Board Game Generation Language
A Brief Introduction

- Overview of BGGL
- BGGL Language Highlights
- Implementing Tic-Tac-Toe with BGGL
- Summary
BGGL Overview: Goals

- Capture the essential components of a board game to assist game coders
- Specialize these components to provide the programmer with a rich code palette
- Eliminate tedious error-checking
- Create an environment for the invention of new board games
BGGL Overview: Strengths

• Versatile board game data types integrated with conventional programming language constructs
• Built-in language features tailored specifically for board games
• Flexible, robust rule specification syntax
BGGL Overview: Weaknesses

- Domain-specificity restricts applicability to other computational domains
- Extensive syntax steepens the learning curve for even the most basic functionality in BGGL
- No extensibility support
BGGL Highlights: Board

- Global variable with convenient manipulation functions

```plaintext
board = <[W, B, W]
[B, W, B]
[W, B, W]>;

/* specifies the following board:

0 1 2
0 W B W
1 B W B
2 W B W
*/
```
BGGL Highlights: Rules

• Rules in BGGL act like functions
• Pieces accepted as targets
• Composed of 4-tuple custom constraint syntax

```cpp
rule pawn_capture(): BP, WP {
    return test 1, diag, false, false;
}

/*
specifies rule for pawn capture on black, white pawns:
length: 1, (how far can it move?)
direction: diag, (how can it move?)
jump: false, (hops another piece?)
emptysquare: false (lands on empty?)
*/```
BGGL Highlights: Move

• Moves interface with Pieces and the Board via 4- or 6- tuples

```plaintext
piece G;
move m = :^:G:0:0:1:1;
/*
G _ _
_ _ _ moves to _ G _
_ _ _ _ _

move syntax = : <movetype> :
  <piece> : <row_source> :
  <col_source> : <row_target> :
  <col_target>;
*/
```
BGGL Tutorial: Tic-Tac-Toe

Critical Code: Game Rule Declarations

```plaintext
rule no_overwrite(): X, O {
  return test , , , true; // the only special constraint is that the destination
  // square should be empty
}

func getpiece(player p) returns piece {
  if (p == p1) { return X; } else { return O; }
}

func getwinner() returns player {
  int i;
  player winner;
  for (i = 0 to 2) {
    if ( <i> == [X,X,X] || <i> == [X,X,X] ||
        <0> == [X,X,X] || <0> == [X,X,X]) {
      winner = p1;
    } else {
      if ( <i> == [O,O,O] || <i> == [O,O,O] ||
          <0> == [O,O,O] || <0> == [O,O,O]) {
        winner = p2;
      }
    }
  }
  return winner;
}
```
game { 
  board = 
  <[_,_,_] 
  [_,_,_] 
  [_,_,_]>; //empty tic tac toe board stored in global variable

  boolean done = false;
  player thisplayer = p1;
  int row; int col;
  piece currpiece;
  print board;
  int countmoves=0;

  while (!done) {

    print "Player " + thisplayer + ": " + getpiece(thisplayer);
    row = input "Enter row coordinate: ", int;
    col = input "Enter col coordinate: ", int;

    currpiece = getpiece(thisplayer);
    move m = :+:currpiece:row:col;
  }
if (no_overwrite():m) {
  apply m;
  if (thisplayer == p1) {
    thisplayer = p2;
  } else {
    thisplayer = p1;
  }
  countmoves = countmoves + 1;
}
else { print "Invalid coordinate"; }
print board;

player winner = getwinner();
if (winner == p1 || winner == p2) {
  print "" + winner + " won!";
  done = true;
}
else {
  if (countmoves == 9) {
    print "It's a draw!";
    done = true;
  }
}
BGGL Conclusion: Framework

input_file.bggl

Lexer → Parser

AST Walker
Semantic Analysis

Exception Handler
Symbol Table
Type System

Interpreter

Console Input → Console Output

Test Execution

Collection of Test Inputs
BGGL Conclusion: Wishlist

- The implementation of turn{ } blocks as a specialized control flow mechanism
- Additional attention to usability via condensed syntax and semantics
- Better support for non-domain-specific tasks
BGGL Conclusion: Take-aways

The next time we build a programming language, we'll...

• Utilize similar directory organization, version control, and testing processes
• Emphasize the importance of initial planning by spending very late nights early in the process, not just at the end