Programming Languages and Translators, Fall 2011

Pacaml

A Design Language for PAC-MAN

Chun-Kang Chen (cc3260)

Hui-Hsiang Kuo (hk2604)

Shuwei Cao (sc3331)

Wenxin Zhu (wz2203)

9/28/2011

1. Introduction

PAC-MAN is an arcade game immensely popular since its original release and is considered as an icon of 1980's popular culture. It has made a great impact on a generation of people and is still appealing to the public for today. However, the game scenes of each stage, the size of the map, wining condition and the number of ghosts, etc in PAC-MAN are all pre-designed and players are unable to create or change them. It could be a great fun if the players can design a PAC-MAN based on their own favorite. Our project is to develop PaCaml, a PAC-MAN game design language, which enables users to create their favorite scenes and design whatever they like to make PAC-MAN more interesting.

2. Language Description

(1) General Description and Game Architecture

The language PaCaml (PAC-MAN + Ocaml) is developed using Ocaml language. It is text-based and java like but has its own primitives. The language has a 'main' function like C and each statement ends with a semicolon. The basic idea and rules in PAC-MAN will not be changed. However, the language allows users to design mazes, number and distribution of eating-dots, number and speed of the ghosts, location of the power pellets, figure of each character, etc. Users can also design the condition of scoring and set the winning condition using the language.

Features users can create or modify using PaCaml:

a. Stages: number of stages, name of each stage
b. Mazes: size of maze, type of barriers, arrangement of barriers
c. Eating-dots: number, distribution, scoring
d. Ghosts: number, name, speed when chasing, speed when escaping, scoring if eaten, character
e. Power pellets: number, arrangement, function
f. PAC-MAN: speed, character
g. Wining condition: score required for each stage, conditions
h. Rewarding of life: condition for life rewarding

Graphics user interface may be implemented in either PaCaml or the presentation of the final demo, depending on the progress of our project.

The design of PAC-MAN is based on grid, which stands for maze in the game representation. The size of the grid will be pre-defined by users. All objects like barriers, eating-dots, power pellets, and starting point of PAC-MAN will be placed in grid.

(2) **Syntax:**

We decide to build PaCaml a Java-like language. When we say Java-like, we mean to imitate Java in both object-oriented design and syntax. Similar to it in a great many programming languages, a semicolon indicates termination of a command while a pair of round brackets helps to locate the arguments. Compiler of PaCaml takes in commands in sequence, and interprets them in form of expression and argument. A mechanism is provided to recognize comments which are delimited by /* … */ in an ungreedy manner, and avoid overwriting reserved words.

(3) **Data Types**

So far as we have considered the details of PaCaml, we have designed the basic data types and operations, as discussed below

Basic Data Type	Example Values	Derived Data Type	Example Values
bool	TRUE, FALSE		
int	-1, 0, 1,	**vector**	(2, 4)
char	' c '	**string**	" char "
array	(1, 2, 3) (' c ', ' h', ' a ', ' r ')		

Game-specific structures are defined on the base of data types stated below.

Data Type	Definition

Map	Store player, enemies, and items.
Player	Store location and color.
Enemy	Store location and color.
Item	Store type, duration
Barrier	Store color
Point	Store two integers: x and y

(4) **Operators**

Basic Operator	Precedence
unary	!
additive	+ -
multiplicative	* /
postfix	++ --
relational	> < >= <=
equality	== !=
logical	&& ||
assignment	+= -=

(5) **Pre-defined Functions:**

Basic Function	Arguments	Explanations
setMap	Map map	Initiate the map
setLevel	String level	Set the level of the game. Types of Level could be either easy or hard.
setPlayer	Player player int x int y	Initiate the player
useEnemy	Enemy enemy	Add enemies to the map.

	Point p	
addItem	Item item Point p	Add an item to a specific location of the map.
addBarrier	Barrier barrier Point p	Add a barrier to a specific location of the map.
getMapWidth		Return the width of the map
getMapHeight		Return the height of the map
getMapItem	Point p	Return a item on a specific location. If there is no item, return null.
MapAvailable	Point p	
RandomAvailable		Return one available point

3. Sample Code

The following sample code initiates the map, creates a player and barriers, and then put them into the map.

```
int width, height;
point p(3,4);
width = 10; height = 5;
map m(width, height); // it will generate a randomly item
setMap(m);
Barrier b(blue);
if (MapAvailable(p))
       addBarrier(b,p);
Player pac(yellow);
while(!MapAvailable(p)) {
  if (p.x + 1 < getMapWidth())
     p.x++;
  else
      p.x = 0;
   if (p.y + 1 < getMapHeight())
     p.y++;
  else
            p = RandomAvailable();
}
addPlayer(pac, p);
```