GBL: Game Building Language

Yiqin Cui(yc3121) Sihao Zhang(sz2558) Ye Cao(yc3113) Shengtong Zhang(sz2539)
February 10, 2016

1 OVERVIEW

Most of us grows up with games. But it is extremely difficult to create a game from zero. We would like to provide such a language which could help game developers to generate their games easily and fast. So here comes GBL. GBL is a user-friendly programming language for game building. GBL is a python-like language with some differences, our compiler will compile it to python. Game developers can use GBL to create any games based on coordinates easily, from Gomoku, Chess to Tactical Wargames(e.g. Fire Emblem), even Role-playing Game (Dragon Quest). The only thing that game developers need to do is providing the game rule, the map information, the picture of items and related settings. Besides, there are lots of build-in function and data type in GBL which could save lots of time for users. We are seeking to develop an intuitive and efficient language so users could focus more time on the design of games not on the technical process.

2 SYNTAX

2.1 KEYWORD

As a strongly-types language, GBL need the values in the both side to be equal typed, there are 4 basic data types, and 6 extended kinds of data types, some keywords would be forbidden to be used as variable name.

2.1.1 BASIC DATA KEYWORDS

1. *int* : signed integers

2. *double* : floating point real values
3. *bool* : boolean value
4. *string* : a contiguous set of characters

2.1.2 COMPOSITE DATA KEYWORDS

1. *list* : holds a sequence of elements of the same type
2. *dict* : holds a sequence of key-value pairs and all the keys have same type, all the values have same type
3. *game* : the game object, which has attributes such as player number, player list and sprite list. And it controls the status of the whole game
4. *player* : the player object of the game, which has attributes such as name and texture, player can be controlled by the Game object and is used to record the action of each player
5. *sprite* : sprite is atomic individual in the game, just as a piece. It has attributes such as texture, position and owner
6. *map* : the map object of the game, which has attributes such as size and texture, also it can store sprites data

2.1.3 BASIC FLOW CONTROL KEYWORDS

1. *for*
2. *if*
3. *elif*
4. *else*
5. *break*
6. *continue*
7. *while*
8. *end* : mark the end of each block

2.1.4 BASIC LOGIC KEYWORDS

1. *and* : and operator
2. *or* : or operator
3. *not* : not operator

2.1.5 BASIC COMPARE KEYWORDS

1. *geq* : judge if a is greater or equal with b
2. *leq* : judge if a is less or equal with b
3. *gt* : judge if a is greater than b
4. *lt* : judge if a is less than b
5. *is* : judge if a is equal with b
6. *neq* : judge if a is not equal with b

2.1.6 OTHER KEYWORDS

1. *fun* : define a new function
2. *print* : print information on the console

2.2 OPERATORS

Operator Group	Operator	Usage
Arithmetic Operators	+	Add operator
	-	Left hand value minus right hand value or negative operator
	*	Multiply operator
	/	Divide operator
	%	Mod operator
Assignment Operators	=	Assignment operator
	+=	Add right hand value to left hand
	-=	Minus right hand value from left hand
	*=	Multiply right hand value to left hand
	/=	Divide left hand value by right hand
	%=	Assign left hand value to left hand value mod right hand value
Bitwise Operators	&	Perform AND operation on each pair of the corresponding bits
	|	Perform OR operation on each pair of the corresponding bits
	^	Perform XOR operation on each pair of the corresponding bits
	~	Perform NOT operation on each pair of the corresponding bits
	> >	Shift to right operator
	< <	Shift to left operator
Domain Operators	@	Mark the domain the function or attribute belongs to
Parenthesis Operators	[]	Access the list's member by index
	()	Place function's parameters
Comment Operators	#	Single-line comment
	#{	Begin of multi-line comment
	#}	End of multi-line comment

3 BUILD-IN FUNCTIONS

There are 4 data type for game construction, programmer can implement the build-in functions of game data type to generate a new game.

3.1 GAME

1. *setGameName()* : set the name of game
2. *setPlayerNum()* : set the player number of the game
3. *initializeGlobalValue()* : initialize other variables in the game
4. *setOperate()* : use function as parameter. Programmer can use this function to set what to do in the game
5. *addIslegal()* : use function as parameter. Programmer can use this function to set how to judge if the action of the Sprite is legal
6. *setIsFinished()* : use function as parameter. Programmer can use this function to set how to judge if the game is finished
7. *setGameover()* : use function as parameter. Programmer can use this function to set what to do after the game is complete
8. *operate()* : execute the game-playing activity in the round
9. *gameOver()* : execute the game over activity
10. *setLastMove()* : set the status of the game after one round

3.2 MAP

1. *setSize()* : set the size of map
2. *setBackground()* : set the texture of map
3. *addSprite()* : add a sprite to the map
4. *removeSprite()* : remove a sprite from the map

3.3 PLAYER

1. *setOperate()* : use function as parameter. Programmer can use this function to set what to do in player's round
2. *initialize()* : initialize other variables of the player
3. *setBackground()* : set the texture of the player
4. *operate()* : execute the game-playing activity of the player in the round

3.4 Sprite

1. *setBackground()* : set the texture of the sprite
2. *setPosition()* : set the position of the sprite

4 Sample Code

Sample code for a simple GoBang game using GBL:

```
fun main()
    setGameName('gobang')@Game
    setPlayerNum(2)@Game
    intializeGlobalValue(initializeGlobal)@Game
    setOperate(gameOperate)@Game
    addIslegal(isLegal1)@Game
    setIsFinished(finished)@Game
    setGameover(gameOver)@Game
    setSize(10, 10)@Map
    setBackground('background.png')@Map
    setOperate(oneMove)@Player
    player1 = initialize('player1')@Player
    player2 = initialize('player2')@Player
    setBackground('player1.png')@player1
    setBackground('player2.png')@player2
    initializePlayer([player1, player2])@Game
    while not isFinished()@Game
        operate()@Game
    end
    gameOver()@Game
end

fun initializeGlobal()
    int turn = 0
    int mapSizeX = 10
    int mapSizeY = 10
    int mapVal[10][10] = {0}
    # if lastMove is illegal, we have to restore it from formerMove
    int formerMove[4] = [-1, -1, -1, -1]
    int lastMove[4] = [-1, -1, -1, -1]
    int winner = -1
end

fun isLegal1()
    if lastMove[2] geq 10 or lastMove[2] lt 0 or lastMove[3] geq 10 or lastMove[3
        lastMove = formerMove
        print 'illegal move!'
        return False
    end
```

```
    if mapVal[lastMove[2]][lastMove[3]] is 1
        lastMove = formerMove
        print 'illegal move!'
        return False
    end
    formerMove = lastMove
    return True
end

fun gameOperate()
    if turn is 0
        print 'player1 turn!'
        operate()@player1
        turn = 1
    else
        print 'player2 turn!'
        operate()@player2
        turn = 0
    end
end

fun oneMove()
    [newX, newY] = getInputFromUser()
    setLastMove(-1, -1, newX, newY)@Game
end

fun finished()
    if five_chess_in_line()
        if turn is 0
            winner = 1
        else
            winner = 0
        end
        return True
    else
        return False
    end
end

fun gameOver()
    if winner is 0
        print 'player1 won!'
    else
        print 'player2 won!'
    end
end

fun playerOperate()
end
```