Pacaml

Pac-Man Game Programming Language

Reference Manual

Chun-Kang Chen (cc3260)

Hui-Hsiang Kuo (hk2604)

Shuwei Cao (sc3331)

Wenxin Zhu (wz2203)

Table of Contents

1. Introduction

PaCaml (PAC-MAN + Ocaml) is a game programming language that facilitates design of elements in PAC-MAN scene, such as maps, characters, items, and win conditions. Compiler of PaCaml is written in OCaml.

This manual provides reference for application and syntax of PaCaml. Examples have been given in the end as a quick start.

2. Lexical conventions

2.1 Comments

A block of comment in PaCaml is delimited by "/*" and "*/". Unlike Java, no single-line specific format, i.e., "//", is provided. Identifying a comment block is a non-greedy operation. In other words, the first "*/" found after “/*” terminates the comment block.

No other constraint is given, considering characters and positions of comments.

Example:

/ These lines*
are not to be parsed by
*the PaCaml compiler */*

2.2 Naming

(1) Variable names are case-sensitive. A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter or the underscore character "_".

(2) Subsequent characters may be letters, digits, or underscore characters.

(3) No duplicate names are allowed in the fields including variables and functions.

(4) Keywords (defined in 2.3) are reserved.

2.3 Keywords

Keywords are reserved identifiers. The following are the keywords in PaCaml:
int, char, string, Map, Player, Item, Barrier, Gift, Ghost, Color
return, continue, break, if, else, do, while, for
pre-defined functions

3. Data Types

Two groups of data types are defined in PaCaml. Primitive data types include integer, character, string, and etc. Game specific data types include map, item, and etc. These two groups are introduced in section 3.1 and section 3.2 respectively.
PaCaml is statically-typed, which means that all variables must first be declared before they can be used.

3.1 Primitive Data Types

3.1.1 Integer (int)

The int data type is a 32-bit signed numeric variant. Negative numbers are signified by the minus sign ('-'), while no white space is allowed between the minus sign and the integral number.

Please notice that a point ('.') is prohibited in integer declaration, even if the value equals to an integer. In other words, 1 is a legal declaration while 1.0 and 1. are illegal. Default value given to an uninitialized variable is 0.

Examples:

0, 2, 256, -102

int a = 10;

In addition to numeric variant, an integer can be used as a presentation of a Boolean value. As convention, 1 stands for true and 0 (or null) stands for false.

Example:

int flag = 1;

if (flag) { … }

3.1.2 Character (char)

A single ASCII alphanumeric letter is denoted as a char. Letters can be either upper or lower case.

Examples:

a, A, d, F, !, =

Declaration:

char ch = 'a';

3.2 Derived Data Types

3.2.1 String

Array of characters is denoted as string. It is defined by double quotes “”.

Example:

“abcd”, “aAbB”

Declaration:

String str = “abcd”;

3.3 Game-specific Types

3.3.1 Map

Map is the abstract container of the game, including player and items. In other words, it is the world of Pac-Man.

3.3.2 Player

Pac-Man is the only object accessed through the data type Player.

Properties are defined in Player, including position, speed, status (normal, dead, invincible, slow, fast, and fighting, and the valid remaining time of the current status (0 means forever).

3.3.3 Item

Roles accessed through the data type Item are ghosts, gifts, and barriers.

Basic properties contained in an item are location, and image. Other important properties include type (i.e., type of the object, such as ghost, gift, and barrier), effect (which changes the status of PacMan), speed, status, move function, escape function, move counter, and maximum of move counter.

Functions move function and escape function are rand(0, 4) by default. Value of maximum of move counter is 5 by default. Value of move counter is set to 0 when PacMan changes the status between "normal" and "fighting".

The following example illustrates how components of items control the flow.

(1) Assume that Item i (ghost) and Player p have been defined in a game.
In condition that p.status != "fighting" and i.move_counter == 0, the server calls i.move_function.

(2) In condition that p.status == "fighting" and i.move_counter ==0, the server calls i.escape_function.

Only one of i.move_function and i.escape_function is called at each time. Result of the function must be integers, whose residue number given modulo 4 determines direction of the next move of i. 0 means up while the consequent ones rotates the direction clockwise.

In this sense, all items are able to move.

3.3.4 Point

Point is a 2D geometry concept, with two components, x, and y, which together depict the position of a player's or an item's.
Example:

Point p = [1, 2];

4. Expressions

4.1 Unary operators

4.1.1 – expression:

Numeric negation

4.1.2 ! expression:

Logical negation

4.2 Additive operators

4.2.1 expression + expression:

(1) Numeric addition
(2) String concatenation
(3) Join of maps

4.2.2 expression – expression:

Numeric subtraction

4.3 Multiplicative operators

4.3.1 expression * expression:

(1) Numeric multiplication;

(2) Expression of identifier * number creates the same identifier with different index by n times and new identifiers inherit all the attributes. e.g. Copy the ghosts.

4.3.2 expression / expression:

Numeric division

4.4 Relational operators

4.4.1 expression < expression:

(1) Less-than;
(2) Comparison of properties of two objects. The value is 1 if expr1 is less powerful than expr2. An example is given as:

```
Player p;
Item i;
if (p.speed < item.speed) { … }
```

According to the default definition, slow < normal < fast, while weak < normal < aggressive < invincible.

4.4.2 expression > expression:

(1) Greater-than;
(2) Comparison of properties of two objects. The value is 1 if expr1 is more powerful than expr2.

4.4.3 expression <= expression:

Similar to 4.4.1, except that comparison of properties also returns 1 if expr1 == expr2 (i.e., the two properties are of the same level. For details, please refer to 4.5.1).

4.4.4 expression >= expression:

Similar to 4.4.12 except that comparison of properties also returns 1 if expr1 == expr2 (i.e., the two properties are of the same level. For details, please refer to 4.5.1).

4.5 Equality operators

4.5.1 expression == expression:

(1) Equal-to
(2) 1 if the two properties are of the same level, and 0 otherwise.

4.5.2 expression != expression:

(1) Not-equal-to
(2) 1 if the two properties are not of the same level, and 0 otherwise.

4.6 expression && expression:

Return 1 if and only if both values are 1; otherwise it return 0.

4.7 expression || expression:

Return 1 if at least one of the two values are 1; otherwise it returns 0.

5. Declarations

Declaration creates new variables with a specified type and an optional values by the following forms:
type_specifier identifier
type_specifier identifier = initialization expression

where type_specifier is one of the following type: int, char, string, Map, Item.

Example:

```
int num;
int num = 14;
```

6. Statements and Blocks

6.1 Expression statement

Expression statement is a construct consisting of variables, operators and functions. It is like real word statements.

Examples:

```
int i=1;

item speed=slow;
map_earth+map_mars;
```

6.2 Conditional statement

Same as Java or C, conditional statement executes a certain code only if a particular condition is true.

Examples:

(1)

```
if (expression)
    {statement}
else
    {statement}
```

(2)

```
if (expression)
    {statement}
elseif (expression)
    {statement}
else
    {statement}
```

6.3 While statement

The while statement continually executes a block of statement while the condition is true.

Example:

```
while (expression)
{ statement }
```

6.4 Blocks

Block is a group of statements surrounded by a pair of braces. It is used in flow control sentences.

Example:

```
while (expression1)
{
    if (expression2)
        {statement1}
    else
        {statement2}
        }
```

6.5 Return statement

A function returns to its caller by means of the return statement, which has the following form:

```
    return ( expression ) ;
```

7. Function Definitions

7.1 Basic Functions

```
func return_data_type function_name (arg1,arg2, arg3)
{
        /*
        statement;
        */
}
```

7.2 Built-in Functions

7.2.1 main

int main()

This function is the entry point for execution of the program. Each PaCaml program must define this function. A return value of zero indicates failure; otherwise, success.

7.2.2 setMap

void setMap(Map map)

Set the map of this game. This function should be called before initiating the player, enemies, and barriers.

7.2.3 setLevel

void setLevel(String level)

Set the level of the game. Types of the Level could be either easy or hard. Example:

setLevel(Level.Easy); /* Difficulty level is set to easy */

7.2.4 setPlayer

int setPlayer(Player player, Point p)

Move the player to a new location.

7.2.5 addGhost

int addGhost (Item item)

Add user-defined ghost object to the current map. A return value of zero indicates failure; otherwise, success.

7.2.6 addGift

int addGift(Item item, Point p)

Add a gift to a specific location of the map. A return value of zero indicates failure; otherwise, success.

7.2.7 addBarrier

int addBarrier(Item item, Point)

Add a barrier to a specific location of the map. A return value of zero indicates failure; otherwise, success.

7.2.8 getMapWidth

int getMapWidth()

Return the current width of the map.

7.2.9 getMapHeight

int getMapHeight()

Return the current width of this map.

7.2.10 getMapItem

Item getMapItem(Point)

Return an item on a specific location, or null if there is no item.

7.2.11 isPointAvailable

int isPointAvailable(Point)

Check if the point of the current map is available.
Return 1 if the point is available, or 0 if this point has been assigned to another item.

7.2.12 getAvailablePoint

void getAvailablePoint()

Return one randomly chosen available point, or null if there is no available point on this map.

7.2.13 print

void print(String)

Print a string on standard output.

8. Examples

```
int main()
{
    int width = 10;
    int height = 5;

    /* generate a map of the game */
    Map m(width, height);
    setMap(m);

    /* create a barrier, and add it to the map */
    Point p = [3,4];
    Item b;
    b.type = Barrier;
    b.color = Color.blue;
    if (isPointAvailable(p))
    {
        addBarrier(b,p);
    }

    /*   create a player of this game */
```

```
        Player pac;
        pac.color = Color.yellow;
        Point initLoc = getAvailablePoint();    /* randomly choose a point    */
        setPlayer(pac, initLoc );
    }
```