Università degli Studi di Pisa

Laurea Specialistica in Informatica

Advanced Programming

Final Term Paper

Start Date: 31/01/2015
Submission deadline: 20/02/2015 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- **to consult or setup an online forum, to request help of consultants in producing the paper**
- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone's code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C# and Java.

The paper must:

1. be in a single PDF file, formatted readably (**font size ≥ 10 pt** with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include the student name
3. provide the solution and the code for each exercise separately, referring to the code of other exercises if necessary.
4. cite references to literature or Web pages from where information was taken.

Introduction

In this project, you will develop a DSL for game development, called GSL (Game Specification Language). Here is an example of the description of a simplified Space Invaders game (https://en.wikipedia.org/wiki/Space_Invaders), where a cannon is moved horizontally and shoots at a horde of descending aliens. The player controls the cannon with the keyboard arrows and shoots a laser beam by pressing the space bar. If the beam reaches an alien it is destroyed, while if an alien lands on the bottom, the game is lost.

For simplicity assume there is a single row of aliens, aliens do not drop bombs and there are no defense bunkers as in the original game.

```
<Game>
   <State> <Start/> <Play/> <Win/> <Lose/> </State>
   <Control>
      <Keyboard> <Left/> <Right/> <Space/> </Keyboard>
   </Control>
```

```
      <Rule> <Destruction/> </Rule>
      <Scene>
         <Transition> <MoveAliens/> <MoveBeam/> </Transition>
         <Graphics> <BattleField/> </Graphics>
         <Events>
            <ShootBeam/>
            <MoveCannonLeft/>
            <MoveCannonRight/>
            <AlienShooted/>
            <AliensLanding/>
         </Events>
      </Scene>
   </Game>
```

The GSL is used to generate code for the game, by means of XSLT transformations. Just for illustrative purposes, here are some fragments of transformations that use JavaScript as target implementation language:

```
   <xsl:template match="/Game">
     var game = new Game();
     var currentState = new Start();
     window.addEventListener("keydown", function keydown(evt) {
       var keycode = evt.which || window.event.keycode;
       game.keyDown(keycode);
     }
     setInterval(function() { game.step(); }, 16);     // timer
     …
   </xslt:template>

   <xsl:template match="/Game/State/Win">
     Win.prototype = new GameState();      // inheritance
     Win.prototype.enter = function(session, params) {
         // session is a GameSession
         session.setStatus("Win");
     }
     Win.prototype.draw(…) { … }
   </xsl:template>

   <xsl:template match="/Game/Control/Keyboard/Space">
      if (keycode == 32)) // space
         game.session.fire(new ShootBeam());
   </xsl:template>
```

Exercise 1

Design a set of classes to represent the syntax of GSL, a subset of XML without attributes and GSLT transformations, a minimal subset of XSLT, whose tags only use a single attribute “match”.

Exercise 2

Implement a recursive descent parser for GSL and GSLT without using external libraries or parser generators. Split the parser into a lexical analyzer, recognizing as tokens just tags and text, and a syntax analyzer, as presented in the slides of the course.

Exercise 3

Design a set of classes to represent a game, including one for each of the element in GSL, and provide methods that implement a generic game mechanism, for example initialization, event firing, state transition and rule application. Explain the role of each class, for example, by a state transition diagram.

Exercise 4

Write a minimal XSL transformer, as a method on GSL trees, to which a set of XSLT transformations is passed, and which applies the proper transformation to that GSL tree, in order to produce the code for running the game.

Exercise 5

Build a full Space Invaders game, completing the GSL and the GSLT of the introduction.

Exercise 6

Describe the technique of tracing garbage collection. Explain which information the garbage collector needs to know about the runtime data structures and the program data structures in order to perform its task. Explain how the collector can obtain such information. Explain why it is difficult to design a collector that could work on code produced by compilers for different programming languages. Discuss any relation or similarity between the information required by a garbage collection and that provided by reflection facilities.