BGGL (Board Game Generator Language)

COMS W4115: Programming Languages and Translators
Professor Stephen Edwards

@cs.columbia.edu
{
Matt Chu (mwc2110)
Steve Moncada (sm2277)
Hrishikesh Tapaswi (hat2107)
Vitaliy Shchupak (vs2042)
}

Overview

Coding any of the familiar board games using a standard object-oriented programming language like C++ or Java would prove decidedly tedious. Even the most expert OOL-user would waste a large amount of time specifying the (often repetitive) characteristics of traditional board games. BGGL seeks to dramatically cut down on the time a board game programmer would traditionally spend tweaking the class structure of his code while significantly enhancing the way gameplay terms (i.e. rules) are implemented. Programmers using BGGL will never spend time devising the most efficient board game framework, instead, they'll start with a palette of boards, regions, pieces, and rules and begin implementing their idea immediately.

Goals

The goal of BGGL is to abstract board games' traditional frameworks and provide the programmer with a rich set of tools that he can intuitively implement the rules and flow of a game. Throughout the development of BGGL, the abstraction and reduction of non-essential or monotonous tasks is a primary focus. Ideally, BGGL will not only be used by programmers seeking to mimic the functionality of existing board games electronically, but utilized as an instrument for developing new games. Indeed, what we find most exciting about BGGL's potential relates most closely to the development of new games. We'd like to develop a module within BGGL that will take the game's text-based code implementation and output a graphical representation of the game. This will be especially useful for developing new games, as it will quickly elucidate possible pitfalls and inspire potential enhancements.

Solving Complex Problems with Language Constructs

As mentioned above, there are many obstacles for a board game programmer that don't involve algorithmic complexity but time-consuming tedium instead. We've built in the following features to BGGL to ameliorate these problems:

- Easy-to-use region-referencing built into board game-specific datatypes.
  ```
  if (rows(TTT) == [X, X, X])
  ```

- Subsets of boards easily accessible through custom datatypes.
  ```
  if (longdiags(TTT) == [X, X, X])
  ```

- Repetitive action structures like board game "moves" specialized
  ```
  thismove = x:y:O:+;
  ```

- 'Turn' blocks are flow control structures specific to BGGL which simulate independent subsets of a board game's lifespan.

- Implicit functions like 'isValidMove' execute common, board-game specific constraint checks.

- Global variables like 'gameover' and 'winner' are implicitly initialized and monitored for straightforward game control.

Sample BGGL Program: Tic-tac-toe

```
/* Construction of a board square board with width and height of 3. */
Board TTT(3, 3);

/* 2 players. */
Player p1, p2;

/* We have two kinds of pieces for tic-tac-toe, "X"s and "O"s */
Piece X, O;

isValidMove
{
        if (TTT(thismove.x, thismove.y) != none)
        {
                return true;
        }
        return false;
}

Player nextPlayer()
{
        if (thisplayer == p1)
        {
                return p2;
        }
        return p1;
}

isGameOver
{
        if (rows(TTT) == [X, X, X] or cols(TTT) == [X, X, X] or
            longdiags(TTT) == [X, X, X])
        {
                gameover = true; // built-in variable
                winner = p1;     // built-in variable
        }

        if (rows(TTT) == [O, O, O] or cols(TTT) == [O, O, O] or
            longdiags(TTT) == [O, O, O])
        {
                gameover = true;
                winner = p2;
        }

        if (movecount(p1) + movecount(p2) == 9)
        {
                gameover = true;
        }
}

getMove()
{
        x, y = input "Enter x, y";
        if (thisplayer == p1)
```

```
        {
                thismove = x:y:X:+;
        }
        else
        {
                thismove = x:y:O:+;
        }
}

/* This is like "int main()" in C/C++/Java. */
game
{
        thisplayer = p1;
        turn main_turn
        {
                getMove();

                trymove(m.piece)
                /* calls isValidMove, enter the block if the move is valid */
                {
                        case X:
                        case O:
                               makemove m;
                }
                fail
                /* enters this block if trymove fails */
                {
                         gototurn main_turn;
                }


                isGameOver;
                /* calls isGameOver() */

                if (gameover == true)
                {
                        print "Game Over! Winner is " + winner;
                        exit;
                }

                thisplayer = nextPlayer();
        }
}
```

Summary

By leveraging the abundant similarities of most board games, BGGL abstracts the class frameworks usually associated with programming something as complex as a board game and provides an intuitive means of developing games. Ideally, BGGL will also support a straightforward protocol for transforming a board game's code into a text-based graphical user interface.