BGD

--The Board Game Designer

Introduction

Xiao Huang - Project Manager

Ke Wu - Language Guru

Yinghui Huang - System Architect

Xingyu Wang - System Integrator

Dechuan Xu - System Tester

Introduction

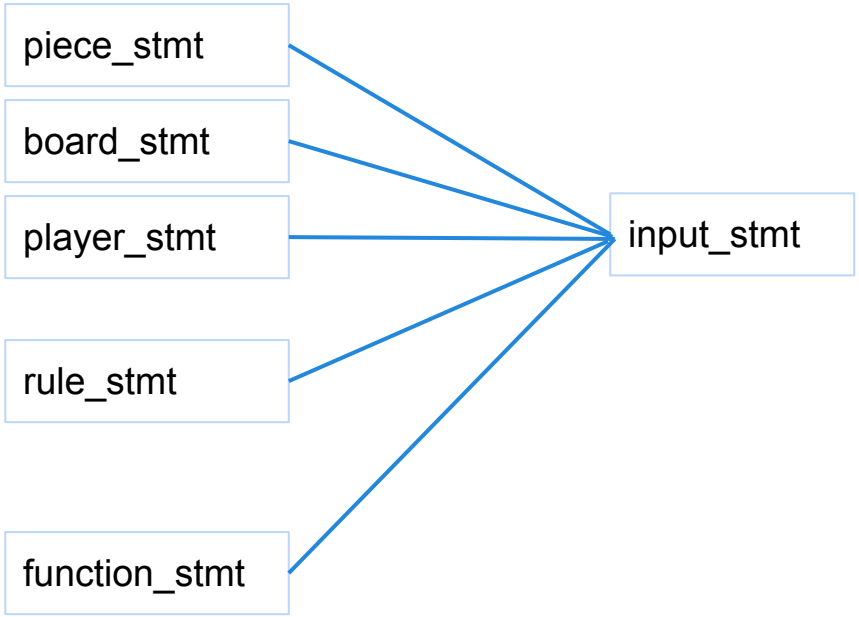
- BGD is a simple language for designing board game applications
- How to use
  - Put your source code in our compiler folder
  - Run $sh compile.sh *your_code.bgd*
  - Run $sh run.sh
  - Enjoy
- Sample Game
  - Tic-Tac-Toe
  - YushenGame

Introduction

- Motivation
  - At very beginning, to be honest...
  - Then, we found it interesting to construct our owner language
  - Want to see our language coming to earth inspires us
- Properties
  - Specify common properties
  - Primitive calculation

Syntactic Constructs: Overview

```
PIECE:
        'stone'

BOARD:
        3 3

PLAYER:
        2

RULE:
        action := add
        action := win

FUNCTION:
       def add piece, position:
                return isEmpty(position)
       def win position:
                if numberInRow(position) >= 3:
                        return YES
                else:
                        return NO
```



Syntactic Constructs: Overview

```
PIECE:
        'stone'
```

State types and numbers of pieces

```
BOARD:
        3 3
```

State size of the game board

```
PLAYER:
        2
```

State number of players

```
RULE:
        action := add
        action := win
```

State what kinds of actions in your game

```
FUNCTION:
       def add piece, position:
                return isEmpty(position)
       def win position:
                if numberInRow(position) >= 3:
                        return YES
                else:
                        return NO
```

State the restriction rules of the actions

Syntactic Constructs: Overview

```
FUNCTION:
        def __init__:
                'STONE' 0 (0,0)
                'STONE' 0 (0,1)
                'STONE' 0 (0,2)
                'STONE' 0 (1,0)
                'STONE' 0 (1,1)
                'STONE' 0 (1,2)
                'STONE' 0 (2,0)
                'STONE' 0 (2,1)
                'STONE' 0 (2,2)
```

If you want to initialize the game board with pieces, you can define an initialize function.

Format:

type player position

Syntactic Constructs:

Primitive data type: int, double, boolean, string

Derived data type: array, pos

- pos: a data type for position on the board, consisting of two int value
- position := (1,1)

Key word: PIECE, PLAYER, BOARD, RULE, FUNCTION, YES, NO, NIL…

Statements:

- if-else statement, while statement, for-loop statement
- function-definition statement

We use indentation to identify a suite

Translator Architecture

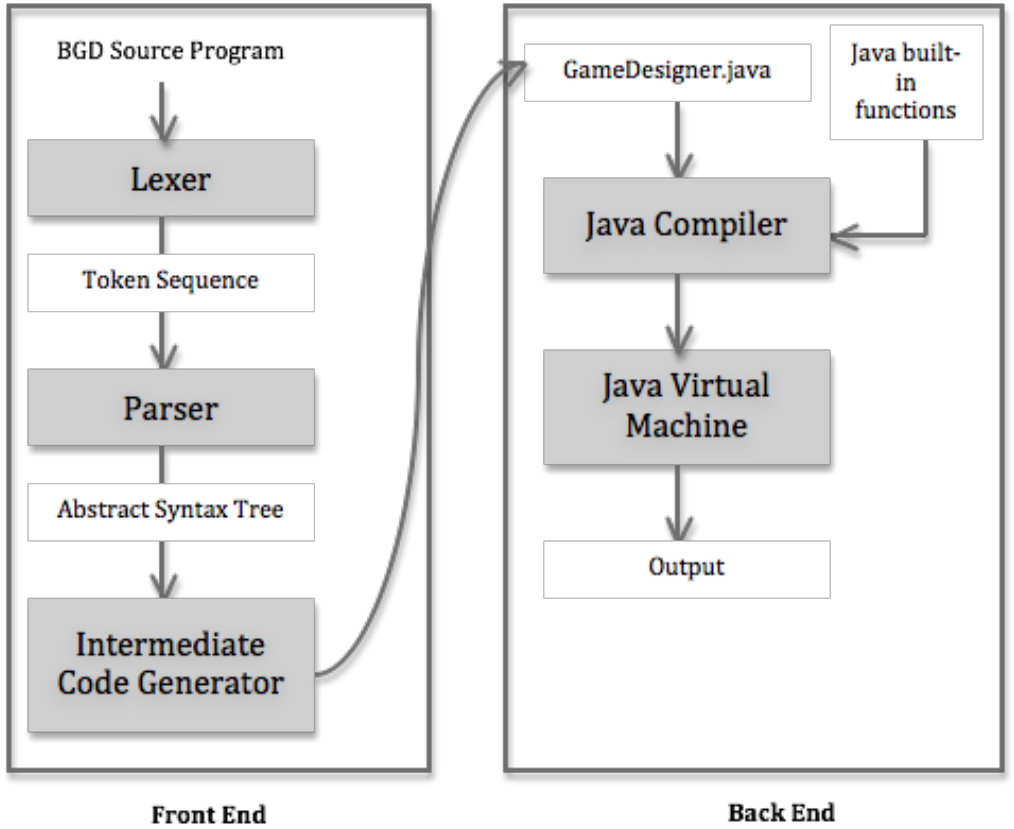
Lexer: lexing.py

Parser: yaccing.py

ICG: traverse.py

Compile shell code: compile.sh

Running shell code: run.sh



Translator Architecture: Front End

tic-tac-toe.bgd

```
PIECE:
        'stone'

BOARD:
        3 3

PLAYER:
        2

RULE:
        action := add
        action := win

FUNCTION:
       def add piece, position:
                return isEmpty(position)
       def win position:
                if numberInRow(position) >= 3:
                        return YES
                else:
                        return NO
```

Token Flow

```
LexToken(PIECE,'PIECE',1,0)
LexToken(:,':',1,5)
LexToken(NEWLINE,'\n',1,6)
LexToken(INDENT,'INDENT',2,7)
LexToken(STRING,"'stone'",2,8)
LexToken(NEWLINE,'\n',2,15)
LexToken(DEDENT,'DEDENT',4,17)
LexToken(BOARD,'BOARD',4,17)
LexToken(:,':',4,22)
LexToken(NEWLINE,'\n',4,23)
LexToken(INDENT,'INDENT',5,24)
LexToken(NUMBER,'3',5,25)
LexToken(NUMBER,'3',5,27)
LexToken(NEWLINE,'\n',5,28)
LexToken(DEDENT,'DEDENT',7,30)
LexToken(PLAYER,'PLAYER',7,30)
LexToken(:,':',7,36)
LexToken(NEWLINE,'\n',7,37)
LexToken(INDENT,'INDENT',8,38)
LexToken(NUMBER,'2',8,39)
```

Translator Architecture: Front End

tic-tac-toe.bgd

```
PIECE:
        'stone'

BOARD:
        3 3

PLAYER:
        2

RULE:
        action := add
        action := win

FUNCTION:
       def add piece, position:
                return isEmpty(position)
       def win position:
                if numberInRow(position) >= 3:
                        return YES
                else:
                        return NO
```

Parser

```
Grammar

Rule 0     S' -> input_stmt
Rule 1     input_stmt -> piece_stmt board_stmt player_stmt rule_stmt function_stmt
Rule 2     piece_stmt -> PIECE : NEWLINE INDENT piece_expr DEDENT
Rule 3     piece_expr -> STRING NUMBER NEWLINE
Rule 4     piece_expr -> STRING NEWLINE

state 0

    (0) S' -> . input_stmt
    (1) input_stmt -> . piece_stmt board_stmt player_stmt rule_stmt function_stmt
    (2) piece_stmt -> . PIECE : NEWLINE INDENT piece_expr DEDENT

    PIECE           shift and go to state 2

    piece_stmt                      shift and go to state 1
    input_stmt                      shift and go to state 3

state 1

    (1) input_stmt -> piece_stmt . board_stmt player_stmt rule_stmt function_stmt
    (7) board_stmt -> . BOARD : NEWLINE INDENT NUMBER NUMBER NEWLINE DEDENT
```

Translator Architecture: Front End

tic-tac-toe.bgd

```
PIECE:
        'stone'

BOARD:
        3 3

PLAYER:
        2

RULE:
        action := add
        action := win

FUNCTION:
       def add piece, position:
                return isEmpty(position)
       def win position:
                if numberInRow(position) >= 3:
                        return YES
                else:
                        return NO
```

AST

```
class Node(object):
    def __init__(self, type, children = [], leaf=None, string = None, token = 'Object'):
        self.type = type #Nonterminal + termianal
        self.children = children #list of children
        self.leaf = leaf #action at this node
        self.string = string #value of attributes
        self.token = token #data type (atom)
```

Translator Architecture: Front End

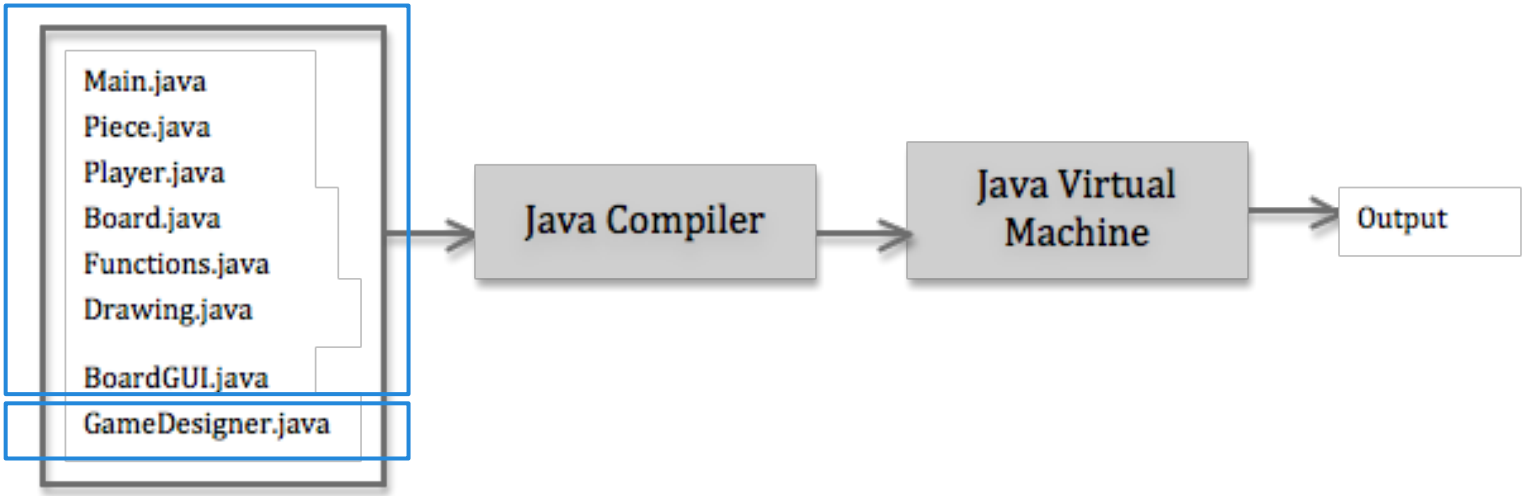
tic-tac-toe.bgd

```
PIECE:
        'stone'

BOARD:
        3 3

PLAYER:
        2

RULE:
        action := add
        action := win

FUNCTION:
       def add piece, position:
                return isEmpty(position)
       def win position:
                if numberInRow(position) >= 3:
                        return YES
                else:
                        return NO
```

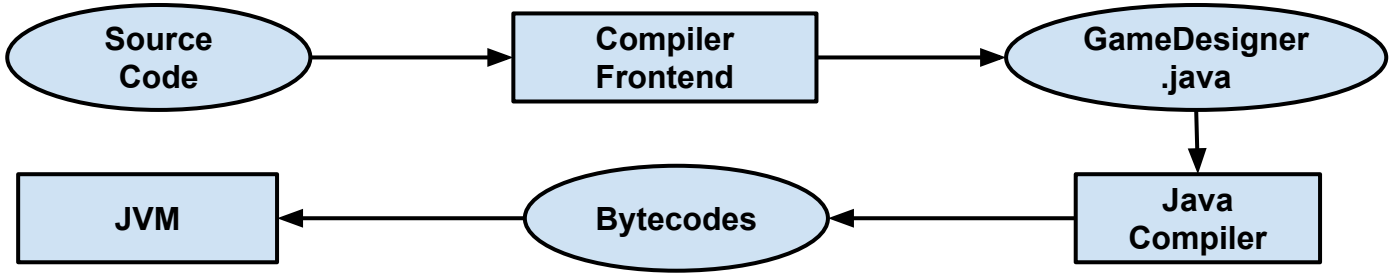
GameDesigner.java

```
import java.lang.*;
import java.util.*;

public class GameDesigner {
        static String[] pieceType = { "stone" };
        static int[] pieceNum = { 0 };
        static int boardRow = 3;
        static int boardCol = 3;
        static int playerNum = 2;

        public static boolean add_res(String piece, Pos position) {
                return Functions.isEmpty(position);
        }

        public static boolean win_res(Pos position) {
                if (Functions.numberInRow(position) >= 3) {
                        return true;
                } else {
                        return false;
                }
        }

        static String[] initPieces = {};
        static int[] initOwner = {};
        static int[][] initPos = {};

        public static boolean move_res(Pos par0, Pos par1) {
                return true;
        }

        public static boolean remove_res(Pos par0) {
                return true;
        }
}
```

Translator Architecture: Back End



Runtime Environment



compile.sh:

```
#!/bin/sh

inputFile=$1

echo "Compile for $inputFile...\n"

#create GameDesigner.java
cd PLTFrontEnd
python traverse.py ../$inputFile > ../warning.log
cp GameDesigner.java ../runnable/GameDesigner.java
echo "GameDesigner.java Done. \n"
#make
cd ../runnable
make clean
make
```

Software Development Environment

Compiler-Generator Tools

PLY (Python Lex-Yacc)

Python 2.7

- Lexer lexing.py
- Parser yaccing.py
- ICG traverse.py

Test Plan

- Mainly focused on frontend
- Based on Python unittest module
- Split into four parts:
  - lexer
  - yaccer
  - traverser
  - general tests

Test Plan: Lexer

1. Lexemes and token types

```
cases = {
    'global' : 'GLOBAL',
    'FUNCTION' : 'FUNCTION',
    '12345' : 'NUMBER',
    '-435384.2523' : 'NUMBER',
    'NO' : 'BOOLEAN',
    "'fhasudfasvsgw87w8ae7r8923b'" : 'STRING',
    "'I want to say:\tThis is awesome!'" : 'STRING',
    "''" : 'STRING',
    'w_32324w':'ID',
    'Zacard' : 'ID',
    '@@asdsafasfasjv' : 'COMMENT',
    }
```

2. Indentation test

3. Sample code test

```
test_indent (__main__.testLexing) ... ok
test_sample_code (__main__.testLexing) ... ok
test_tokens (__main__.testLexing) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.185s

OK
```

Test Plan: Yaccer

- Feed yaccer with test cases and inspect result non-terminals manually.

```
    forin_loop = """
for pos in positions:
    pos.x := 0"""
```

```
test_forin_loop (__main__.testYaccing) ... atom
power
factor
term
operand
comparison
not_test
and_test
or_test
expr
atom
power
trailer
power
atom
power
factor
term
operand
comparison
not_test
and_test
or_test
expr
assign_stmt
simple_stmt
stmt
suite_stmt
suite
for_stmt

ok
```

Test Plan: Traverser

- Feed traverser with test cases to generate ICG (java code), and inspect code manually.

```
test_tic_tac_toe (__main__.testTraverse) ... import java.lang.*;
import java.util.*;
public class GameDesigner{
static String[] pieceType = {"stone"};
static int[] pieceNum = {0};
static int boardRow = 3;
static int boardCol = 3;
static int playerNum = 2;

public static boolean add_res (String piece,Pos position)
{
return Functions.isEmpty(position);
}
public static boolean win_res (Pos position)
{
if(Functions.numberInRow(position)>=3)
{
return true;
}
else
{
return false;
}
}
public static boolean move_res(Pos par0,Pos par1)
{
return true;
}
}

----------------------------------------------------

ok

----------------------------------------------------------------------
Ran 4 tests in 0.768s

OK
```

Test Plan: General Tests

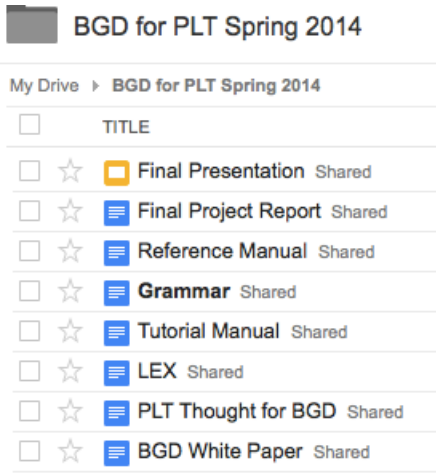
- PLY
  - illegal characters warning for Lex
  - Syntax error and SR conflicts warnings for Yacc
  - Parsing table generated
- Java
  - Eclipse compile-time debugging
  - GUI

PROJECT MANAGEMENT

- **S**tart early
  - As early as Jan.28th
  - Detailed discussion followed

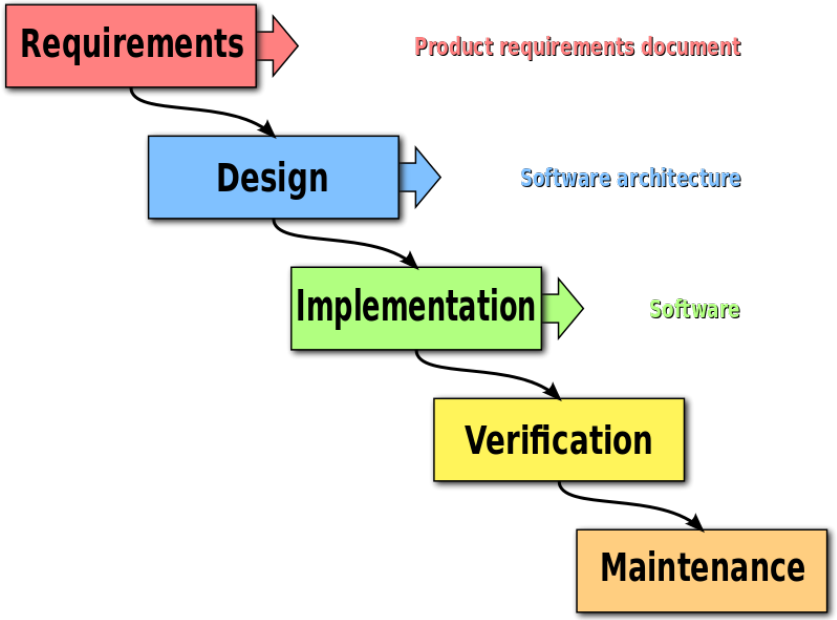
- **C**ommunicate often
  - Meet Every Tuesday

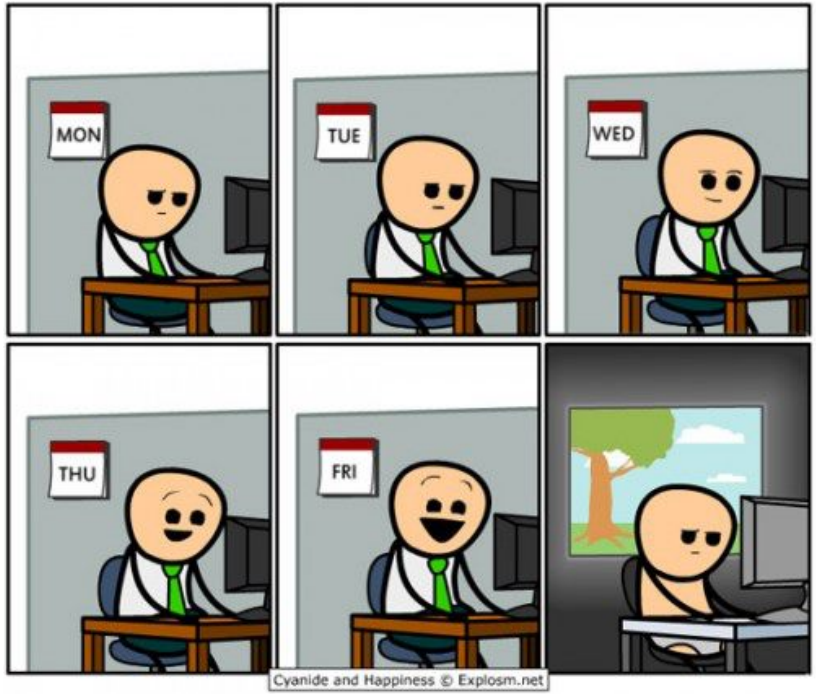
- **K**eep all files up to date



PROJECT MANAGEMENT

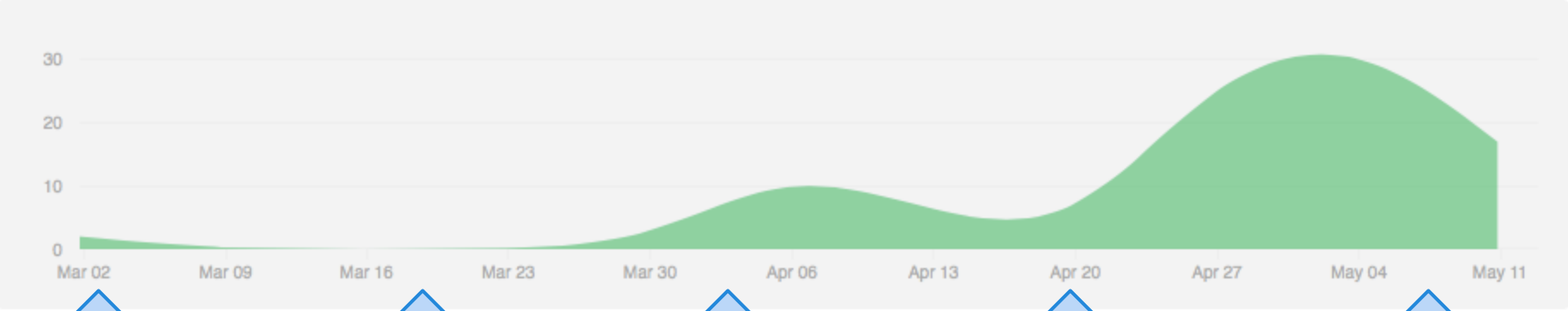
- **W**aterfall Model





PROJECT MANAGEMENT

Github Commits



Project Begin

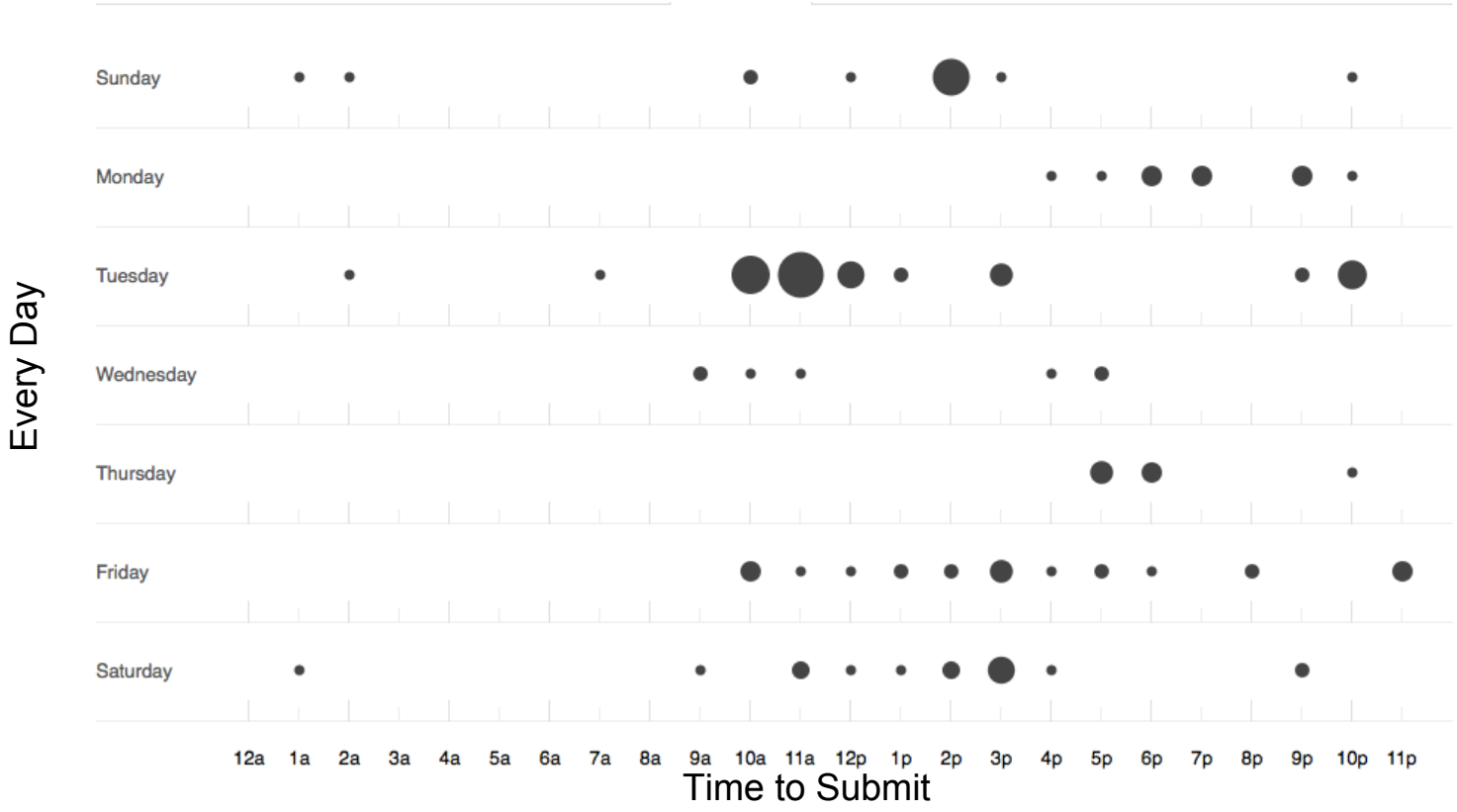
Spring Break

Tutorial & LRM

Before Final

Final Version

PROJECT MANAGEMENT



Conclusion

- Lessons Learned
  - **P**lan ahead
  - **M**eet often
  - **D**ivide work reasonably
  - **D**ebug before commit is important!

Conclusion

- What Worked Well
  - Tuesday after developing…
  - Divide work/everyone responsible for one part

Conclusion

- What If We Could Start Over
  - Setup grammar asap
  - Raise more language examples at first for test

Conclusion

- Why Use Our Language
  - Easy to use
    - 12 lines to implement Gomoku game
  - Fun to design your own game
    - YushenGame
  - Cross Platform
    - Java as target language

Demo

Demo For our Language:

Let’s have some fun!

Demo

Program 1: YushenGame

Program 2: Gomoku

Program 3: Init

We had fun in a team!

THANKS!

Q&A