle list -> Coordinate.coordinate -> unit
8 end;;
```

Listing 9: Particle Manager Interface

```
 1 (*
 2  * The particle manager is supposed to handle all the actions that are to
 3  * be performed on the particles on a tick of the simulation.
 4  * @author Simon Symeonidis
 5  *)
 6
 7 open CoordinateHelper
 8 open Physics;;
 9
10 module ParticleManager = struct
11   (* Iterate each particle, and apply its tick *)
12   let rec tick particles =
13     if particles = [] then () else begin
```

```
14        Physics.apply (List.hd particles);
15        tick (List.tl particles);
16      end;;
17
18    (* Bounce against X wall *)
19    let bounce_x b p =
20      if p # get_x > CoordinateHelper.fs b || p # get_x < 0.0
21      then p # set_angle_xy (p # get_angle_xy +. Physics.half_pi)
22      else ();;
23
24    (* Bounce against Y wall *)
25    let bounce_z b p =
26      if p # get_z > CoordinateHelper.th b || p # get_z < 0.0
27      then p # set_angle_xz (p # get_angle_xz +. Physics.half_pi)
28      else ();;
29
30    (* Bounce against Z wall *)
31    let bounce_y b p =
32      if p # get_y > CoordinateHelper.sc b || p # get_y < 0.0
33      then p # set_angle_xy (p # get_angle_xy +. Physics.half_pi)
34      else ();;
35
36    let bounce bounds particle =
37      bounce_x bounds particle;
38      bounce_y bounds particle;
39      bounce_z bounds particle;;
40
41    (* Check collisions with the wall *)
42    let rec collision_check particles bounds =
43      if particles = [] then () else begin
44        bounce bounds (List.hd particles);
45        collision_check (List.tl particles) bounds;
46      end;;
47  end;;
```

Listing 10: Particle Manager Implementation

```
1  (* @author Simon Symeonidis
2   * Particle class for the bodies that are to be simulated *)
3
4  open Coordinate;;
5
6  class particle :
7    object
8      val mutable mass         : float
9      val mutable velocity     : float
10     val mutable acceleration : float
11     val mutable group        : int
12     val mutable label        : string
13     val mutable angle_xy     : float
14     val mutable angle_xz     : float
15     val mutable coord        : Coordinate.coordinate
16
17     method get_mass         : float
18     method get_velocity     : float
```

```
19      method get_acceleration : float
20      method get_group        : int
21      method get_label        : string
22      method get_angle_xz     : float
23      method get_angle_xy     : float
24      method get_x            : float
25      method get_y            : float
26      method get_z            : float
27
28      method set_mass         : float  -> unit
29      method set_velocity     : float  -> unit
30      method set_acceleration : float  -> unit
31      method set_group        : int    -> unit
32      method set_label        : string -> unit
33      method set_angle_xy     : float  -> unit
34      method set_angle_xz     : float  -> unit
35
36      method move_x : float -> unit
37      method move_y : float -> unit
38      method move_z : float -> unit
39
40      method to_string : string
41
42    end;;
```

Listing 11: Particle Interface

```
1  (* Particle object that contains data and behaviour of the floating bodies
2   * of the simulation.
3   * @author Simon Symeonidis
4   *)
5
6  open CoordinateHelper
7  open String;;
8
9  class particle =
10   object (self)
11     val mutable mass         = 0.0
12     val mutable velocity     = 0.0
13     val mutable acceleration = 0.0
14     val mutable group        = 0
15     val mutable label        = ""
16     val mutable angle_xz     = 0.0
17     val mutable angle_xy     = 0.0
18     val mutable coord        = (0.0 , 0.0 , 0.0)
19
20     method get_mass = mass
21     method get_velocity = velocity
22     method get_acceleration = acceleration
23     method get_group = group
24     method get_label = label
25     method get_angle_xz = angle_xz
26     method get_angle_xy = angle_xy
27     method get_x = CoordinateHelper.fs coord
28     method get_y = CoordinateHelper.sc coord
```

```
29      method get_z = CoordinateHelper.th coord
30
31      method set_mass i_mass        = mass <- i_mass
32      method set_velocity i_vel     = velocity <- i_vel
33      method set_acceleration i_acc = acceleration <- i_acc
34      method set_group i_g          = group <- i_g
35      method set_label i_l          = label <- i_l
36      method set_angle_xy i_a       = angle_xy <- i_a
37      method set_angle_xz i_a       = angle_xz <- i_a
38
39      method move_x x = coord <- CoordinateHelper.add_fs x coord
40      method move_y y = coord <- CoordinateHelper.add_sc y coord
41      method move_z z = coord <- CoordinateHelper.add_th z coord
42
43      (* Mainly for printing the particle information on screen *)
44      method to_string =
45        "Prtcl [label:" ^ label ^ " "
46        ^ "group:" ^ (string_of_int group) ^ "] "
47        ^ "[v:" ^ (string_of_float velocity)
48        ^ " a:" ^ (string_of_float acceleration)
49        ^ " m:" ^ (string_of_float mass)
50        ^ "] " ^ CoordinateHelper.to_string coord
51    end;;
```

Listing 12: Particle Implementation

```
1 (* Coordinate data type
2  * @author Simon Symeonidis
3  *)
4
5 module Coordinate : sig
6     type coordinate  = float * float * float
7     type coordinate2 = float * float
8   end;;
```

Listing 13: Coordinate Interface

```
1 (* Coordinate implementation *)
2
3 module Coordinate = struct
4     type coordinate  = float * float * float
5     type coordinate2 = float * float
6   end;;
```

Listing 14: Coordinate Implementation

```
1 (* @author Simon Symeonidis *)
2
```

```
 3 open Particle;;
 4 open CoordinateHelper;;
 5
 6 module Physics : sig
 7   val v1 : float -> float -> float -> float
 8   val s1 : float -> float -> float -> float
 9   val s2 : float -> float -> float -> float
10   val s3 : float -> float -> float -> float
11   val v2 : float -> float -> float -> float
12
13   val apply : Particle.particle -> unit
14
15   val pi : float
16   val half_pi : float
17   val vecx : float -> float -> float
18   val vecy : float -> float -> float
19   val vecz : float -> float -> float
20 end;;
```

Listing 15: Physics Interface

```
 1 (* Some elementary physics functions that we use in order to make particles
 2  * move around
 3  *
 4  * @author Simon Symeonidis *)
 5
 6 open CoordinateHelper
 7 open Particle;;
 8
 9 module Physics = struct
10   (* SUVAT Equations *)
11
12   let v1 u a t = u +. a *. t;;
13   let v2 u a s = sqrt((u *. u) +. 2.0 *. a *. s)
14
15   let s1 u a t = u *. t +. 0.5 *. a *. t *. t;;
16   let s2 u v t = 0.5 *. (u +. v) *. t;;
17   let s3 u a t = u *. t -. 0.5 *. a *. t *. t;;
18
19   (* Thanks to
20    * http://caml.inria.fr/mantis/view.php?id=5173 *)
21   let pi = 4. *. atan 1.
22
23   (* quickhand *)
24   let half_pi = pi /. 2.
25
26   (* Get the x vector value from a 3d vec *)
27   let vecx theta v = cos(pi /. 2. -. theta) *. v
28
29   (* Get the y vector value from a 3d vec *)
30   let vecy theta v = cos(theta) *. v
31
32   (* Get the z vector value from a 3d vec *)
33   let vecz theta v = sin(theta) *. v
34
```

```
35   (* Apply the physics on one particle *)
36   let apply particle =
37     let vel = particle # get_velocity in
38     let angle_xy = particle # get_angle_xy in
39     let angle_xz = particle # get_angle_xz in
40     particle # move_x (vecx angle_xy vel);
41     particle # move_y (vecy angle_xy vel);
42     particle # move_z (vecz angle_xz vel);;
43 end;;
```

Listing 16: Physics Implementation