Crall Language Specification

Dietrich Hartman and Yo Akiyama

December 10, 2018

1 Introduction

Using conventional programming languages, creating a game even as simple as a 2-dimensional maze can require a moderate level of programming experience. Such a requisite thwart creative opportunities for those who wish to create games, but lack proficiency in coding. Through the development of a new programming language, we hope to construct a programming language that will allow an inexperienced coder to create a simple Dungeon Crawler type game. With this, we name our programming language Crall.

This programming language will offer a more intuitive approach to programming this type of game. We hope that Crall can simplify this process significantly, and offer an interactive introduction to basic game development.

2 Design Principles

Crall is a minimalistic, command based programming language that drastically simplifies the Dungeon Crawler game development process. Programmers first establish the dimensions of the map, and add desired specifications using basic commands separated by new lines. Thus, a programmer only needs to understand the structure and function calls of Crall in order to create their work. More advanced techniques such as loops and recursion are unnecessary and unsupported in Crall.

Running a valid .crall program materializes the code in the form of a two-dimensional map printed onto the console. All objects and spaces within the map are represented by ASCII characters.

3 Examples

```
Example 1:
//This example is a simple 5x5 map with one
// wall at position (3,0) and the goal at
// position (4,0). The player will start
// at position (2,0)

dimension: 5x5
start: (2,0)
goal: (4,0)
wall: (3,0)


Example 2:
//This example is a 5x5 map that has no walls
//but a simple trap that ends the game if the
//player walks over it since it does 100 damage
dimension: 5x5
start: (2,0)
goal: (2,3)
```

```
trap: (2,2),100 //Set a trap at point [2,2] that does 100 damage


Example 3:
//A 20x10 map featuring a teleporter
//at position (13,5) that transports
//the player to position (0,5).
//This map features a horizontal wall from
// position (0,2) to position (3,2)
// Player will start at (0,0), and the goal
// is positioned at (0,5)

dimension: 20x10
start: (0,0)
goal: (0,5)
wall: (0,2),(3,2)
teleporter: (13,5),(0,5)
```

All of the above examples can be run by passing in the file with dotnet run:

```
dotnet run example_1.crall
```

Programs in Crall are evaluated by stringing together Sequence Operations and stepping backward through the AST, performing each command as it is reached. Since Crall is composed only of one line commands, evaluation of any program is as simple as walking through the given sequence. For example, in Example 1, Crall evaluates the program by calling the SetDimension function first, then stepping back and calling the SetStart function, then SetGoal, and finally Set Wall. This produces a map of the desired size and layout described in Example 1.

4 Language Concepts

As previously discussed, we developed Crall in order to simplify the game development process for inexperienced users. As a result, programmers only need to understand function calls and basic two-dimensional array structure. Although the objects are actually stored in a map, we relieve the programmer of the burden of having to know how maps work by converting the map into an array for them. The programmer is only responsible for entering function calls separated by new lines. Through this programming language, we hope to offer users an entertaining medium for learning elementary computer programming.

5 Formal Syntax

The syntax for Crall is constructed almost entirely of function calls. In order to create a game, the user must first specify the dimensions of the array. Otherwise, the program is invalid. Any edit to the map requires function calls to set the starting point, goal, walls, traps, and teleporters in their desired locations. After running the .crall code, the user can type and enter w/a/s/d into the console to control where their player moves, allowing users to play the games they constructed.

```
<expr>          ::= <SeqOp>
                  |  <SetDim>
                  |  <SetGoal>
                  |  <SetStart>
                  |  <SetWall>
                  |  <SetLongWall>
```

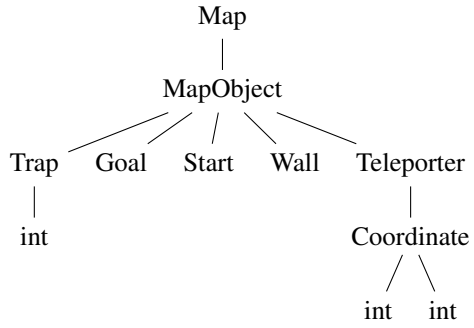
```
                    |  <SetTrap>
                    |  <SetTeleporter>
                    |  <number>
<number>           ::= 0 | 1 | 2 ... 997 | 998 | 999
<SeqOP>            ::= <expr> <expr>
<SetDim>           ::= dimensions: <number>x<number>
<SetStart>`        ::= start: (<number>,<number>)
<SetGoal>          ::= goal: (<number>,<number>)
<SetWall>          ::= wall: (<number>,<number>)
<SetLongWall>      ::= wall: (<number,<number>),(<number>,<number>)
<SetTrap>          ::= trap: (<number>,<number>),<number>
<SetTeleporter>    ::= teleporter: (<number>,<number>),(<number>,<number>)
```

6 Semantics

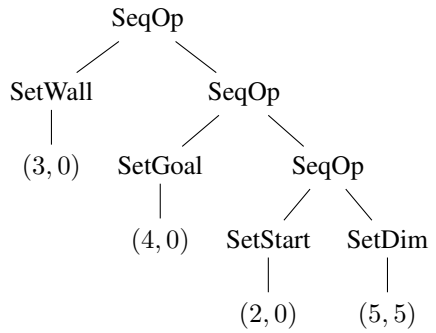
Due to the command-based nature of Crall, the only primitive-type is an integer. The user will specify dimensions and locations of objects by providing a coordinate. These coordinates will be in the form (int, int).

The language features commands when the user begins playing the constructed games. These commands will be something like move up/down/left/right to move the character around the Map. After creating the game, the user can initialize game play, and use these commands to play the game.

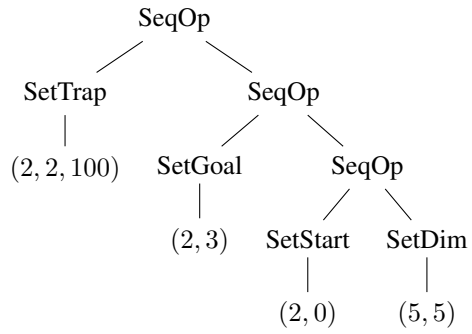
The program is represented using the following class hierarchy.



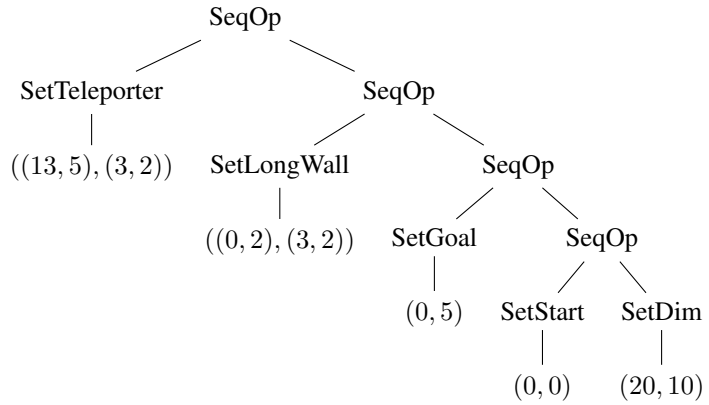
The abstract syntax tree for example 1 can be shown as



The AST for example 2 can be shown as



Lastly, the AST for example 3 can be shown as



Running a valid .crall program in the console will produce a 2-dimensional grid with the program's specifications. The user will then be able to enter w/a/s/d to move the player up/left/down/right in the grid. This game play will continue until the player dies or lands on the goal.

The following table provides a description of each language element.

Syntax	Abstract Syntax	Type	Meaning
dimension: nxm	SetDim of int*int	int*int → Dimension of int*int	Sets the dimensions of the map
start: (n,m)	SetStart of int*int	int*int → Player of int	Sets the coordinates of the starting position to (n,m)
goal: (n,m)	SetGoal of int*int	int*int → Goal	Sets the coordinates of the goal to (n,m)
wall: (n,m)	SetWall of int*int	int*int → Wall	Sets a wall at the coordinate (n,m)
wall: (n,m),(x,y)	SetLongWall of (int*int)*(int*int)	(int*int)*(int*int) → Wall	Sets a wall that extends from coordinate (n,m) to coordinate (x,y)
trap: (n,m),d	SetTrap of int*int*int	int*int*int → Trap of int	Sets a trap at coordinate (n,m) that does d damage
teleporter: (n,m),(x,y)	SetTeleporter of (int*int)*(int*int)	(int*int)*(int*int) → Teleporter of int*int	Sets a teleporter at coordinate (n,m) that transports player to coordinate (x,y)

7 Remaining Work

For our final implementation of Crall, we incorporated the game play portion of the program. While not exactly part of the programming language, including this feature allows users to interact with their creation, enhancing their experience and providing another medium through which the users can better understand and improve their code. Nevertheless, there exist additional concepts that, if included, could further improve the versatility of the programming languages.

The remainder of our work will focus on improving the game play portion of the code, and coming up with more features to add into the map. First, we can improve the game play portion by creating a user-friendly GUI to play the game outside of the terminal. Additionally, allowing the user to play the game using the up/down/left/right arrows on the keyboard would further enhance the users' experience.

There exists a plethora of Dungeon Cralwer games, so for Crall to be considered a truly robust tool for programming Dungeon Crawlers, we must add more features that allow the user to create whatever crawler game the user pleases.