GridLok
---
PLT Spring 2016 Final Project
players{
    "Julian Edwards",
    "Laura Hu",
    "Alice Hwang",
    "Bryan Yu"
}
The Language
Our goal:

The purpose of GridLok is to more easily facilitate the creation of grid-based games, such as Tic-Tac-Toe, Minesweeper, or even Chess, along with their user interface. Our language implements various unique for-loops to make it easier to code for different parts of the board. Additionally, GridLok makes use of the SDL C library to render the images for the game user interface.
Language specifics:

```python
game{  
  board{  
    dimensions{5, 5}  
    image{"images/bg.png"}  
  }  
  players{"tom", "jerry"}  
  turnOrder{"tom", "jerry"}  
  piece{  
    name{"tom"}  
    image{"images/tom.png"}  
    onClick{}  
  }  
  piece{  
    name{"jerry"}  
    image{"images/jerry.png"}  
    onClick{}  
  }  
  setup{  
    place{"tom", 0, 0}  
    place{"jerry", 5, 5}  
  }
}
```
For loops:

Format:

for pieceID in row(int){...}
for pieceID in col(int){...}
for pieceID surrounding(int,int){...}
for (intID,intID) surrounding(int,int){...}
for intID(int x,int y){...}
for(intID,intID) in board{...}
for intID(int,int), intID(int,int){...}
for all pieceID in board{...}

Use:

for pieces in a row r
for pieces in a column c
for pieces surrounding a coordinate (x,y)
for coordinates surrounding a coordinate (x,y)
for loop over a range of ints (x,y)
for loop over all coordinates of a board
nested for loops for ranges of ints
for all spaces of a board
Source Code
scanner.mll

Total: 67 Lines
open Ast

type var_ref = string

type var_new = string

type func_name = string

type obj_field = string

type expr_detail =
| Parent of expr_with_type
| Id of var_ref
| BoolLiteral of bool
| IntLiteral of Int
| StringLiteral of string
| Binop of expr_with_type * Ast.op * expr_with_type
| Unop of Ast.op * expr_with_type
| Access of var_ref * obj_field
| BoardAccess of expr_with_type * expr_with_type
| Rand of expr_with_type * expr_with_type
| Empty
| and expr_with_type = Ast.typ * expr_detail

type board_typed = {
    x: expr_with_type;
    y: expr_with_type;
    img: expr_with_type;
}

(*piece definitions*)

type piece_typed = {
    name: string;
    img: string;
    onTurnTyped: stmt_detail list;
    onClickTyped: stmt_detail list;
}

(*instructions for the setup of the board, rendered before any turns are completed*)

type setup_typed = stmt_detail list

(*conditions that end the game, specified as functions that return booleans. if not specified, we assume the functions indefinitely return 0.*)

type conditions_typed = {
    win_typed: stmt_detail list;
    lose_typed: stmt_detail list;
    draw_typed: stmt_detail list;
}

(type stmt_detail =
| Expr of expr_with_type
| Return of expr_with_type
| If of expr_with_type * stmt_detail list * stmt_detail list
| (* for varname surrounding (int x, int y))
| ForSurrounding of var_new * expr_with_type * expr_with_type * stmt_detail list
| (*expr varname in routing routine*)
| ForRow of var_new * expr_with_type * stmt_detail list
| (* for varname in col(int colnum) *)
| ForCol of var_new * expr_with_type * stmt_detail list
| (* for (int x, int y) surrounding (int x, int y))
| ForSurroundingCoords of var_new * var_new * expr_with_type * expr_with_type * stmt_detail list
| (*for varname(int min, int max)*)
| ForMinMax of var_new * expr_with_type * expr_with_type * stmt_detail list
| (*for varname(int min, int max), varName2(int min2, int max2) *)
| ForNested of var_new * expr_with_type * expr_with_type * var_new * expr_with_type * expr_with_type * stmt_detail list
| (*for (int x, int y) in board *)
| ForBoard of var_new * var_new * stmt_detail list
| ForAll of var_new * stmt_detail list
| Assign of var_ref * expr_with_type
| Where of var_new * expr_with_type
| Call of func_name * expr_with_type list)
open Sast
open Ast

(* environments *)
type symbol_table =
  { parent: symbol_table option;
    mutable variables: (Ast.typ -> string) list;
    mutable declFuncs: (Ast.typ -> (Ast.typ list)) list; (*list of usable functions: type, name, list of argument types*)
  }

let rec var_local (scope: symbol_table) name =
  let.name =
  List.exists(fun (_,s) -> s = name) scope.variables

let rec find_variable (scope: symbol_table) name =
  try List.find (fun (_,s) -> s = name) scope.variables
  with Not_found ->
    match scope.parent with
    | Some(parent) -> find_variable parent name
    | None -> raise Not_found

let rec find_function (scope: symbol_table) name =
  try List.find (fun (_,_,_,s) -> s = name) scope.declFuncs
  with Not_found ->
    match scope.parent with
    | Some(parent) -> find_function parent name
    | None -> raise Not_found

(* expr *)
let rec expr env =
  Ast.StringLiteral() -> Ast.Str, Sast.StringLiteral();
  Ast.BinOp (op, e1, e2) ->
    let e1 = expr env e1 and e2 = expr env e2 in
      (evaluates left and right hand expressions)
  let t1, t2 = e1 and t3 = e2 in
    (returns types of left and right expressions)
  let op type Fail expectedType =
    with Ast.String_of_op e ^ ^ is only for defined for "^expectedType"
      in Fail with
          | if t1, t2 = t1.Int then Ast.Int, Sast.BinOp(t1, t2, e1, e2)
          else op type Fail "int"
        | Ast.Less | Ast.Greater | Ast.Leq | Ast.Geq
          | if t1, t2 = t1.Int then Ast.Bool, Sast.BinOp(t1, t2, e1, e2)
          else op type Fail "int"
        | Equal | Not
          | if t1, t2 = t1.Int || t1, t2 = t1.Bool || t1, t2 = t1.String then Ast.Bool, Sast.BinOp(t1, t2, e1, e2)
          else op type Fail "int and int, or bool and bool, or string and string"
          | And | Or
          | if t1, t2 = t1.Bool then Ast.Bool, Sast.BinOp(t1, t2, e1, e2)
          else op type Fail "bool"
          | StrEqual
            | if t1, t2 = t1.String then Ast.Bool, Sast.BinOp(t1, t2, e1, e2)
            else op type Fail "string"
open Printf

let _ =
  let lexbuf = Lexing.from_channel stdin in
  let ast = Parser.program Scanner.token lexbuf in
  let prog = Semant.semcheck ast in
  Codegen.printprog prog;
Compiling a .gl file
gridlok.sh

./gridlok.sh [.gl file] [optional executable name]

- If executable name not given, defaults to test

```bash
#!/bin/bash
set -e
if [[ $# -eq 1 ]];
then
cd src
make > /dev/null
cd ../
src/gridlok < $1
gcc test.c -W -lSDL2 -lSDL2_image -o testrm test.c
elif [[ $# -eq 2 ]];
then
cd src
make > /dev/null
cd ../
src/gridlok < $1
gcc test.c -W -lSDL2 -lSDL2_image -o $2rm test.c
else
echo "Usage: ./gridlok.sh [.gl file] [executable name(optional)]"
fi
```
Testing
run_tests.sh

/run_tests.sh

- Checks tests/ directory and takes all files starting with test_ or fail_ and ending with .gl
- For test_ files:
  - Makes executable, then runs it and redirects stdout to a .out file of the same filename in tests/output/ directory
- For fail_ files:
  - Same as test_ files, but redirects stderr to the .out file
- Checks diff between the .out file in the tests/ directory and the .out file in the tests/output/ directory
- If no difference, deletes .diff file and returns OK message
- If there is a difference, keeps .diff file and returns FAILED message
- All test messages are logged in test.log file
Example Code
Tic-tac-toe

```
game()
  board{
    dimensions {3,3}
    image {"images/green_board.png"}
  }
  players {"black", "white"}
  turnOrder {"black", "white"}
  piece {
    name {"placeHolder"}
    image {"images/transparent.png"}
    onTurn{
      onClick{
        if(turn=="black"){
          changeType("blackPiece")
        } else{
          changeType("WhitePiece")
        }
      }
    }
  }
  drawCondition{
    def bool d(true)
    for all sp in board{
      if(sp.type=="placeHolder"){
        set d {false}
      } if(d){
        print("It's a draw...")
      }
    }
    return d
  }
  winCondition{
    def bool a(false) AND (board[0][0]==board[0][1] AND board[0][1]==board[0][2]) OR
    (board[0][0]==board[1][0] AND board[1][0]==board[2][0]) OR
    (board[0][2]==board[1][1] AND board[1][1]==board[2][2]) OR
    (board[2][0]==board[1][1] AND board[1][1]==board[2][2])
    def bool b( board[1][1]=="placeHolder" AND (board[0][0]==board[0][1] AND board[0][1]==board[0][2]) OR
      (board[0][0]==board[1][0] AND board[1][0]==board[1][1] AND board[1][1]==board[1][2]) OR
      (board[0][1]==board[1][0] AND board[0][1]==board[1][1] AND board[0][2]==board[1][2]) OR
      (board[1][1]==board[0][1] AND board[1][1]==board[1][2] AND board[1][2]==board[2][2])
    )
    )
    if a OR b OR c{
      if(turn == "black") {
        print("White wins!")
      } else {
        print("Black wins!")
      }
    }
    return a OR b OR c
  }
```
Minesweeper

```javascript
function setup()
{
    for (i, j) in board[
        def int count(0)
        if(board[i][j]==empty) {
            for p surrounding (i, j) {
                if(p.type=="mine")
                    set count (count + 1)
            }
            if(count==0) {
                place("zero", i, j)
                setVisibility(0, i, j)
            }
            if(count==1) {
                place("one", i, j)
                setVisibility(1, i, j)
            }
            if(count==2) {
                place("two", i, j)
                setVisibility(2, i, j)
            }
            if(count==3) {
                place("three", i, j)
                setVisibility(3, i, j)
            }
            if(count==4) {
                place("four", i, j)
                setVisibility(4, i, j)
            }
            if(count==5) {
                place("five", i, j)
                setVisibility(5, i, j)
            }
            if(count==6) {
                place("six", i, j)
                setVisibility(6, i, j)
            }
            if(count==7) {
                place("seven", i, j)
                setVisibility(7, i, j)
            }
            if(count==8) {
                place("eight", i, j)
                setVisibility(8, i, j)
            }
        }
    }
}
```
GridLok vs. Java

```java
import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.io.*;

public class Minesweeper extends JFrame implements MouseListener, ActionListener {
    private int rows = 18;
    private int columns = 18;
    private int mines = 18;
    private JLabel txtMinesRight;
    private JLabel txtTime;
    private JTextArea txtTest;
    private JButton btnStart;
    private Square[] buttons = new Square[rows][columns];
    private boolean started = false;
    private boolean finished = false;
    private int minesLeft = mines;
    private int fieldsLeft = rows * columns - minesLeft;
    private int currentTime = 0;
    private Timer timer;
    private JMenuItem itemNewGame;
    private JMenuItem itemFastest;
    private JMenuItem itemOut;
    private JMenuItem itemHelp;
    private JMenuItem itemAbout;
    private int bestScore;
    private JPanel field;

    public Minesweeper() {
        Container contentPane = getContentPane();
        getContentPane().setLayout(new BorderLayout());
        this.setTitle("Minesweeper");
    }

    /**
     * Main method to start the game
     */
    public static void main(String args[]) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        Minesweeper msw = new Minesweeper();
        msw.setJMenuBar(msw.myMenu());
        msw.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        msw.setVisible(true);
        msw.setResizable(false);
    }
}
```

543 Lines?! 130 😎
GridLok vs. Command Line C++

- 192 Lines?! 80 😎
- NO GUI?! WHAT IS THIS S&$%?

```cpp
/*{header|C++}*/
#include <windows.h>
#include <iostream>
#include <string>

//
using namespace std;

//
enum players { Computer, Human, Draw, None };
const int iWin[8][3] = {{ 0, 1, 2 }, { 3, 4, 5 }, { 6, 7, 8 }, { 0, 3, 6 }, { 1, 4, 7 }, { 2, 5, 8 }, { 0, 4, 8 }, { 2, 4, 6 }};

//

/*{out}*/ Computer plays 'X' and human plays '0'
<pre>
  1  |  2  |  X
  __|____|____
  X  |  5  |  6
  __|____|____
  7  |  0  |  9

Enter your move (1 - 9)
</pre>
```
Future Ideas

- Standard library of pieces, boards, games, movement functions, etc.
- Picking up pieces
- Improve final conditions
- Additional game settings (timers, etc…) 
- More GUI features
Lessons learned

- Start early
- Ask the TA questions frequently
- Start early
- Jane Street is not for us
- Start early
- Good thing we like each other
winCondition{
    def str grade{“A”}
    print{“Thank you!”}
    return {true}
}