GridLok

PLT Spring 2016 Final Project

```
players{
    “Julian Edwards”,
    “Laura Hu”,
    “Alice Hwang”,
    “Bryan Yu”
}
```

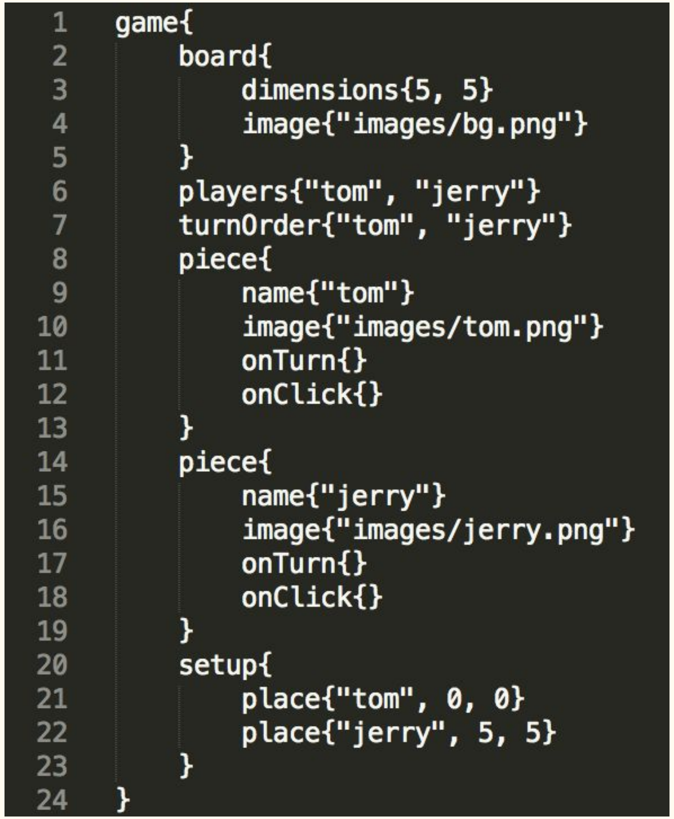
The Language

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:

game	visible	place
board	def	removePiece
dimensions	set	remove
image	if	click
players	else	changeType
turnOrder	for	break
piece	place	setVisibility
name	in	
onTurn	row	
onClick	col	
setup	surrounding	
winCondition	AND	
loseCondition	OR	
drawCondition	all	
return	print	

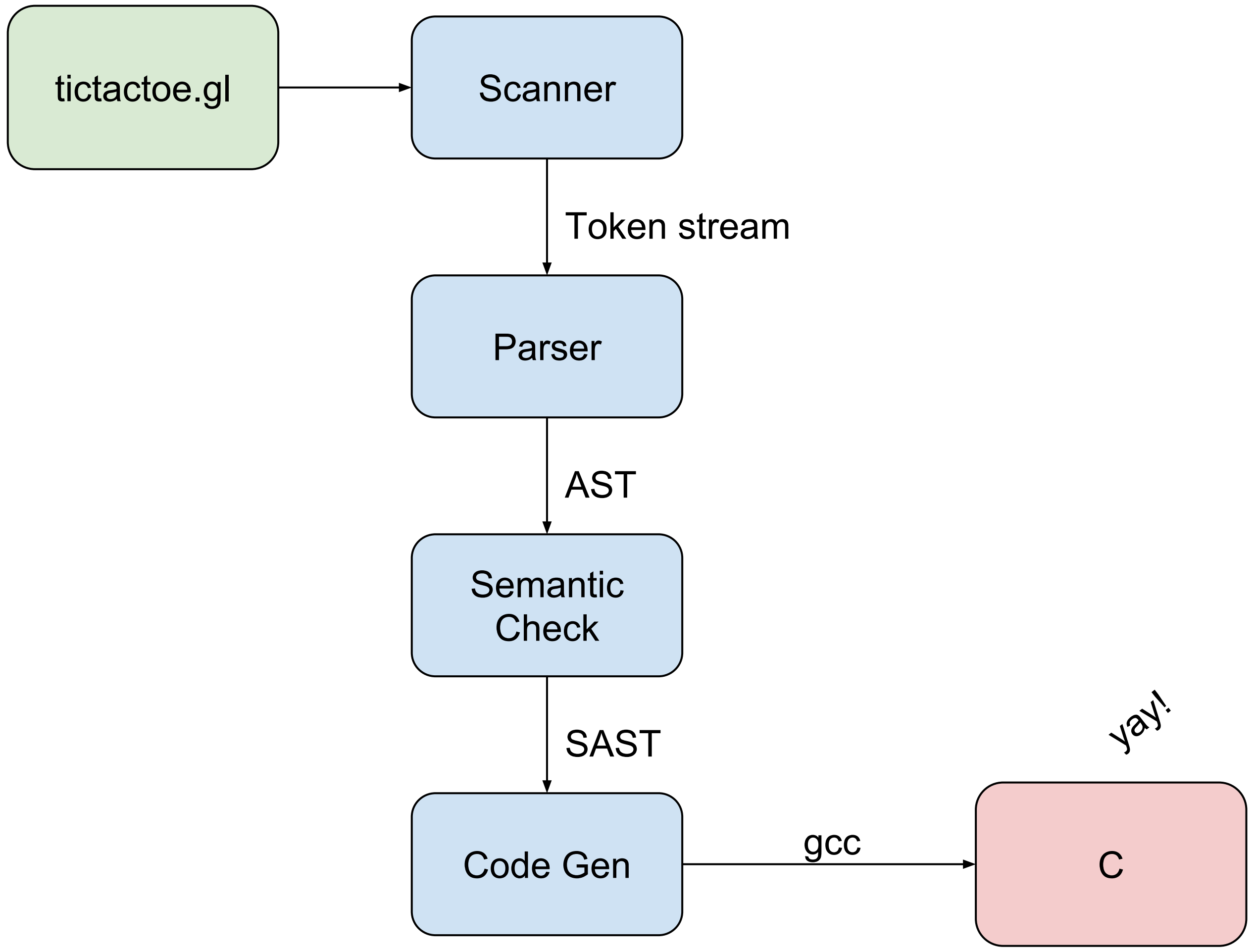


```
game{
    board{
        dimensions{5, 5}
        image{"images/bg.png"}
    }
    players{"tom", "jerry"}
    turnOrder{"tom", "jerry"}
    piece{
        name{"tom"}
        image{"images/tom.png"}
        onTurn{}
        onClick{}
    }
    piece{
        name{"jerry"}
        image{"images/jerry.png"}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

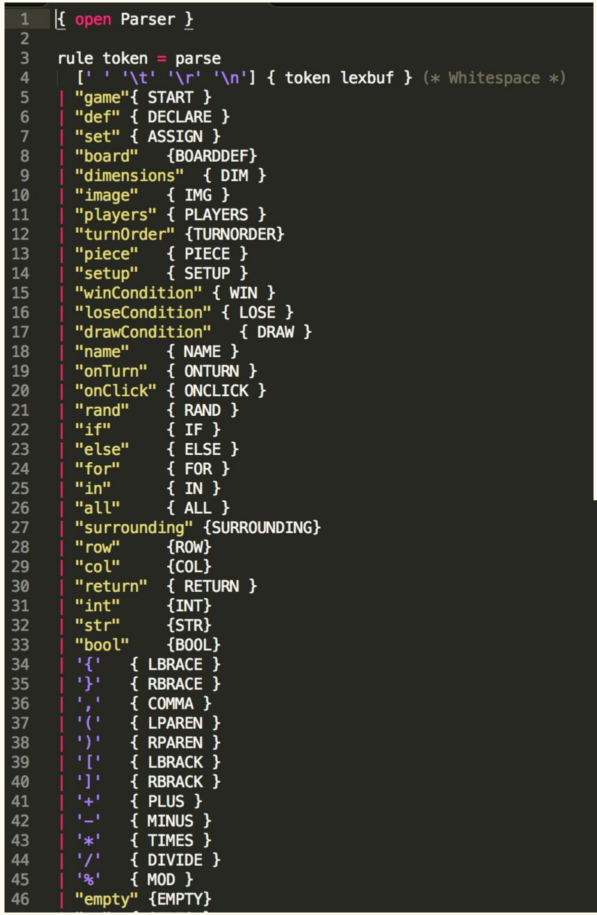
For loops:

Format:	Use:
`for pieceID in row(int){...}`	for pieces in a row r
`for pieceID in col(int){...}`	for pieces in a column c
`for pieceID surrounding(int,int){...}`	for pieces surrounding a coordinate (x,y)
`for (intID,intID) surrounding(int,int){...}`	for coordinates surrounding a coordinate (x,y)
`for intID(int x,int y){...}`	for loop over a range of ints (x,y)
`for(intID,intID) in board{...}`	for loop over all coordinates of a board
`for intID(int,int), intID(int,int){...}`	nested for loops for ranges of ints
`for all pieceID in board{...}`	for all spaces of a board

Source Code

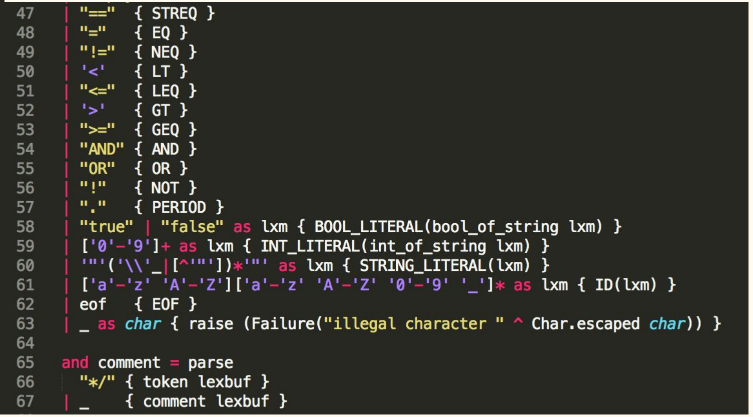

tictactoe.gl
Scanner
Token stream
Parser
AST
Semantic Check
SAST
Code Gen
gcc
C
yay!

scanner.mll



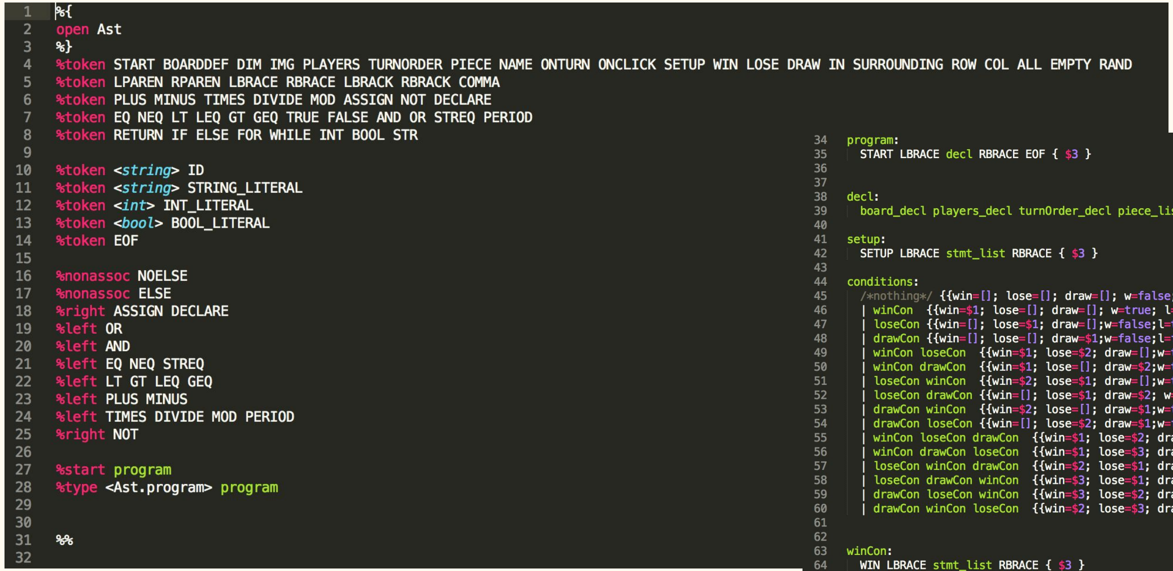
```
{ open Parser }

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "game"{ START }
| "def" { DECLARE }
| "set" { ASSIGN }
| "board"    {BOARDDEF}
| "dimensions"  { DIM }
| "image"    { IMG }
| "players" { PLAYERS }
| "turnOrder" {TURNORDER}
| "piece"    { PIECE }
| "setup"    { SETUP }
| "winCondition" { WIN }
| "loseCondition" { LOSE }
| "drawCondition"   { DRAW }
| "name"     { NAME }
| "onTurn"  { ONTURN }
| "onClick" { ONCLICK }
| "rand"     { RAND }
| "if"       { IF }
| "else"     { ELSE }
| "for"      { FOR }
| "in"       { IN }
| "all"      { ALL }
| "surrounding" {SURROUNDING}
| "row"      {ROW}
| "col"      {COL}
| "return"  { RETURN }
| "int"      {INT}
| "str"      {STR}
| "bool"     {BOOL}
| '{'   { LBRACE }
| '}'   { RBRACE }
| ','   { COMMA }
| '('   { LPAREN }
| ')'   { RPAREN }
| '['   { LBRACK }
| ']'   { RBRACK }
| '+'   { PLUS }
| '-'   { MINUS }
| '*'   { TIMES }
| '/'   { DIVIDE }
| '%'   { MOD }
| "empty" {EMPTY}
```

Total: 67 Lines

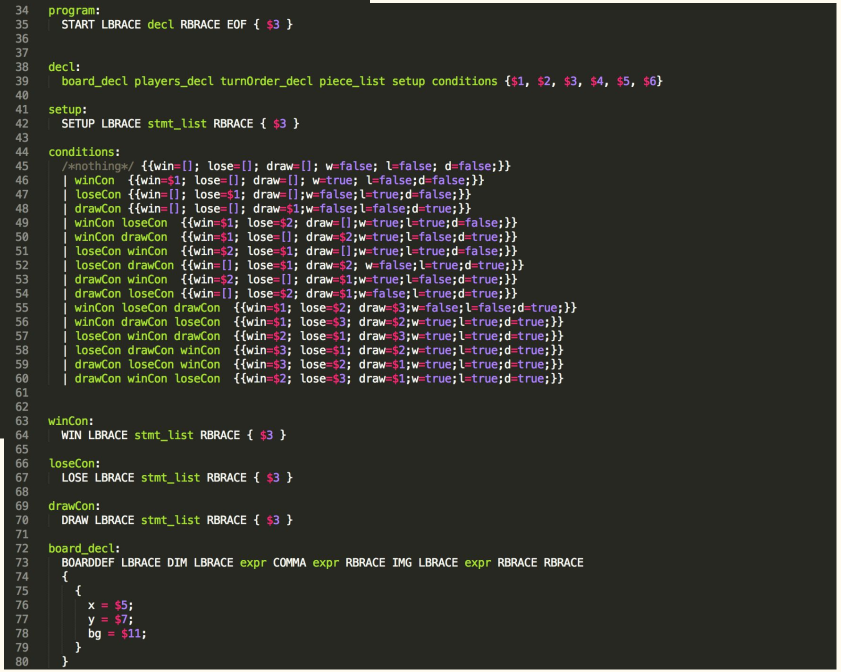


```
| "=="   { STREQ }
| "="    { EQ }
| "!="   { NEQ }
| '<'    { LT }
| "<="   { LEQ }
| '>'    { GT }
| ">="   { GEQ }
| "AND" { AND }
| "OR"  { OR }
| "!"    { NOT }
| "."    { PERIOD }
| "true" | "false" as lxm { BOOL_LITERAL(bool_of_string lxm) }
| ['0'-'9']+ as lxm { INT_LITERAL(int_of_string lxm) }
| '"'('\\'_|[^'"'])*'"' as lxm { STRING_LITERAL(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']* as lxm { ID(lxm) }
| eof    { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }

and comment = parse
  "*/" { token lexbuf }
| _     { comment lexbuf }
```

parser.mly



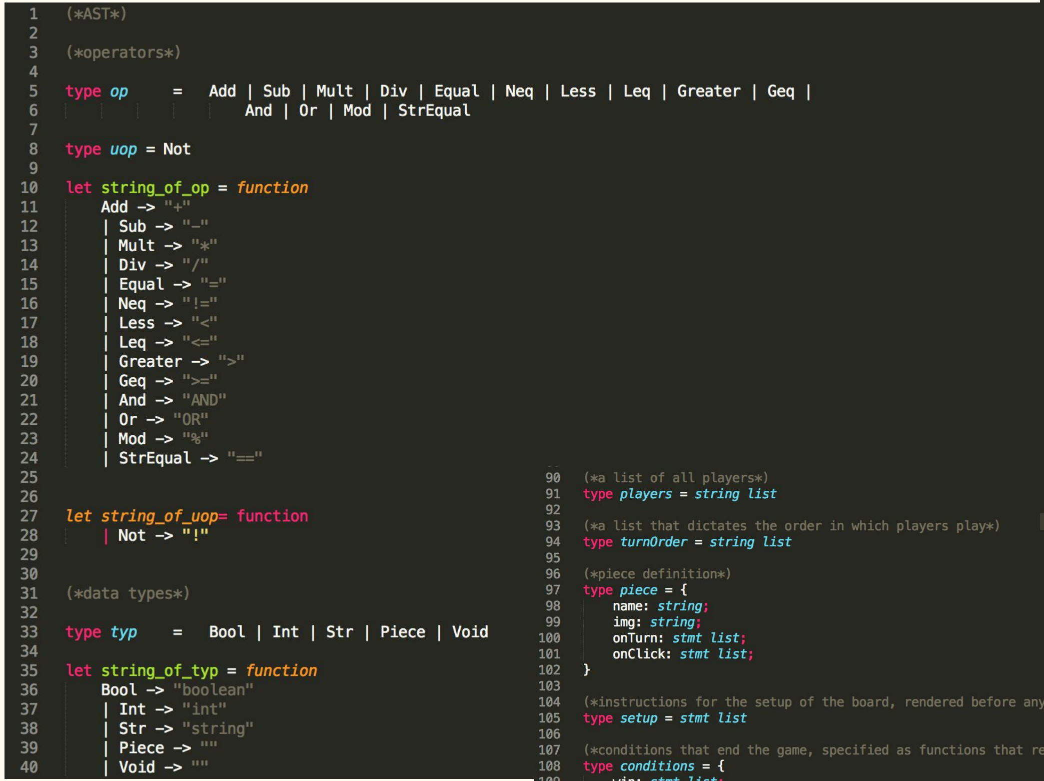
```
%{
open Ast
%}
%token START BOARDDEF DIM IMG PLAYERS TURNORDER PIECE NAME ONTURN ONCLICK SETUP WIN LOSE DRAW IN SURROUNDING ROW COL ALL EMPTY RAND
%token LPAREN RPAREN LBRACE RBRACE LBRACK RBRACK COMMA
%token PLUS MINUS TIMES DIVIDE MOD ASSIGN NOT DECLARE
%token EQ NEQ LT LEQ GT GEQ TRUE FALSE AND OR STREQ PERIOD
%token RETURN IF ELSE FOR WHILE INT BOOL STR

%token <string> ID
%token <string> STRING_LITERAL
%token <int> INT_LITERAL
%token <bool> BOOL_LITERAL
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN DECLARE
%left OR
%left AND
%left EQ NEQ STREQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MOD PERIOD
%right NOT

%start program
%type <Ast.program> program


%%

```



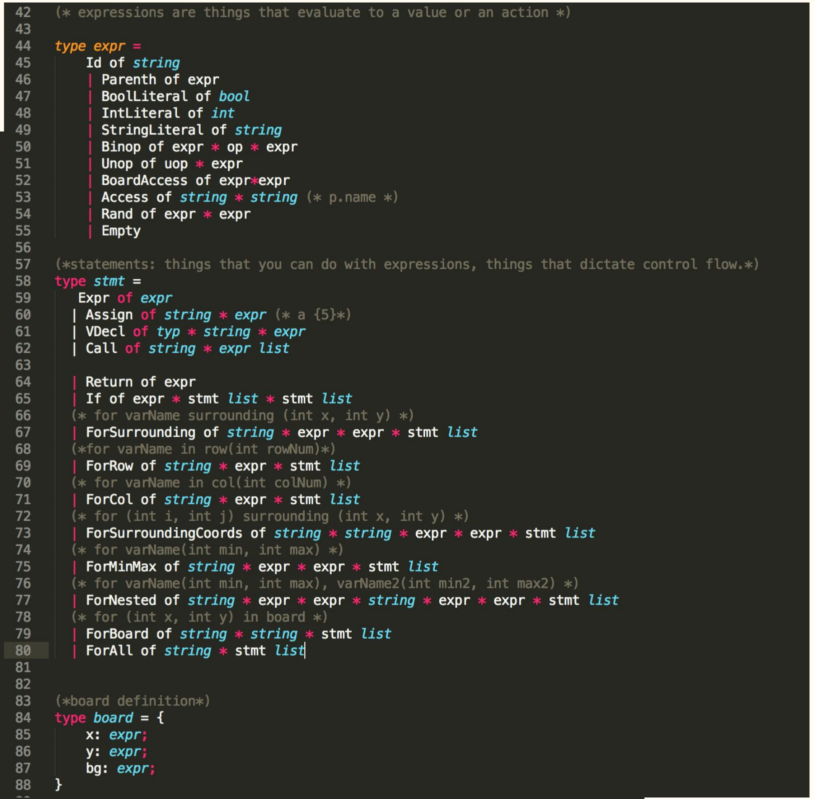
```
program:
  START LBRACE decl RBRACE EOF { $3 }


decl:
  board_decl players_decl turnOrder_decl piece_list setup conditions {$1, $2, $3, $4, $5, $6}

setup:
  SETUP LBRACE stmt_list RBRACE { $3 }

conditions:
  /*nothing*/ {{win=[]; lose=[]; draw=[]; w=false; l=false; d=false;}}
  | winCon  {{win=$1; lose=[]; draw=[]; w=true; l=false;d=false;}}
  | loseCon {{win=[]; lose=$1; draw=[];w=false;l=true;d=false;}}
  | drawCon {{win=[]; lose=[]; draw=$1;w=false;l=false;d=true;}}
  | winCon loseCon  {{win=$1; lose=$2; draw=[];w=true;l=true;d=false;}}
  | winCon drawCon  {{win=$1; lose=[]; draw=$2;w=true;l=false;d=true;}}
  | loseCon winCon  {{win=$2; lose=$1; draw=[];w=true;l=true;d=false;}}
  | loseCon drawCon {{win=[]; lose=$1; draw=$2; w=false;l=true;d=true;}}
  | drawCon winCon  {{win=$2; lose=[]; draw=$1;w=true;l=false;d=true;}}
  | drawCon loseCon {{win=[]; lose=$2; draw=$1;w=false;l=true;d=true;}}
  | winCon loseCon drawCon  {{win=$1; lose=$2; draw=$3;w=false;l=false;d=true;}}
  | winCon drawCon loseCon  {{win=$1; lose=$3; draw=$2;w=true;l=true;d=true;}}
  | loseCon winCon drawCon  {{win=$2; lose=$1; draw=$3;w=true;l=true;d=true;}}
  | loseCon drawCon winCon  {{win=$3; lose=$1; draw=$2;w=true;l=true;d=true;}}
  | drawCon loseCon winCon  {{win=$3; lose=$2; draw=$1;w=true;l=true;d=true;}}
  | drawCon winCon loseCon  {{win=$2; lose=$3; draw=$1;w=true;l=true;d=true;}}


winCon:
  WIN LBRACE stmt_list RBRACE { $3 }

loseCon:
  LOSE LBRACE stmt_list RBRACE { $3 }

drawCon:
  DRAW LBRACE stmt_list RBRACE { $3 }

board_decl:
  BOARDDEF LBRACE DIM LBRACE expr COMMA expr RBRACE IMG LBRACE expr RBRACE RBRACE
  {
    {
      x = $5;
      y = $7;
      bg = $11;
    }
  }
```

Total: 173 Lines

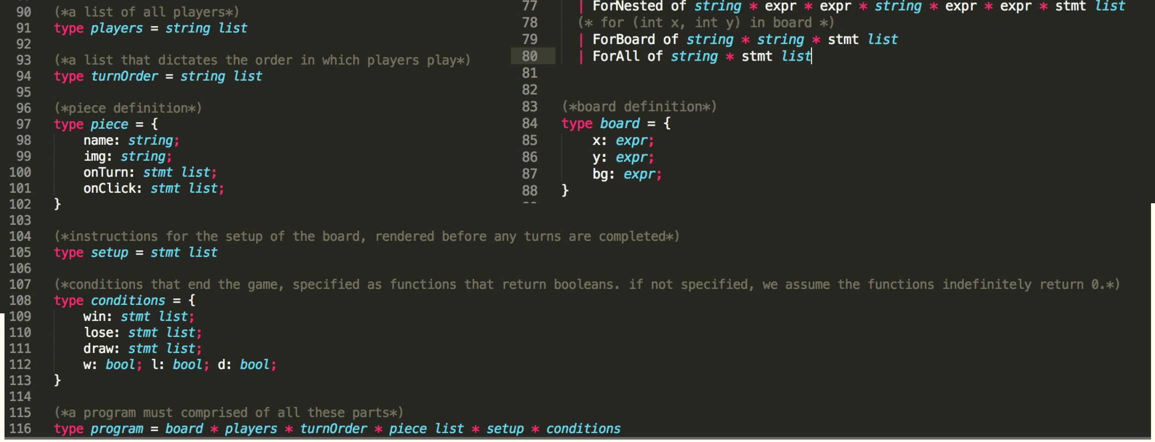
ast.ml



```ocaml
(*AST*)

(*operators*)

type op     =   Add | Sub | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq |
                    And | Or | Mod | StrEqual

type uop = Not

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "AND"
  | Or -> "OR"
  | Mod -> "%"
  | StrEqual -> "=="


let string_of_uop= function
  | Not -> "!"


(*data types*)

type typ    =   Bool | Int | Str | Piece | Void

let string_of_typ = function
    Bool -> "boolean"
  | Int -> "int"
  | Str -> "string"
  | Piece -> ""
  | Void -> ""
```



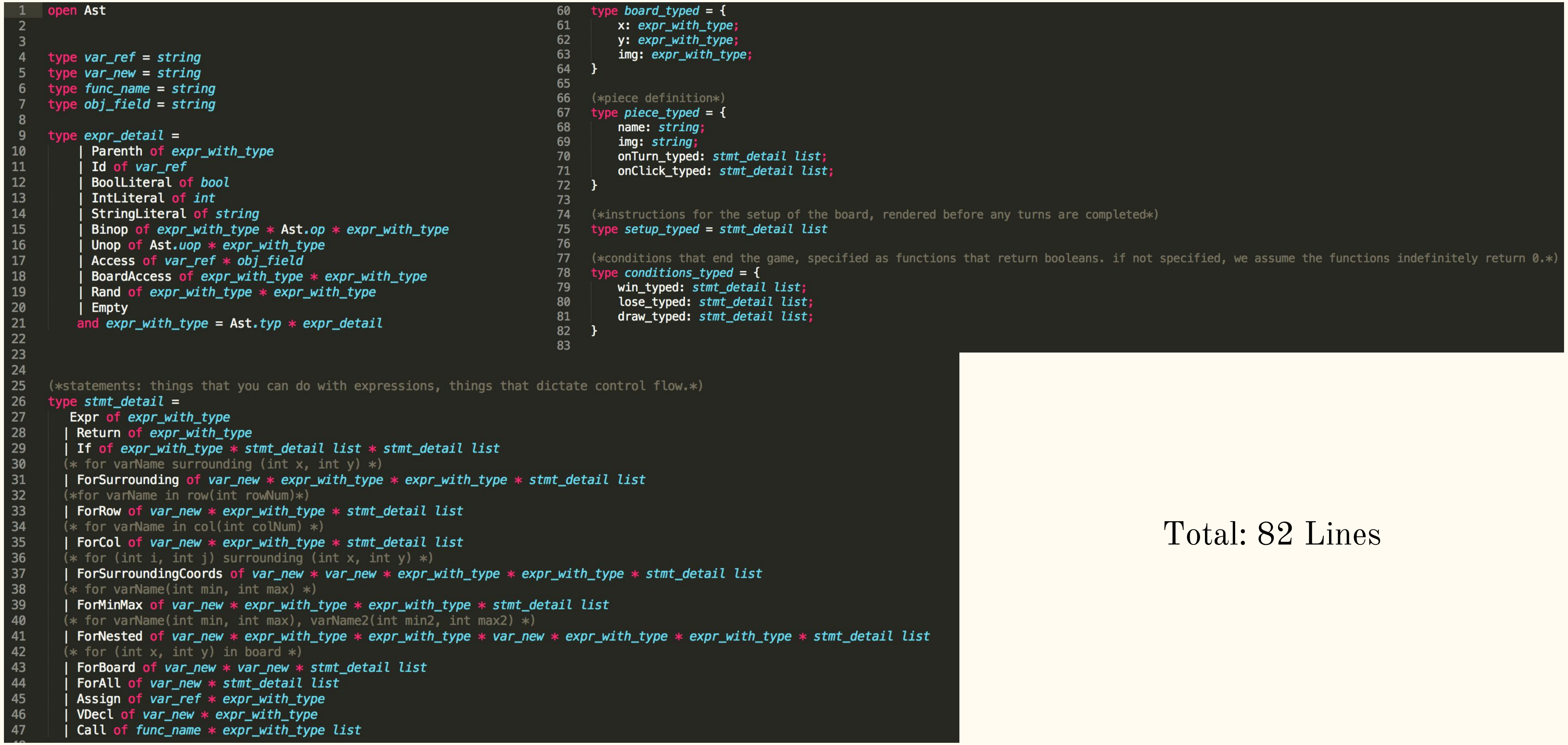
```ocaml
(* expressions are things that evaluate to a value or an action *)

type expr =
    Id of string
     | Parenth of expr
     | BoolLiteral of bool
     | IntLiteral of int
     | StringLiteral of string
     | Binop of expr * op * expr
     | Unop of uop * expr
     | BoardAccess of expr*expr
     | Access of string * string (* p.name *)
     | Rand of expr * expr
     | Empty

(*statements: things that you can do with expressions, things that dictate control flow.*)
type stmt =
   Expr of expr
  | Assign of string * expr (* a {5}*)
  | VDecl of typ * string * expr
  | Call of string * expr list

  | Return of expr
  | If of expr * stmt list * stmt list
  (* for varName surrounding (int x, int y) *)
  | ForSurrounding of string * expr * expr * stmt list
  (*for varName in row(int rowNum)*)
  | ForRow of string * expr * stmt list
  (* for varName in col(int colNum) *)
  | ForCol of string * expr * stmt list
  (* for (int i, int j) surrounding (int x, int y) *)
  | ForSurroundingCoords of string * string * expr * expr * stmt list
  (* for varName(int min, int max) *)
  | ForMinMax of string * expr * expr * stmt list
  (* for varName(int min, int max), varName2(int min2, int max2) *)
  | ForNested of string * expr * expr * string * expr * expr * stmt list
  (* for (int x, int y) in board *)
  | ForBoard of string * string * stmt list
  | ForAll of string * stmt list


(*board definition*)
type board = {
    x: expr;
    y: expr;
    bg: expr;
}
```



```ocaml
(*a list of all players*)
type players = string list

(*a list that dictates the order in which players play*)
type turnOrder = string list

(*piece definition*)
type piece = {
    name: string;
    img: string;
    onTurn: stmt list;
    onClick: stmt list;
}

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

(*conditions that end the game, specified as functions that return booleans. if not specified, we assume the functions indefinitely return 0.*)
type conditions = {
    win: stmt list;
    lose: stmt list;
    draw: stmt list;
    w: bool; l: bool; d: bool;
}

(*a program must comprised of all these parts*)
type program = board * players * turnOrder * piece list * setup * conditions
```

Total: 129 Lines

sast.ml

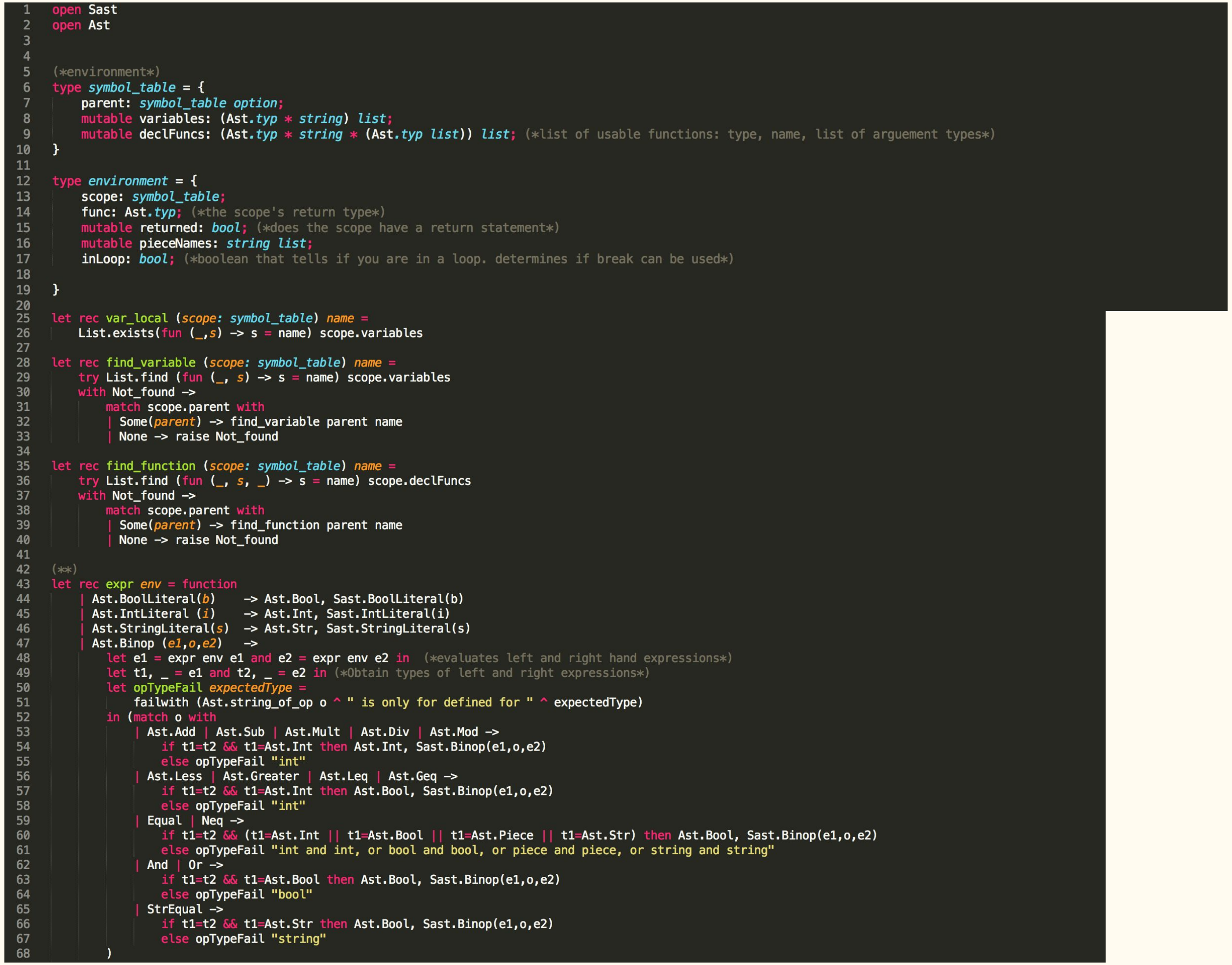


```ocaml
open Ast


type var_ref = string
type var_new = string
type func_name = string
type obj_field = string

type expr_detail =
      | Parenth 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.uop * 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



(*statements: things that you can do with expressions, things that dictate control flow.*)
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
  (*for varName in row(int rowNum)*)
  | 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 i, int j) 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
  | VDecl of var_new * expr_with_type
  | Call of func_name * expr_with_type list
```

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

(*piece definition*)
type piece_typed = {
    name: string;
    img: string;
    onTurn_typed: stmt_detail list;
    onClick_typed: 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;
}

```

Total: 82 Lines

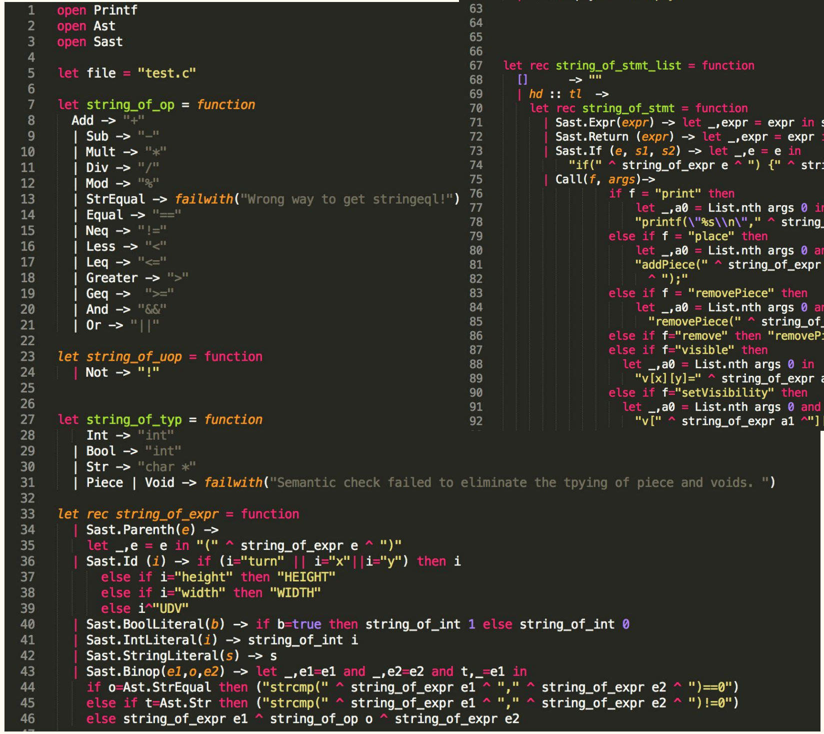
semant.ml



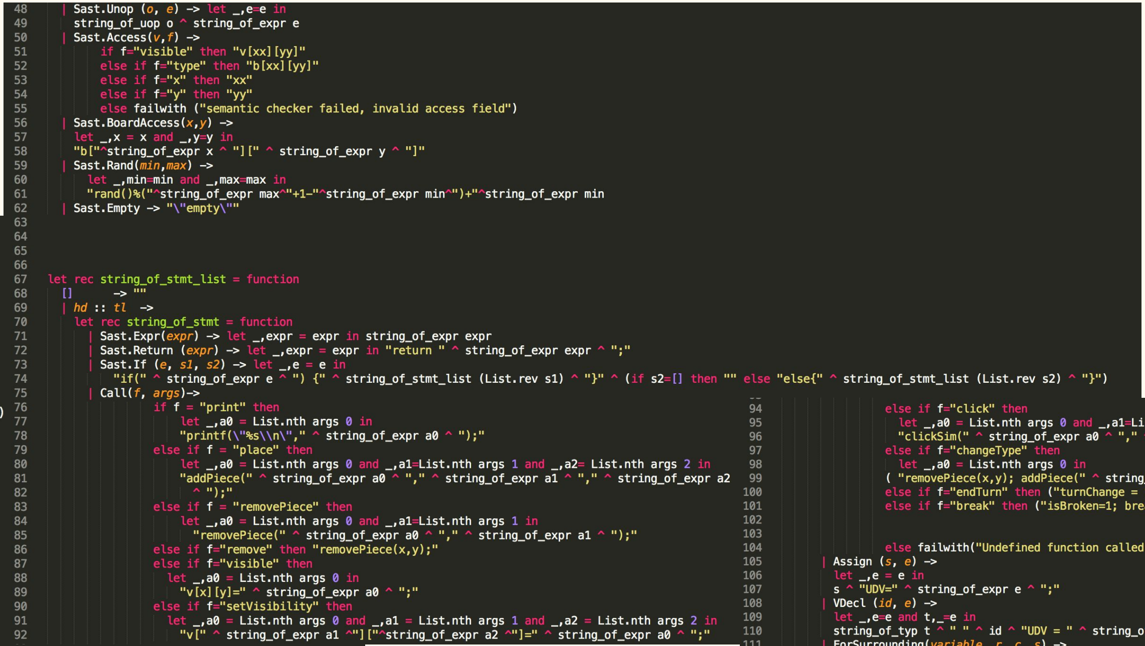
```ocaml
open Sast
open Ast


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

type environment = {
    scope: symbol_table;
    func: Ast.typ; (*the scope's return type*)
    mutable returned: bool; (*does the scope have a return statement*)
    mutable pieceNames: string list;
    inLoop: bool; (*boolean that tells if you are in a loop. determines if break can be used*)

}

let rec var_local (scope: symbol_table) 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

(**)
let rec expr env = function
    | Ast.BoolLiteral(b)    -> Ast.Bool, Sast.BoolLiteral(b)
    | Ast.IntLiteral (i)    -> Ast.Int, Sast.IntLiteral(i)
    | Ast.StringLiteral(s)  -> Ast.Str, Sast.StringLiteral(s)
    | Ast.Binop (e1,o,e2)   ->
        let e1 = expr env e1 and e2 = expr env e2 in  (*evaluates left and right hand expressions*)
        let t1, _ = e1 and t2, _ = e2 in (*Obtain types of left and right expressions*)
        let opTypeFail expectedType =
            failwith (Ast.string_of_op o ^ " is only for defined for " ^ expectedType)
        in (match o with
            | Ast.Add | Ast.Sub | Ast.Mult | Ast.Div | Ast.Mod ->
                if t1=t2 && t1=Ast.Int then Ast.Int, Sast.Binop(e1,o,e2)
                else opTypeFail "int"
            | Ast.Less | Ast.Greater | Ast.Leq | Ast.Geq ->
                if t1=t2 && t1=Ast.Int then Ast.Bool, Sast.Binop(e1,o,e2)
                else opTypeFail "int"
            | Equal | Neq ->
                if t1=t2 && (t1=Ast.Int || t1=Ast.Bool || t1=Ast.Piece || t1=Ast.Str) then Ast.Bool, Sast.Binop(e1,o,e2)
                else opTypeFail "int and int, or bool and bool, or piece and piece, or string and string"
            | And | Or ->
                if t1=t2 && t1=Ast.Bool then Ast.Bool, Sast.Binop(e1,o,e2)
                else opTypeFail "bool"
            | StrEqual ->
                if t1=t2 && t1=Ast.Str then Ast.Bool, Sast.Binop(e1,o,e2)
                else opTypeFail "string"
        )
```

Total: 328 Lines

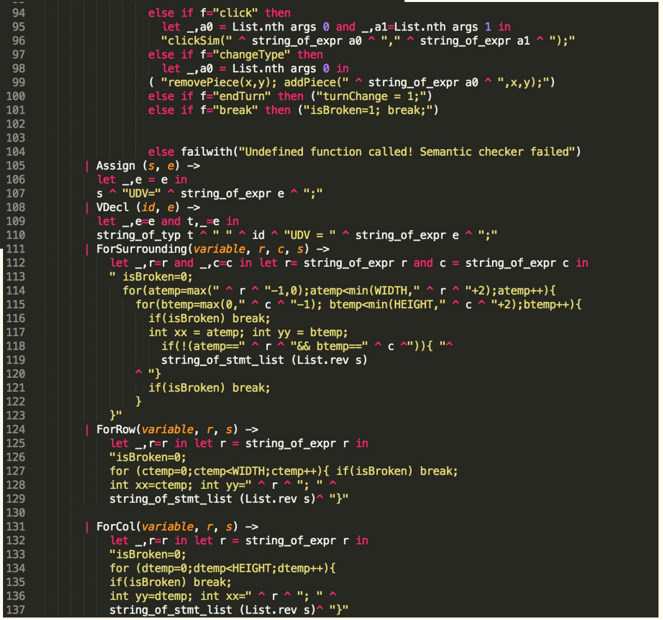
codegen.ml



```ocaml
open Printf
open Ast
open Sast

let file = "test.c"

let string_of_op = function
  Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | StrEqual -> failwith("Wrong way to get stringeql!")
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq ->  ">="
  | And -> "&&"
  | Or -> "||"

let string_of_uop = function
  | Not -> "!"


let string_of_typ = function
    Int -> "int"
  | Bool -> "int"
  | Str -> "char *"
  | Piece | Void -> failwith("Semantic check failed to eliminate the tpying of piece and voids. ")

let rec string_of_expr = function
  | Sast.Parenth(e) ->
    let _,e = e in "(" ^ string_of_expr e ^ ")"
  | Sast.Id (i) -> if (i="turn" || i="x"||i="y") then i
      else if i="height" then "HEIGHT"
      else if i="width" then "WIDTH"
      else i^"UDV"
  | Sast.BoolLiteral(b) -> if b=true then string_of_int 1 else string_of_int 0
  | Sast.IntLiteral(i) -> string_of_int i
  | Sast.StringLiteral(s) -> s
  | Sast.Binop(e1,o,e2) -> let _,e1=e1 and _,e2=e2 and t,_=e1 in
    if o=Ast.StrEqual then ("strcmp(" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ ")==0")
    else if t=Ast.Str then ("strcmp(" ^ string_of_expr e1 ^ "," ^ string_of_expr e2 ^ ")!=0")
    else string_of_expr e1 ^ string_of_op o ^ string_of_expr e2
```



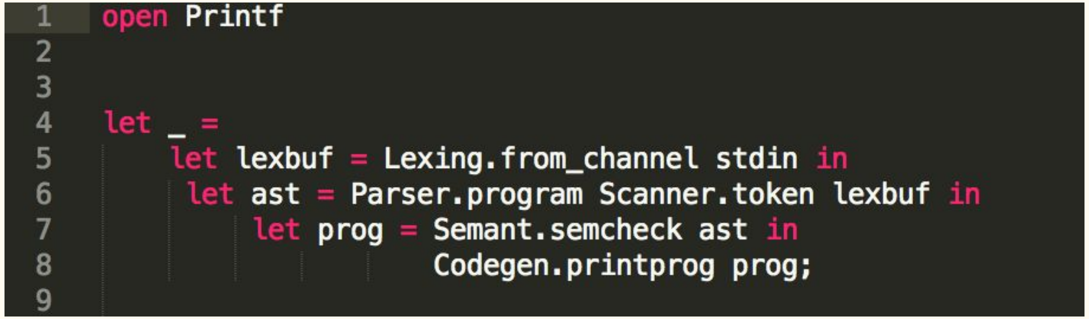
```ocaml
  | Sast.Unop (o, e) -> let _,e=e in
    string_of_uop o ^ string_of_expr e
  | Sast.Access(v,f) ->
        if f="visible" then "v[xx][yy]"
        else if f="type" then "b[xx][yy]"
        else if f="x" then "xx"
        else if f="y" then "yy"
        else failwith ("semantic checker failed, invalid access field")
  | Sast.BoardAccess(x,y) ->
    let _,x = x and _,y=y in
    "b["^string_of_expr x ^ "][" ^ string_of_expr y ^ "]"
  | Sast.Rand(min,max) ->
      let _,min=min and _,max=max in
      "rand()%("^string_of_expr max^"+1-"^string_of_expr min^")+"^string_of_expr min
  | Sast.Empty -> "\"empty\""




let rec string_of_stmt_list = function
  []      -> ""
  | hd :: tl  ->
    let rec string_of_stmt = function
      | Sast.Expr(expr) -> let _,expr = expr in string_of_expr expr
      | Sast.Return (expr) -> let _,expr = expr in "return " ^ string_of_expr expr ^ ";"
      | Sast.If (e, s1, s2) -> let _,e = e in
          "if(" ^ string_of_expr e ^ ") {" ^ string_of_stmt_list (List.rev s1) ^ "}" ^ (if s2=[] then "" else "else{" ^ string_of_stmt_list (List.rev s2) ^ "}")
      | Call(f, args)->
                  if f = "print" then
                      let _,a0 = List.nth args 0 in
                      "printf(\"%s\\n\"," ^ string_of_expr a0 ^ ");"
                  else if f = "place" then
                      let _,a0 = List.nth args 0 and _,a1=List.nth args 1 and _,a2= List.nth args 2 in
                      "addPiece(" ^ string_of_expr a0 ^ "," ^ string_of_expr a1 ^ "," ^ string_of_expr a2
                        ^ ");"
                  else if f = "removePiece" then
                      let _,a0 = List.nth args 0 and _,a1=List.nth args 1 in
                        "removePiece(" ^ string_of_expr a0 ^ "," ^ string_of_expr a1 ^ ");"
                  else if f="remove" then "removePiece(x,y);"
                  else if f="visible" then
                    let _,a0 = List.nth args 0 in
                      "v[x][y]=" ^ string_of_expr a0 ^ ";"
                  else if f="setVisibility" then
                    let _,a0 = List.nth args 0 and _,a1 = List.nth args 1 and _,a2 = List.nth args 2 in
                      "v[" ^ string_of_expr a1 ^"]["^string_of_expr a2 ^"]=" ^ string_of_expr a0 ^ ";"
```



```ocaml
                  else if f="click" then
                    let _,a0 = List.nth args 0 and _,a1=List.nth args 1 in
                    "clickSim(" ^ string_of_expr a0 ^ "," ^ string_of_expr a1 ^ ");"
                  else if f="changeType" then
                    let _,a0 = List.nth args 0 in
                  ( "removePiece(x,y); addPiece(" ^ string_of_expr a0 ^ ",x,y);")
                  else if f="endTurn" then ("turnChange = 1;")
                  else if f="break" then ("isBroken=1; break;")


                  else failwith("Undefined function called! Semantic checker failed")
      | Assign (s, e) ->
        let _,e = e in
        s ^ "UDV=" ^ string_of_expr e ^ ";"
      | VDecl (id, e) ->
        let _,e=e and t,_=e in
        string_of_typ t ^ " " ^ id ^ "UDV = " ^ string_of_expr e ^ ";"
      | ForSurrounding(variable, r, c, s) ->
          let _,r=r and _,c=c in let r= string_of_expr r and c = string_of_expr c in
          " isBroken=0;
            for(atemp=max(" ^ r ^ "-1,0);atemp<min(WIDTH," ^ r ^ "+2);atemp++){
              for(btemp=max(0," ^ c ^ "-1); btemp<min(HEIGHT," ^ c ^ "+2);btemp++){
                if(isBroken) break;
                int xx = atemp; int yy = btemp;
                  if(!(atemp==" ^ r ^ "&& btemp==" ^ c ^")){ "^
                  string_of_stmt_list (List.rev s)
              ^ "}
                if(isBroken) break;
              }
          }"
      | ForRow(variable, r, s) ->
          let _,r=r in let r = string_of_expr r in
          "isBroken=0;
          for (ctemp=0;ctemp<WIDTH;ctemp++){ if(isBroken) break;
          int xx=ctemp; int yy=" ^ r ^ "; " ^
          string_of_stmt_list (List.rev s)^ "}"

      | ForCol(variable, r, s) ->
          let _,r=r in let r = string_of_expr r in
          "isBroken=0;
          for (dtemp=0;dtemp<HEIGHT;dtemp++){
          if(isBroken) break;
          int yy=dtemp; int xx=" ^ r ^ "; " ^
          string_of_stmt_list (List.rev s)^ "}"
```

Total: 549 Lines

gridlok.ml



```ocaml
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;

```

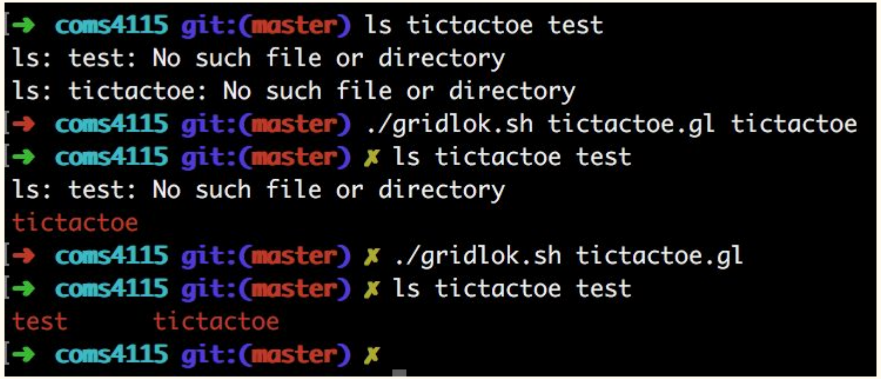
Total: 8 Lines

Compiling a .gl file

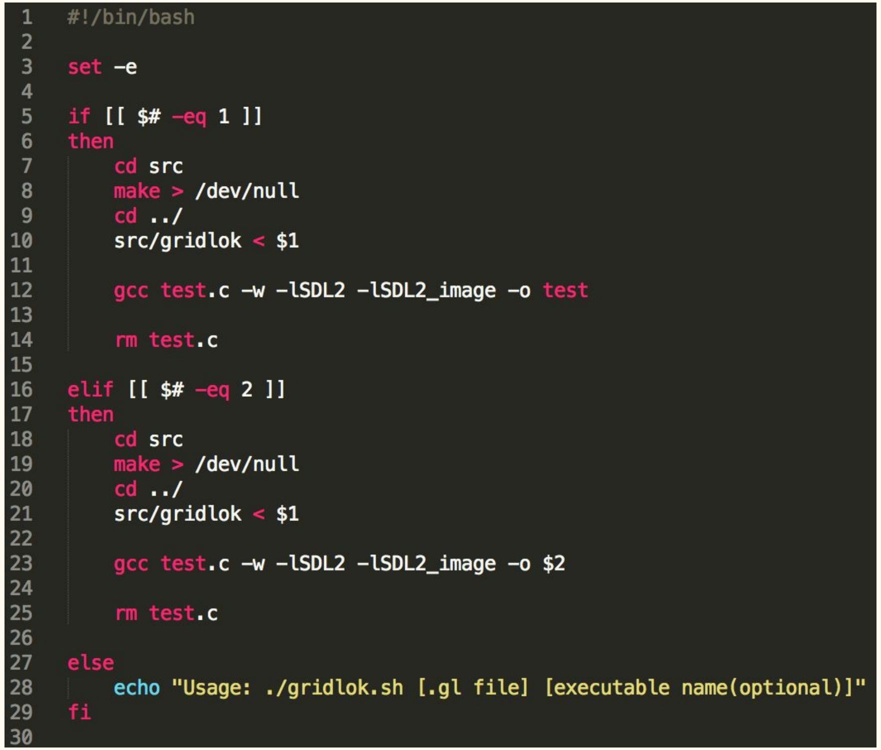
gridlok.sh

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

- If executable name not given, defaults to test



```
→ coms4115 git:(master) ls tictactoe test
ls: test: No such file or directory
ls: tictactoe: No such file or directory
→ coms4115 git:(master) ./gridlok.sh tictactoe.gl tictactoe
→ coms4115 git:(master) ✗ ls tictactoe test
ls: test: No such file or directory
tictactoe
→ coms4115 git:(master) ✗ ./gridlok.sh tictactoe.gl
→ coms4115 git:(master) ✗ ls tictactoe test
test      tictactoe
→ coms4115 git:(master) ✗
```



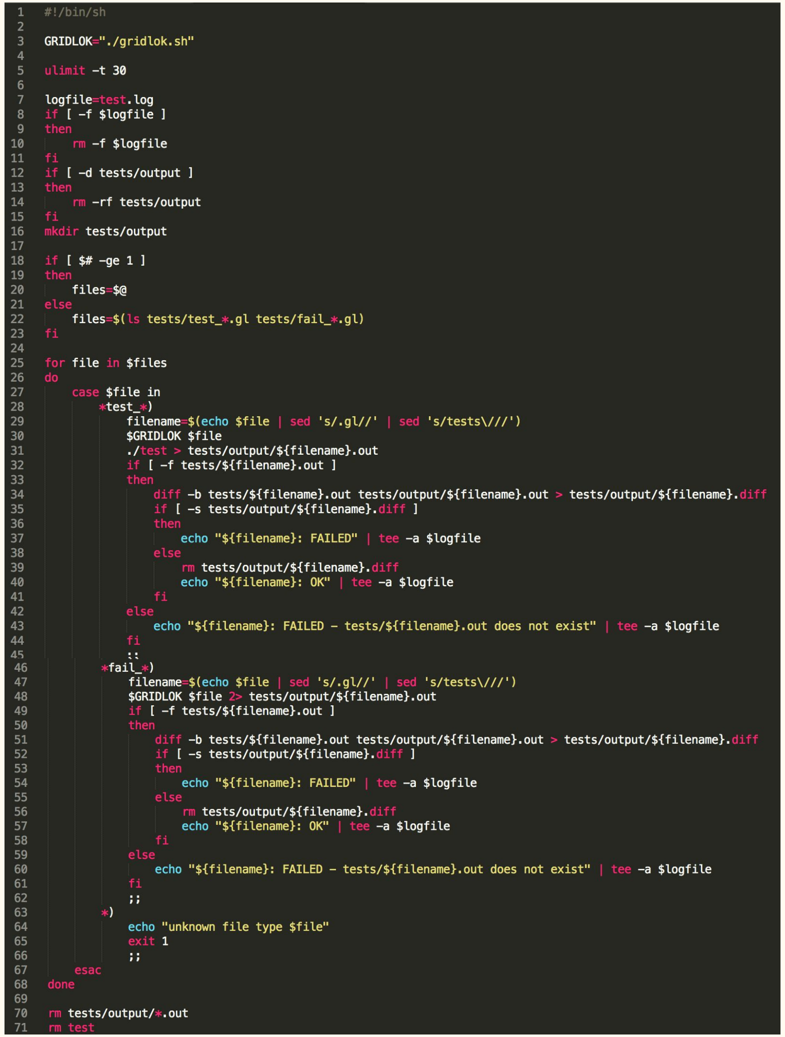
```
#!/bin/bash

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`

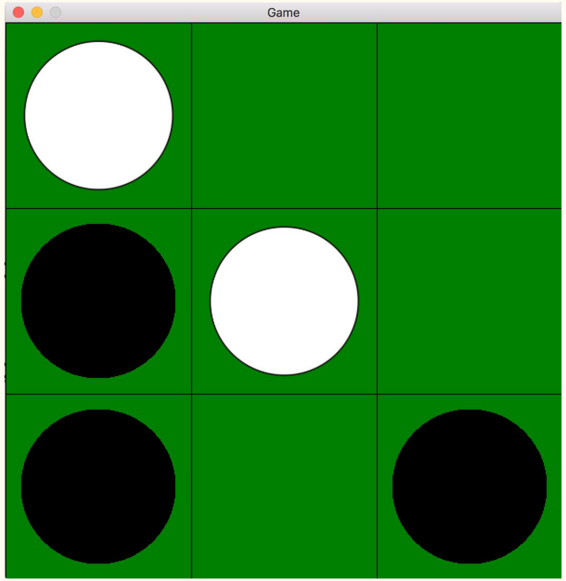
- Checks tests/ directory and takes all files starting with test_ or fail_ and ending with .gl
- For test_ files:
  - Makes executable, then runs it and redirects stdout to a .out file of the same filename in tests/output/ directory
- For fail_ files:
  - Same as test_ files, but redirects stderr to 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

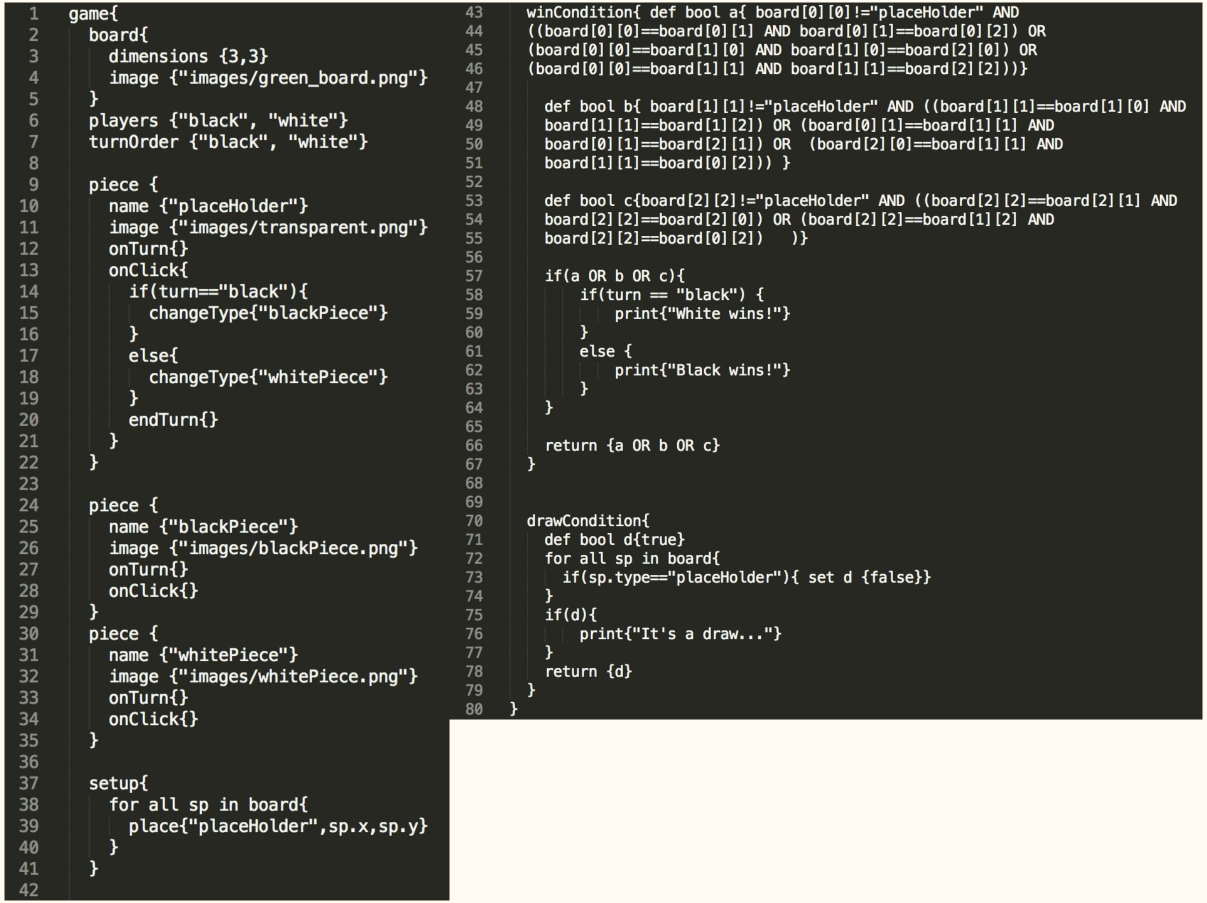


```sh
#!/bin/sh

GRIDLOK="./gridlok.sh"

ulimit -t 30

logfile=test.log
if [ -f $logfile ]
then
    rm -f $logfile
fi
if [ -d tests/output ]
then
    rm -rf tests/output
fi
mkdir tests/output

if [ $# -ge 1 ]
then
    files=$@
else
    files=$(ls tests/test_*.gl tests/fail_*.gl)
fi

for file in $files
do
    case $file in
        *test_*)
            filename=$(echo $file | sed 's/.gl//' | sed 's/tests\///')
            $GRIDLOK $file
            ./test > tests/output/${filename}.out
            if [ -f tests/${filename}.out ]
            then
                diff -b tests/${filename}.out tests/output/${filename}.out > tests/output/${filename}.diff
                if [ -s tests/output/${filename}.diff ]
                then
                    echo "${filename}: FAILED" | tee -a $logfile
                else
                    rm tests/output/${filename}.diff
                    echo "${filename}: OK" | tee -a $logfile
                fi
            else
                echo "${filename}: FAILED - tests/${filename}.out does not exist" | tee -a $logfile
            fi
            ;;
        *fail_*)
            filename=$(echo $file | sed 's/.gl//' | sed 's/tests\///')
            $GRIDLOK $file 2> tests/output/${filename}.out
            if [ -f tests/${filename}.out ]
            then
                diff -b tests/${filename}.out tests/output/${filename}.out > tests/output/${filename}.diff
                if [ -s tests/output/${filename}.diff ]
                then
                    echo "${filename}: FAILED" | tee -a $logfile
                else
                    rm tests/output/${filename}.diff
                    echo "${filename}: OK" | tee -a $logfile
                fi
            else
                echo "${filename}: FAILED - tests/${filename}.out does not exist" | tee -a $logfile
            fi
            ;;
        *)
            echo "unknown file type $file"
            exit 1
            ;;
    esac
done

rm tests/output/*.out
rm test
```

Example Code

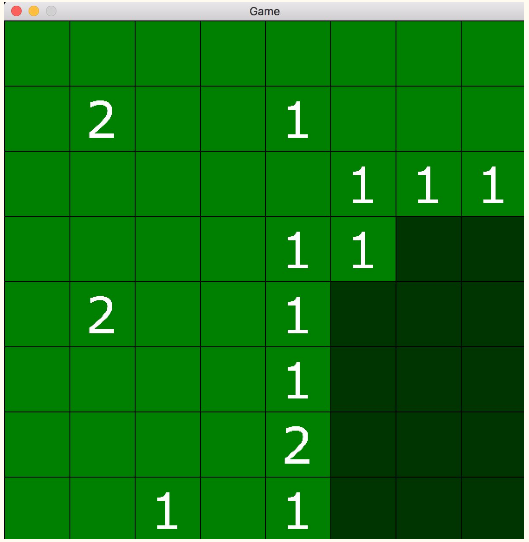
Tic-tac-toe

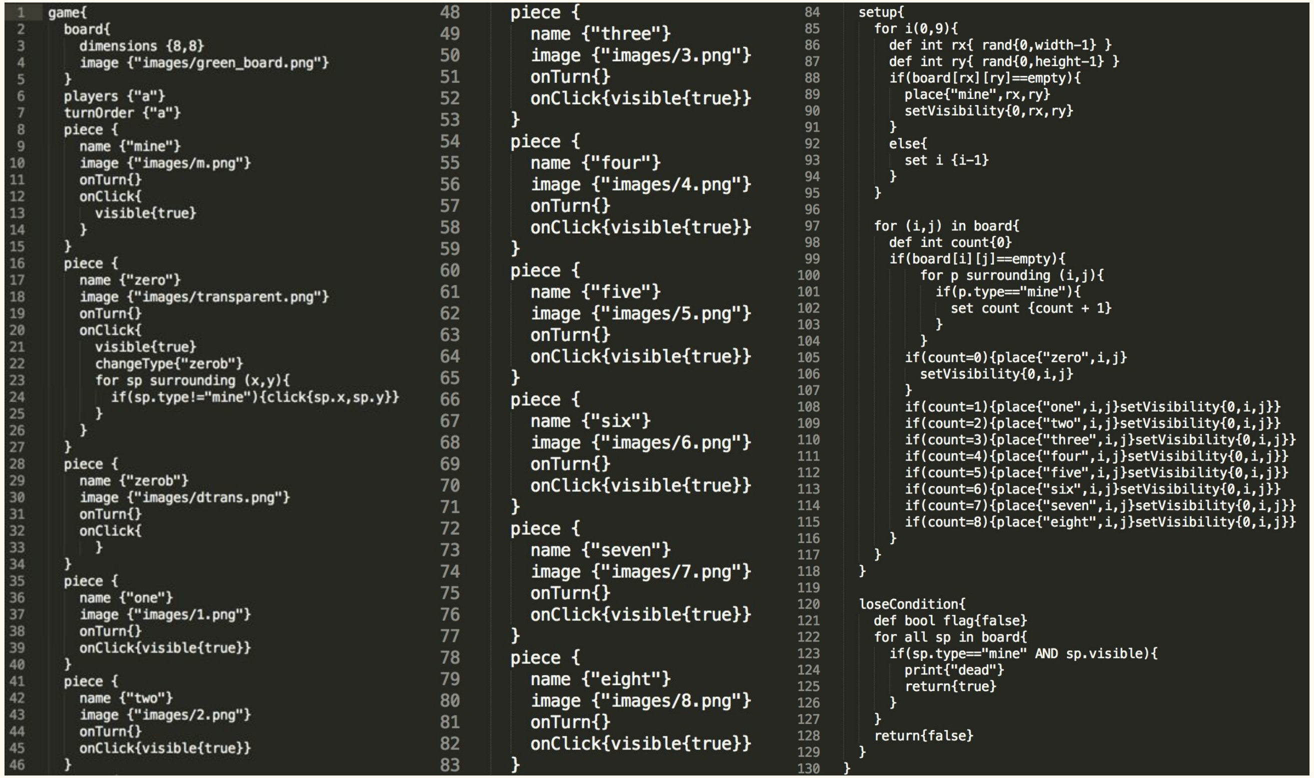




```
game{
  board{
    dimensions {3,3}
    image {"images/green_board.png"}
  }
  players {"black", "white"}
  turnOrder {"black", "white"}

  piece {
    name {"placeHolder"}
    image {"images/transparent.png"}
    onTurn{}
    onClick{
      if(turn=="black"){
        changeType{"blackPiece"}
      }
      else{
        changeType{"whitePiece"}
      }
      endTurn{}
    }
  }

  piece {
    name {"blackPiece"}
    image {"images/blackPiece.png"}
    onTurn{}
    onClick{}
  }
  piece {
    name {"whitePiece"}
    image {"images/whitePiece.png"}
    onTurn{}
    onClick{}
  }

  setup{
    for all sp in board{
      place{"placeHolder",sp.x,sp.y}
    }
  }

  winCondition{ def bool a{ board[0][0]!="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[2][0]) OR
  (board[0][0]==board[1][1] AND board[1][1]==board[2][2]))}

    def bool b{ board[1][1]!="placeHolder" AND ((board[1][1]==board[1][0] AND
    board[1][1]==board[1][2]) OR (board[0][1]==board[1][1] AND
    board[0][1]==board[2][1]) OR  (board[2][0]==board[1][1] AND
    board[1][1]==board[0][2])) }

    def bool c{board[2][2]!="placeHolder" AND ((board[2][2]==board[2][1] AND
    board[2][2]==board[2][0]) OR (board[2][2]==board[1][2] AND
    board[2][2]==board[0][2])    )}

    if(a OR b OR c){
        if(turn == "black") {
            print{"White wins!"}
        }
        else {
            print{"Black wins!"}
        }
    }

    return {a OR b OR c}
  }


  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}
  }
}
```

Minesweeper

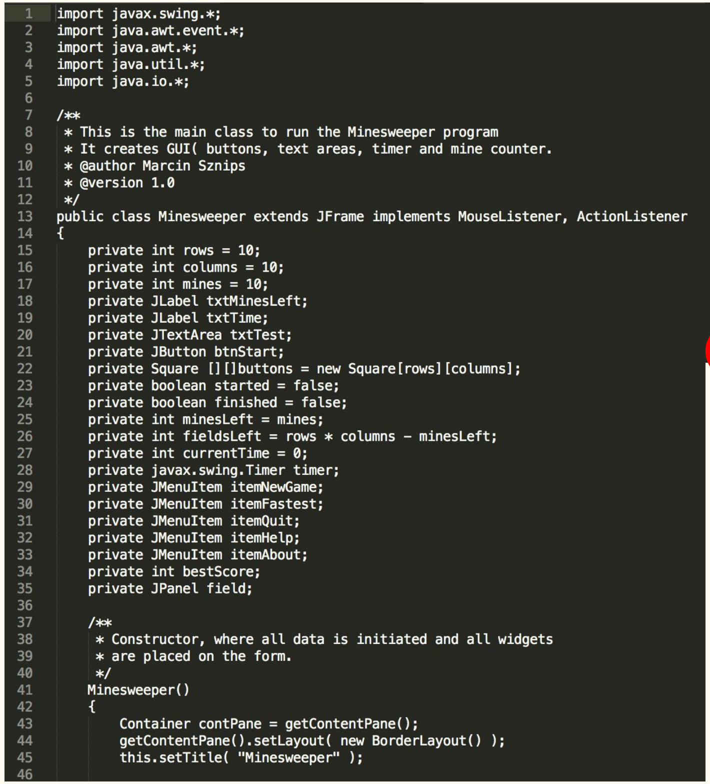




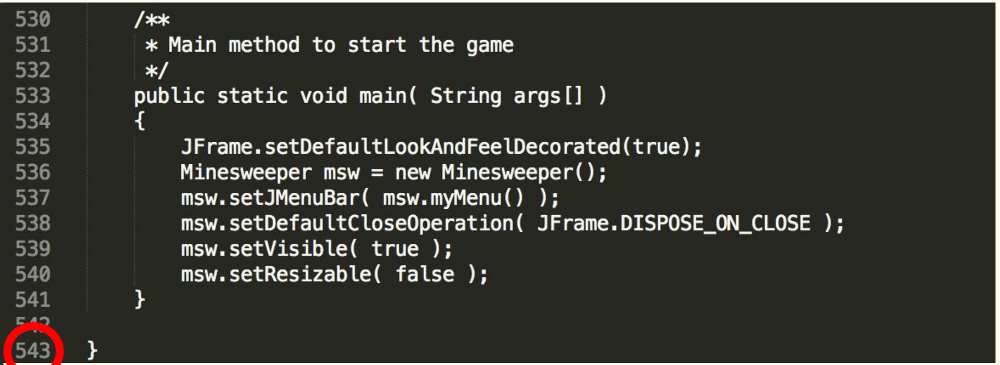
```
game{
  board{
    dimensions {8,8}
    image {"images/green_board.png"}
  }
  players {"a"}
  turnOrder {"a"}
  piece {
    name {"mine"}
    image {"images/m.png"}
    onTurn{}
    onClick{
      visible{true}
    }
  }
  piece {
    name {"zero"}
    image {"images/transparent.png"}
    onTurn{}
    onClick{
      visible{true}
      changeType{"zerob"}
      for sp surrounding (x,y){
        if(sp.type!="mine"){click{sp.x,sp.y}}
      }
    }
  }
  piece {
    name {"zerob"}
    image {"images/dtrans.png"}
    onTurn{}
    onClick{
      }
  }
  piece {
    name {"one"}
    image {"images/1.png"}
    onTurn{}
    onClick{visible{true}}
  }
  piece {
    name {"two"}
    image {"images/2.png"}
    onTurn{}
    onClick{visible{true}}
  }
```

```
  piece {
    name {"three"}
    image {"images/3.png"}
    onTurn{}
    onClick{visible{true}}
  }
  piece {
    name {"four"}
    image {"images/4.png"}
    onTurn{}
    onClick{visible{true}}
  }
  piece {
    name {"five"}
    image {"images/5.png"}
    onTurn{}
    onClick{visible{true}}
  }
  piece {
    name {"six"}
    image {"images/6.png"}
    onTurn{}
    onClick{visible{true}}
  }
  piece {
    name {"seven"}
    image {"images/7.png"}
    onTurn{}
    onClick{visible{true}}
  }
  piece {
    name {"eight"}
    image {"images/8.png"}
    onTurn{}
    onClick{visible{true}}
  }
  setup{
    for i(0,9){
      def int rx{ rand{0,width-1} }
      def int ry{ rand{0,height-1} }
      if(board[rx][ry]==empty){
        place{"mine",rx,ry}
        setVisibility{0,rx,ry}
      }
      else{
        set i {i-1}
      }
    }

    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{0,i,j}}
        if(count=2){place{"two",i,j}setVisibility{0,i,j}}
        if(count=3){place{"three",i,j}setVisibility{0,i,j}}
        if(count=4){place{"four",i,j}setVisibility{0,i,j}}
        if(count=5){place{"five",i,j}setVisibility{0,i,j}}
        if(count=6){place{"six",i,j}setVisibility{0,i,j}}
        if(count=7){place{"seven",i,j}setVisibility{0,i,j}}
        if(count=8){place{"eight",i,j}setVisibility{0,i,j}}
      }
    }
  }

  loseCondition{
    def bool flag{false}
    for all sp in board{
      if(sp.type=="mine" AND sp.visible){
        print{"dead"}
        return{true}
      }
    }
    return{false}
  }
}
```

GridLok vs. Java



```
 1  import javax.swing.*;
 2  import java.awt.event.*;
 3  import java.awt.*;
 4  import java.util.*;
 5  import java.io.*;
 6
 7  /**
 8   * This is the main class to run the Minesweeper program
 9   * It creates GUI( buttons, text areas, timer and mine counter.
10   * @author Marcin Sznips
11   * @version 1.0
12   */
13  public class Minesweeper extends JFrame implements MouseListener, ActionListener
14  {
15      private int rows = 10;
16      private int columns = 10;
17      private int mines = 10;
18      private JLabel txtMinesLeft;
19      private JLabel txtTime;
20      private JTextArea txtTest;
21      private JButton btnStart;
22      private Square [][]buttons = new Square[rows][columns];
23      private boolean started = false;
24      private boolean finished = false;
25      private int minesLeft = mines;
26      private int fieldsLeft = rows * columns - minesLeft;
27      private int currentTime = 0;
28      private javax.swing.Timer timer;
29      private JMenuItem itemNewGame;
30      private JMenuItem itemFastest;
31      private JMenuItem itemQuit;
32      private JMenuItem itemHelp;
33      private JMenuItem itemAbout;
34      private int bestScore;
35      private JPanel field;
36
37      /**
38       * Constructor, where all data is initiated and all widgets
39       * are placed on the form.
40       */
41      Minesweeper()
42      {
43          Container contPane = getContentPane();
44          getContentPane().setLayout( new BorderLayout() );
45          this.setTitle( "Minesweeper" );
46
```

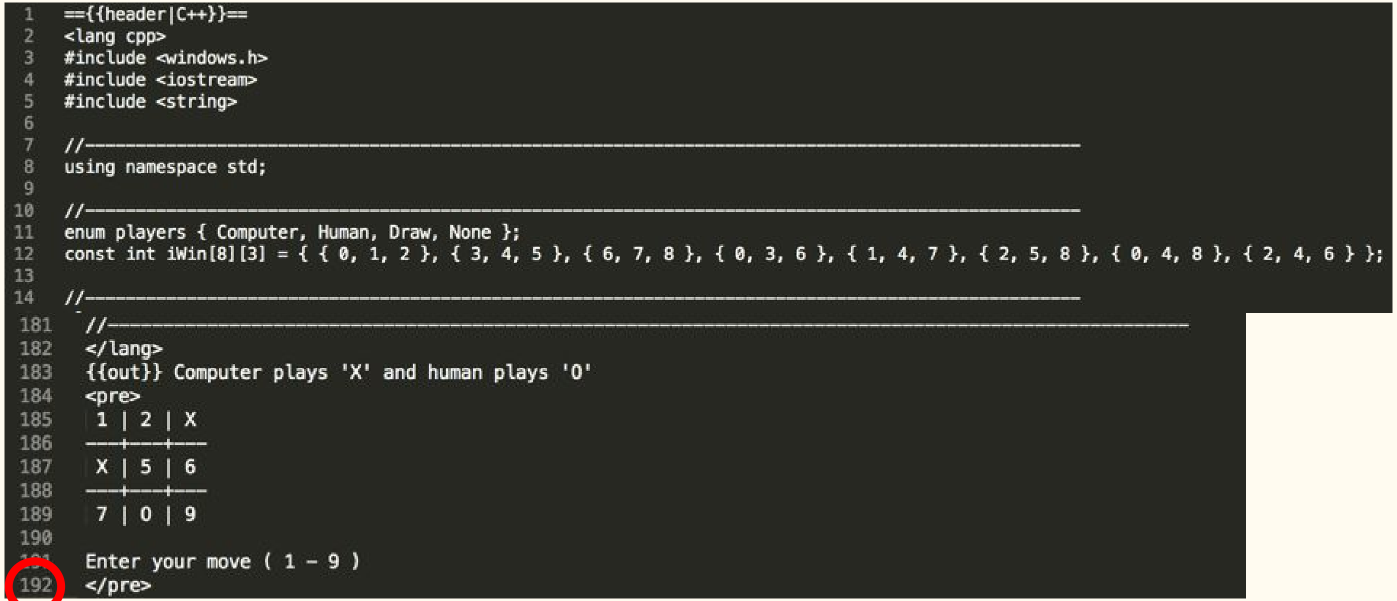


```
530      /**
531       * Main method to start the game
532       */
533      public static void main( String args[] )
534      {
535          JFrame.setDefaultLookAndFeelDecorated(true);
536          Minesweeper msw = new Minesweeper();
537          msw.setJMenuBar( msw.myMenu() );
538          msw.setDefaultCloseOperation( JFrame.DISPOSE_ON_CLOSE );
539          msw.setVisible( true );
540          msw.setResizable( false );
541      }
542
543  }
```

543 Lines?! 130 😎

GridLok vs. Command Line C++

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



```
1   =={{header|C++}}==
2   <lang cpp>
3   #include <windows.h>
4   #include <iostream>
5   #include <string>
6
7   //--------------------------------------------------------------------------------------------------
8   using namespace std;
9
10  //--------------------------------------------------------------------------------------------------
11  enum players { Computer, Human, Draw, None };
12  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 } };
13
14  //--------------------------------------------------------------------------------------------------
181 //--------------------------------------------------------------------------------------------------
182 </lang>
183 {{out}} Computer plays 'X' and human plays 'O'
184 <pre>
185  1 | 2 | X
186 ---+---+---
187  X | 5 | 6
188 ---+---+---
189  7 | O | 9
190
191 Enter your move ( 1 - 9 )
192 </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}
}
```