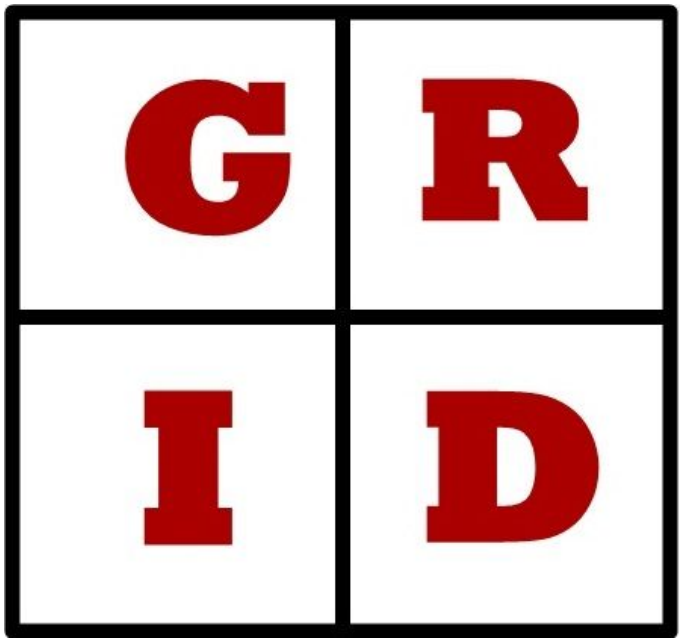


GridLang: Grid Based Game Development Language

Programming Language and Translators - Spring 2017
Prof. Stephen Edwards

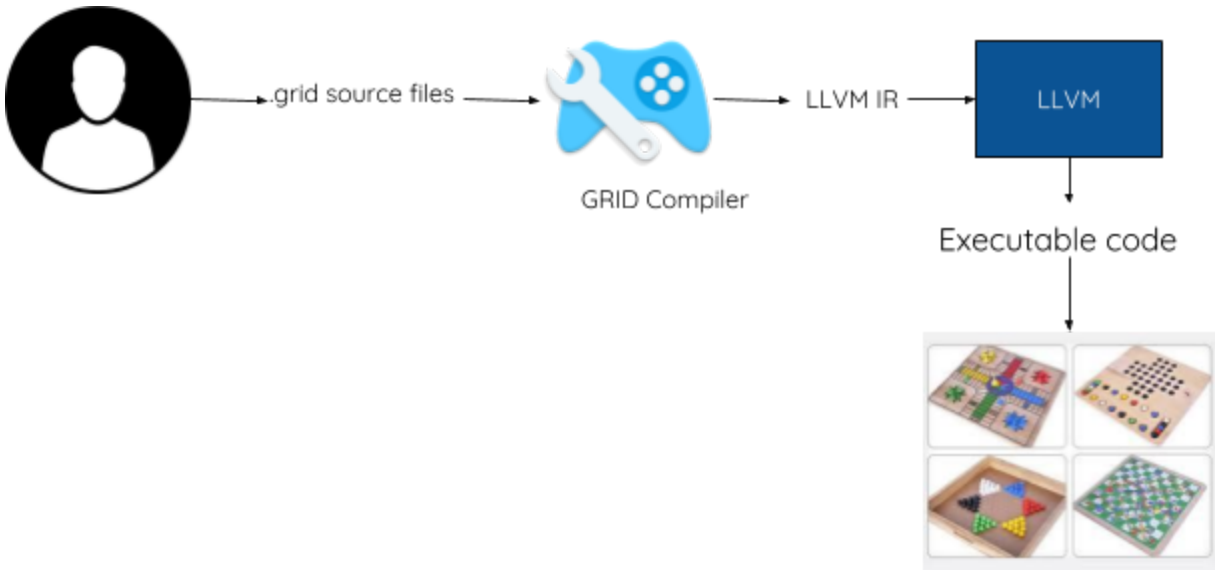
Akshay Nagpal	**an2756**@columbia.edu
Dhruv Shekhawat	**ds3512**@columbia.edu
Parth Panchmatia	**psp2137**@columbia.edu
Sagar Damani	**ssd2144**@columbia.edu

I. Introduction

GRID is a language to design games in an intuitive and expressive manner. It enables developers to quickly prototype grid-based games and get a programmatic view of it. It simplifies the process of defining rules for a game, creating a grid and manipulating them. There are in-built language components focused on game development that enables developers to express more with less lines of code.

II. Language

The GRID Compiler will be written purely in OCaml. The compiler will convert the GRID source code into an LLVM intermediate representation (IR) that will eventually be translated to machine code by the LLVM backend. Our language can be run on various platforms as LLVM guarantees machine independence.



III. Language Features

a. Comments

i. Single Line comments

```
# this is a single line comment
```

ii. Multi Line comments

```
##
This is a
Multi line comment.
##
```

b. Variable Declaration

<type-identifier> <variable-name-1>, ..., <variable-name-n>;

c. Data Types

Data Type	Example
int	int a = 3;
bool	bool myBool = true;
float	float a = 5.93;
string	string myString = “abc”
coordinate	coordinate c = (2,3)

d. In-Built classes

- **Grid**: Grid is a 2-dimensional global structure with cells. A specific cell in the grid can be referenced by its coordinates. Eg: Grid(3,4) will be the grid cell on the third row

and the fourth column. Each cell holds a list of Players and a list of Items. These can be referenced as Grid(3,4).playerList and Grid(3,4).itemList.

- **Player**: Player has the following attributes:
    - position: a coordinate denoting the position of the Player on the grid.
    - win: true/false
    - displayString: Denotes the string that will be displayed for the Player in drawGrid().
    - The getters and setters for the aforementioned attributes will be built-in.
- **Item**: An item is anything on the board other than the player. Item has the following attributes:
    - position: coordinates
    - displayString: Denotes the string that will be displayed for the Item in drawGrid().
    - interaction(): interaction() is a function that is triggered when a Player lands on the same grid cell as that item. The behavior of the function will be defined by the programmer.
    - The getters and setters for position and displayString will be built-in.
    - To create custom items, the item will be declared as:
      custom Item newItem;
      The developer can then add properties to this custom item using the addProperty() function described in the section below.

e. In-Built functions

- **createGrid(int rowCount, int colCount)**: Creates the global Grid with rowCount rows and colCount columns.
- **drawGrid()**: Draws the grid on the console.
- **traverse(Coordinate c1, Coordinate c2, listType)**: Traverses a row, a column or a diagonal in the grid with start coordinate c1 and end coordinate c2. Returns a list of the listType specified.
  Eg: traverse((1,1), (3,3), playerList) returns a list of players in the cells along the diagonal from (1,1) to (3,3).
- **addProperty(player/item, name, dataType)**: Add a property to all players or a specific player or a custom item.
  Eg 1: Add property ‘score’ to player p1.
  Player p1; addProperty(p1, score, float); p1.score= 43.3;
  Eg 2: Add property ‘score’ to all players.
  addProperty(Player, score, float)
- **random(0,1)**: Generate a pseudo-random number between 0.000 to 1.000.
- **gameOver()**: Displays the winner and exits the program.

f. Operators

- ‘+’ (Addition) : Adds values on either side of the operator.
- ‘-’ (Subtraction) : Subtracts right-hand operand from left-hand operand.
- * (Multiplication) : Multiplies values on either side of the operator.
- / (Division) : Divides left-hand operand by right-hand operand.
- % (Modulus) : Divides left-hand operand by right-hand operand and returns remainder.
- ++ (Increment) : Increases the value of operand by 1.
- -- (Decrement) : Decreases the value of operand by 1.
- Relational Operators: >, <, =, ==, <=, >=, != (standard meanings)
- Logical Operators: and, or, ! (standard meanings)
- Assignment Operators: =, +=, -=, *=, /=, %= (standard meanings)

g. Language specific Operators

- >> (shift right by) : Player/Item >> int : Shift Player or Item datatype right by given number of steps.
- << (shift left by) : Player/Item << int : Shift Player or Item datatype left by given number of steps.
- ^^ (shift up by) : Player/Item ^^ int : Shift Player or Item datatype up by given number of steps.
- __ (shift down by) : Player/Item __ int : Shift Player or Item datatype down by given number of steps.

h. Input/Output

- print(String s): prints s on console.
- prompt(String msg, dataType): prompts user for input of a certain dataType.

IV. Programming Features

a. If Loop

```
if (<condition>) {
        <statement1>
        <statement2>
} else {
        <statement3>
        <statement4>
}
```

If the condition is true, then it executes the statements in the first block as limited by the '{}' parentheses, else it executes the statements in the second block as limited by the '{}' parentheses.

b. For Loop

```
int n = 5;
for (int i = 1; i <= n; i++) {
        <statement1>
        <statement2>
}
```

While the condition mentioned by the second expression is true, the loop continues iterations, each time executing the statements in the block limited by the '{}' parentheses.

c. While Loop

```
int i =1;
int n = 5;
while (i != n) {
        <statement1>
        i++;
}
```

While the condition mentioned by the expression is true, the loop continues iterations, each time executing the statements in the block limited by the '{}' parentheses.

d. Main Function

```
returntype main() {
        returntype result;
        <statement1>
        <statement2>
        return result;
}
```

Every program must have a main function. The program starts execution from the main function. On successful execution, the main function returns a value of type returntype as specified in the return statement.

e. User Defined Functions

```
returntype function_name (parametertype parameter){
        returntype result;
        <statement1>
        <statement2>
          return result;
}
```

A user defined function is represented by the above syntax. It consists of the returntype followed by function name followed by the function parameters.

V. Example Program

Tic-tac-toe in GRID:

```
Grid g1 = createGrid(3,3);
Player p1, p2;
Item x, o;
x.setDisplayString("X");
o.setDisplayString("O");

boolean checkGameEnd(Player p, Item item){
        boolean flag = true;
        coordinate start[] =    { (0,0), (0,0), (0,0), (2,0), (2,0),(1,0),(0,2),(0,1) };
        coordinate end[] =      { (0,2), (2,0), (2,2), (2,2), (0,2),(1,2),(2,2),(2,1) };
        for(i=0; i<start.length; i++) {
                iList = traverse( start[i], end[i] ).itemList;
                for(j=0; j < itemList.length; j++) {
                        if(iList[j] == item.displayString){
                                continue;
                        }
                        else{
                                flag = false;
                                break;
                        }
                }
```

```
                if(flag){
                        p.win = true;
                        return true;
                }
                else {
                        return false;
                }
        }
}

void main() {
        int input = prompt("Press 1 to start game\nPress 2 to exit", int );
        if(input == 1) {
                while( !checkGameEnd( p1, x ) or !checkGameEnd( p2, o ) ){
                        drawGrid();
                        print("Input Format: 0,0");
                        coordinate xc = prompt("Player 1: Input location for X",
coordinate);
                        Grid(xc) = x;
                        if( checkGameEnd( p1, x ) ){
                                gameOver();
                        }
                        else{
                                drawGrid();
                                xc = prompt("Player 2: Input location for O",
coordinate );
                                Grid(xc) =  o;
                        }
                }
                gameOver();
        }
}
```