Project Manager: Lilia Nikolova (lpn2112)
Language Gurus: Maxim Sigalov (ms4772), Dhruvkumar Motwani (dgm2138)
System Architect: Srihari Sridhar (ss4964)
Verification and Validation: Richard Munoz (rtm2129)

Senet

30th September 2015

OVERVIEW

Past projects for Programming Languages and Translators have included languages for expressing the setup and flow of playing card games. Inspired by such languages, we propose to extend the domain to general, two-dimensional board games. Examples of games that might be expressed in our proposed language are tic-tac-toe, checkers, and chess. A similar idea has been investigated by Romein, Bal and Grune (1995), who described a language called *Multigame* that compiled to a parallel game playing program.[1] The authors focused their research on parallelized artificial intelligence to find optimal moving while playing games created in *Multigame*. In part due to this research focus, the authors restricted the class of games that could be described in *Multigame* to those with fixed-sized boards (thereby excluding card games) and to those where all players have perfect information.

We propose a similar language focused on simple expression of board games; however, we will construct our compiler to create games that may be played interactively on the command line. The players will execute the game program of their choice after which they will be presented with prompts that navigate them through the game. We have named our new language *Senet* after one of the oldest-known board games, which traces its origins back to ancient Egypt.

GOALS

With our proposed language, we aim to provide:

1. Intuitive, relatively high-level expression of the setup and flow of board games;
2. Simple description of boards and pieces; and
3. Static, strong typing, and a mix of C and Python syntax to minimize the learning curve.

[1] J. Romein, H. Bal and D. Grune. (1995). Multigame - A Very High Level Language for Describing Board Games. ASCI 95, pp. 278-287.

SPECIFICATIONS

Operators

Senet includes basic math operators found in a wide variety of programming languages (`+, -, /, *, **, %`), logical operators (`or, and, not`), comparison operators (`>, <, >=, <=, ==, !=`). The meanings of these operators are similar to their meanings in Python, except that all math operations are purely integral.

Built-in Types

Senet includes a number of basic types from Python as shown in the table below.

Basic Types	**Meaning**
int	32-bit Integer
char	Char
str	String (list of char)
bool	Boolean (True or False)
list	C-like arrays
set	Unique sets
void	type of None, a value used to represent the absence of a value

The language is object-oriented, with inheritance (but no multiple inheritance). As such, the standard classes shown in the table below are built in and meant to be extended by the programmer.

Standard Classes	Meaning	Methods & Variables

Board	Defines board geometries, win conditions, cleanup methods	`remove(int x)`: removes the piece at index `x`
		
		`owns(int x)`: returns the number of the player at index `x` of `cells`
		
		`full()`: returns True if all spots are occupied, else False
		
		`cells`: list of board cells, each element is either None or the piece at that spot

Piece	Defines possible moves, keeps track of position, owning player, and other needed variables	`place(Board b, int x)`: places the piece on board `b` at index `x` of `b.cells` `owner`: the owner of the piece `fixed`: whether or not the piece can be overwritten

Control Flow

The language includes `if`, `for`, and `while` which operate as usual. Also, `for … in` syntax can be used as in Python. All games must have two program sections: `@setup`; which contains functions, objects, and parameters used to set up the game; and `@turns`, which contains several "phase" functions each of which operate as a "`while True`" loop but can call other functions in the `@turns` section.

Syntax

The language's syntax borrows C-style brackets, semicolons, and function and variable definitions. Lines (or the remainder of a line) can be commented one at a time with "`#`".

EXAMPLE PROGRAM

Tic-tac-toe is a two-player game that is played on a three row, three column board. The players take turns placing either an “X” or an “O” in each cell. A player wins if three of their pieces fall in a line (vertical, horizontal, or diagonal). The game ends in a draw if all cells are full and no player has won. Below, we describe how our language could be used to create an interactive tic-tac-toe game.

```
@setup
{
	class b (Boards.Rect(3, 3)) {
		three_in_a_row(player) {
			for (l in {[0, 1, 2], [3, 4, 5],
				[6, 7, 8], [0, 4, 7],
				[1, 5, 8], [2, 6, 9],
				[0, 4, 8], [2, 4, 6]}) {
					# Tests each line
				if (b.owns(l[0]) == player and
					b.owns(l[1]) == player and
					b.owns(l[2]) == player) {
						return True;
				}
				# owns checks the owner of a piece at an index,
				# and returns -1 if the space is empty
			}
			return False;
		}

		bool won (int player) { # checks if the player won
			if three_in_a_row(player)
				return True;
			return False;
		}

		bool draw() {
			if b.full()
				return True;
			return False;
		}
	}

	class Mark (Piece) { # inherits from Piece
		self.fixed = True; # piece cannot be overwritten
		char repr () {
			if (self.owner == 0) { # self.owner is the id of the owner
				return ‘X’;
			} else {
				return ‘Y’;
			}
		}
	};
```

```
	N_PLAYERS = 2; # number of players, this is mandatory
	N_PHASES = 1; # number of distinct game phases, 1 by default
	print("Input coordinates of square to place")
	print("in i.e. \"22\" or \"10\".\n"); # prompts the players

	reset()
	{
		char c;  #defaults to ' '
		while(c!='y' && c!='n') {
			print("Do you want to continue playing?\n");
			print("Type y to continue and n to exit\n");
			c=read(1)[0];
		}
		if(c=='y') {
			restart(); # restarts the game
		}
		else {
			exit(); # exits from the game
		}
	}
}


@turns
{
begin () {
	# this is basically just "while True" with only 1 phase
	# players input moves by typing coordinates, e.g. "11" or "02"
	int a = stoi(read(1)); # reads one character and converts it to an int
	int c = stoi(read(1));
	if (Mark.place(b, b.toindex(a, c))) {
		if (b.won(ON_MOVE)) { #ON_MOVE is the index of the player
			print("Player " + itos(ON_MOVE + 1) + " wins.\n");
			print("Congratulations!");
			reset();
			# Instead of ending the game entirely, could we
			# ask if the players want to continue playing? Say, a reset
			# function which either calls exit or clears the board
			# for a new game??
		}
		if (b.draw()) {
			print("Game ends in a draw.\n");
			reset();
		}
		pass(); # if the move was legal, went through successfully,
			# and the game is not over, pass the turn to the
			# next player
	}
}
```