XML for Video Games

CS 297 Report

Ru Chen

Advisor: Dr. Chris Pollett
Department of Computer Science
San Jose State University

Dec 2005

Introduction:

When designing a video game, a game designer will think of the main characters and actions of the game. All the details of the game will be listed in a specification file. The specification file describes the look and feel of the game's characters, maps of levels and environments, background, rules of the game, content, behavior, etc. The game developer will then put all these concepts together and implement it into graphics. Game design is long and complicated process. My project proposed a XML language that can help game designers and developers design and test video games.

XML (Extensible Markup language) is a standardized text format for representing structured data. XML has many great features that make it ideal for game design. One of them is its extensibility. Unlike the other markup language HTML, XML allows us to define our own tags and our own document structure. So it is very flexible when it comes to defining our game components using XML tags. Also, XML is not tied to any programming languages or any operating system and it is quite straightforward to parse XML files using many programming languages. And last, since it is a plain text file, XML can be read and edited using any text-editing tool. Creating and testing games will be very easy. One can simply edit the XML file for the game and test the effects.

The main purpose of this project is to define a XML language for video games, which will be helpful for game designers and developers to create and generate video games. This XML language will allow one to define the initial setup of a game such as the name and version of the game, resource type and directory, environment and background, as well as properties and attributes of a game object such as its shape, size, location, color, what kind of physical forces (gravity, spring, friction) apply to it, and what game rules its AI uses. The XML language will also specify what kind of input devices (for instance, does the game controller need force-feedback), what kind of sound card features, and what kind of media play features are needed for the game. Using this XML language, one can define a set of tags to represent all those game elements.

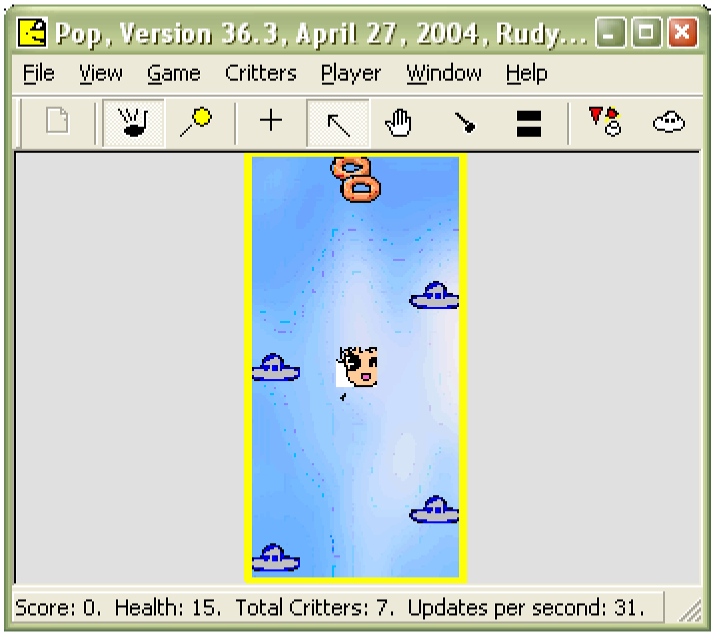
The following introduces four deliverables I have accomplished for CS297. Section 2 introduces the Gauntlet game. Section 3 talks about a family tree XML file and a Java family tree parser. Section 4 is a XML file that describes the Centipede game that was a very popular game in the 80s. Section 5 introduces a simplified game engine to implement the centipede game XML file. And finally section 6 gives a brief overview of my future work.

Section 2: Deliverable One:

2.1 Goal

The goal of deliverable one was to develop a new game using Rudy Rucker’s pop framework. The game is called Gauntlet. The game has one player and the player’s goal is to eat as many as doughnuts as possible before the aliens at the side of the screen shoot the player to death. The player cannot shoot but he can move up and down. The player's motion is constrained to a vertical band in the middle of the screen and is controllable using the up and down arrow keys. The doughnuts will appear at either the top of the screen or the bottom of the screen. The player “eats” a doughnut when he bumps into it.

The initial health of the player is 20. The player gets 100 points every time he eats a doughnut and his health will go up by 2. The aliens cannot move but they try to shoot at the player as he moves up and down. Hitting the player will takes away some of the player's health. Initially, the damage of a bullet is one and it increases by one every time the player eats a doughnut. The program was written and complied using Visual C++. The following is a screen shot of the Gauntlet game.



Score: 0. Health: 15. Total Critters: 7. Updates per second: 31.

2.2 What I have learned:

When working on deliverable one, I have learned how to use the Pop framework to develop a new game. Pop framework is built on top of the Microsoft Foundation Classes (MFC) and it is composed of a number of classes organized in a useful pattern that can be useful for computer game development. To develop my Gauntlet game, I have written some customized classes that implement some base classes in the Pop framework. For example, I have written the cCritterGauntletPlayer class that implements the cCritter class. The collide function of my cCritterGauntletPlayer class was written on top of the collide function of cCritter. The following code is my customized collide function. The code specifies that if the player bumps into a doughnut, his health will be increased by a constant and his score will go up. Also the strength of the bullet will increase by one.

```
BOOL cCritterGauntletPlayer::collide(cCritter *pcritter)
{
        BOOL collideflag = cCritter::collide(pcritter);

  if (collideflag && pcritter->IsKindOf(RUNTIME_CLASS(cCritterGauntletProp)))
 {
                setHealth(health() + cGameGauntlet::DEFAULTHEALTHDELTA );
                addScore(cGameGauntlet::SCOREDELTA);
                ((cGameGauntlet *)pgame())->increaseHitStrength(1);
                playSound("Ding");
                pcritter->die();
   }
        return collideflag;
}
```

The game engine I am going to write for my CS 298 project should accomplish a similar task as the Pop framework except that it can be used to implement more general games. My XML schema will define a standard to describe all the elements of a video game and the game developer can simply write a XML file to develop his own game. One of the advantages of using this XML language over the Pop framework is that the developer can customize the game simply by editing the values of the XML elements, instead of reading the lengthy program to find out which line of code to change.

Section 3: Deliverable Two

3.2 Goal:

The purpose of deliverable 2 is to describe a family tree using XML language and write a parser in Java to display the family tree. For this deliverable, I have written the familytree.xml, which describes the structure of a family and the familytree.dtd file, which is used to validate the XML file. Here is a fragment of the DTD file:

```
<!ELEMENT familytree (person*)>
<!ELEMENT person (name, birth, dad?, mom?, spouse?, death?, pic?, occupation? )>
<!ELEMENT name (firstname, lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

The above code specifies that a family tree is composed of one of more person elements. Each person element contains a number of elements such as name, birth and death information. The name element is made up of firstname and lastname. Both firstname and lastname are parsed character data. A java program is written to validate the XML file based on the DTD file. If the xml file is not valid, the program will print out error messages and if it is valid, it will then parse the familytree.xml file and display the elements of the tree.

3.2 What I have learned:

For my final project, I will come up with a schema for my XML language. While working on this deliverable, I studied the descendant of XML schema – DTD. DTD stands for Document Type Definition. It contains the elements, entities, attributes that are used in a XML document. Once a DTD file is created and a XML document is written based on the DTD, the DTD file is used to check if the XML file follows all the rules defined in the DTD file. This is called validation. In my java program, I use the Simple API for XML (SAX) to parse the XML file against the DTD file. In SAX, an XML tree is viewed as a stream of events generated by the parser. Some of these events are: the start and end tags of the elements are encountered, character data is encountered. Each event will invoke a corresponding callback method that the programmer writes. In my program, every time the parser reads in an element, it displays its name, its attributes if there is any and its value. Here are some sample outputs of my program.

```
dad : p1
mom : p2
spouse : p4
pic : pictures/p3.jpg
occupation : doctor
```

Section 4: Deliverable 3

4.1. Goal

The goal of deliverable 3 is to learn how to define the Centipede game using XML. The Centipede game is a very popular game in the 80s. The player of the game is represented by a small, insect-like character named “gnome”. The player moves the character at the bottom of the screen and shots at the centipede that is moving from the top of the screen down through some mushrooms. The centipede will drop down one level when it hits a mushroom. The centipede is composed of multiple segments. If the player shots at the center segment of the centipede, the centipede will be split into two pieces and each piece will continue moving down the screen. If the player shots at the any other segments of the centipede, a mushroom will be created. My centipede.xml file should be able to describe all the game objects and rules so that one can read this XML file and know how to create and play the Centipede game. The following is a part of the centipede.xml file that describes the player object of the game.

```
<player>
           <name>Gnomo</name>
           <shape>Insect-like</shape>
           <graphics>player.jpg</graphics>
           <position>
                <horizontal>0</horizontal>
                <vertical>0</vertical>
            </position>
            <lives>3</lives>
            <max-speed>3</max-speed>
            <on_collision>
              <collision_case id="1">
                <with_object>bomb</with_object>
                <lives>-1</lives>
               </collision_case>
               <collsion_case id="2">
                  <with_object>vertical_wall</with_object>
                  <position_change>move:0:0</position_change>
                </collsion_case>
             </on_collision>
             <move>
                <x_direction></x_direction>
                <y_direction></y_direction>
            </move>
             <die>
                <lives>0</lives>
              </die>
              <listener>
                <mouse_listner />
                <keyboard_listener />
              </listener>
        </player>
```

One can easily changes the name, graphics, speed and other factors of the player by editing the XML file.

4.2 What I have learned.

When working on this deliverable, I started working on the schema for my XML language for games. As I mentioned above, my XML language can specify the initial set up of a game such as name of the game and version, resource type and directory, environment and background, as well as properties and attributes of a game object, input and output devices and game rules. So my schema for this language should define all of these elements. Although I have tried to cover as many game elements as possible in my centipede.xsd file, it is not a complete schema yet. There are many elements a video game contains. The goal of my CS298 project is to come up with a complete schema that covers all elements in all video games. The schema of my XML language for games will be written on top of the schema I created for this deliverable.

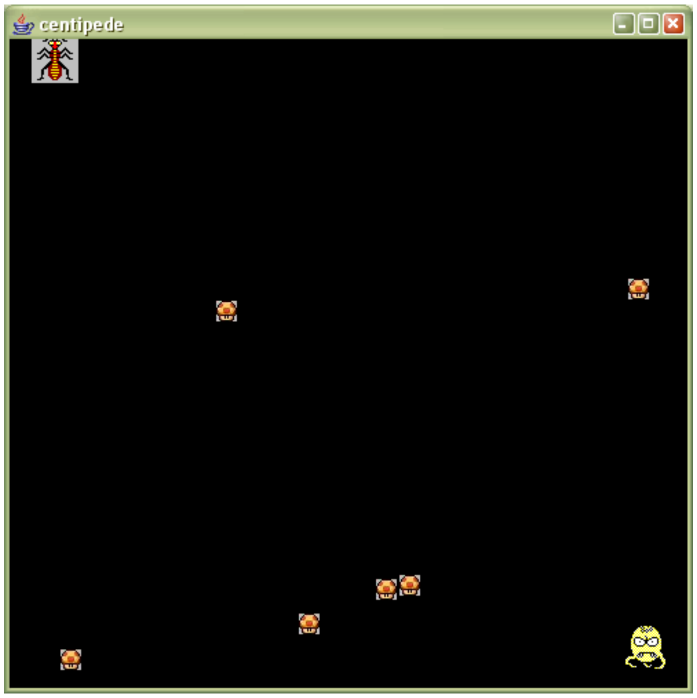
Section 5: Deliverable 4

5.1. Goal

The goal of deliverable 4 is to write a simple game engine to parse a XML file for the centipede game. This XML file is a simplified version of what I did for deliverable 3. My game engine will show the objects of the centipede game on the screen. The following is a segment of my centipede.xml file that describes the objects of this game.

```
<level id="1">
    <object>
        <player>
            <name>Gnomo</name>
            <graphics>player.gif</graphics>
            <position>
                <horizontal>450</horizontal>
                <vertical>450</vertical>
             </position>
        </player>
        <rival>
            <name>centipede_segment</name>
            <graphics>centipede_segment.gif</graphics>
            <position>
                  <horizontal>20</horizontal>
                  <vertical>20</vertical>
            </position>
        </rival>
        <not-movable>
                  <number>6</number>
                  <center>
                           <horizontal>250</horizontal>
                           <vertical>250</vertical>
                  </center>
                  <name>mushroom</name>
                   <graphics>mushroom.gif</graphics>
                   <position>
                             <horizontal>random</horizontal>
                             <vertical>random</vertical>
                     </position>
        </not-movable>
    </object>
</level>
```

And the following is a screen shot of the output of my program.



5.2 What I have learned:

It is not hard for an object-orientated programmer to see that we can map some of these element tags to Java classes. In my simplified game engine, I have defined a DrawObject class that has variables like name, graphics and position. I have also created a Player class, a Rival class and a Non-movable class, which extend the DrawObject class. The DrawObject class has a function called paint(), which paints the object itself on the screen. The following code shows how the game objects get displayed.

```
protected void paint(Graphics g){
                BufferedImage img =loadImage(gra);
                 g.drawImage(img, pos.getH(), pos.getV(),j);
}
private BufferedImage loadImage(String urlstr) {
        URL url=null;
        try {
                  url = getClass().getClassLoader().getResource(urlstr);
                  return ImageIO.read(url);
        } catch (Exception e) {
                System.out.println ("Error: "+e.getClass().getName()+" "+e.getMessage()
                System.exit(0);
                  return null;
                         }
}
```

This simplified game engine contains a parser, which reads in the centipede.xml file line by line. When it reads in a player, a rival and not-movable tag, the parser will create a corresponding java object. The program will then call the paint function of each object and show the object on the screen.

For my CS298 project, I am going to write a game engine to demonstrate my XML language for games. The game engine will display the game objects the way similar to this simplified game engine. My game engine will contain a parser. When the parser reads in a game object, it will then create a new object and the object will show itself on the screen.

Section 6: Future Work

The goal of my project is to define a XML language that is general enough to describe data for any video game and specific enough to cover every aspect of a video game. I will come up with a schema for this XML to define the legal elements of this language. All the game XML documents must follow this schema in order for it to be read by the game engine. A game engine will be implemented in Java to demonstrate how this XML language works. The XML parser in my game engine will parse the game XML document against the schema to decide if the document is valid or not. I will also measure how easy it is create games with this XML language versus other scripting languages. Since for some speed conscious game developer, using Java and XML to develop games may not be acceptable, I will write some tests to show how efficient various simple video games on my engine are compared to other engines.

Reference:

[RR03] Rudy Rucker. Software engineering and computer games / Rudy Rucker. Addison-Wesley, 2003.
[HM02] Harold, R. E., & Means, S.W. . XML in a Nutshell (2nd ed.). O'Reilly. 2002
[MM05] Mike McShaffry. Game Coding Complete, Second Edition. Paraglyph press. 2005
[MD00] Mark Deloura. Game Programming Gems. Delmar Thomson Learning. 2000
[MD01] Mark Deloura. Game Programming Gems 2. Delmar Thomson Learning. 2001
[BS04] Brian Schwab. AI Game Engine Programming. Delmar Thomson Learning. 2004
[D01] Patrick Dickinson. Building a Game Engine with Reproducible Behavior. http://www.gamasutra.com/features/20010713/dickinson_03.htm. 2001
[OL04] Oli Wilkinson. Creating Moddable Games with XML and Scripting. http://www.gamedev.net/reference/programming/features/modxml1/ 2004