a language for designing board games

Lauren Pully (Project Manager)
Jesse Bentert (Tools & Language Guru)
John Graham (System Architect)
Daniel Wilkey (System Integrator)
Yipeng Huang (Tester & Validator)

1

Overview

John Graham (System Architect)

2

Outline of a ROLL Program

Jesse Bentert (Tools & Language Guru)

3

ROLL Syntax

Daniel Wilkey (System Integrator)

4

Compiler Architecture & Testing

Yipeng Huang (Tester & Validator)

5

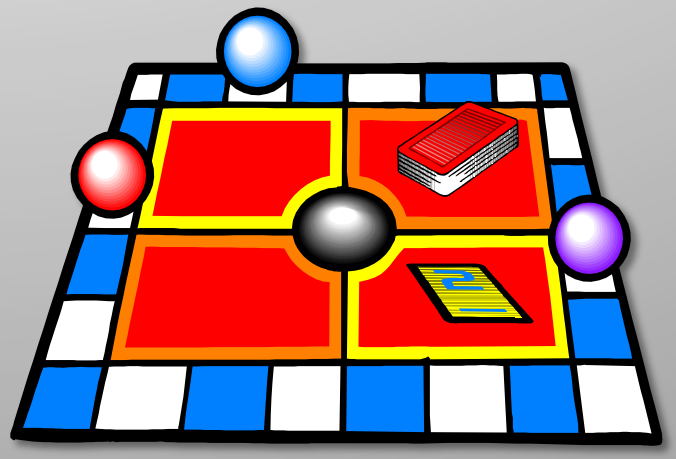
Conclusion & Demo

Lauren Pully (Project Manager)

ROLL

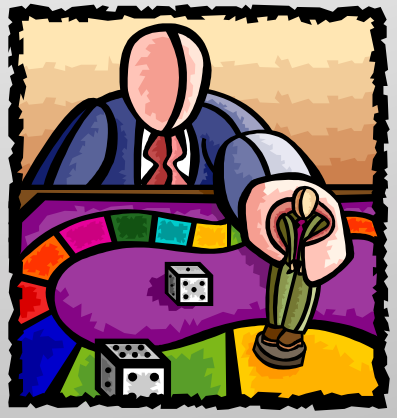
A language to design and play board games

≠



Features of ROLL

Flexible



Dynamic

Extendable

Simple

Eliminates Overhead

Code is easy to understand

1

Overview

John Graham (System Architect)

2

Outline of a ROLL Program

Jesse Bentert (Tools & Language Guru)

3

ROLL Syntax

Daniel Wilkey (System Integrator)

4

Compiler Architecture & Testing

Yipeng Huang (Tester & Validator)

5

Conclusion & Demo

Lauren Pully (Project Manager)

Outline of a ROLL program

```
Game Chutes {
    Players {
        NumPieces
        StartOn
        setupPlayers() {
            NumPlayers
            Player Names
        }
    }
    Board {
        NumTiles
        make Tile()
        landsOn() {
            Jump
        }
        goalCheck() {
            declareWinner
        }
    }
    Dice {
        make Die()
        roll() {
            move
            moveReverse
        }
    }
}
```

```
Game Chutes {
    Players {
        NumPieces
        StartOn
        setupPlayers() {
            NumPlayers
            Player Names
        }
    }
    Board {
        NumTiles
        make Tile()
        landsOn() {
            Jump
        }
        goalCheck() {
            declareWinner
        }
    }
    Dice {
        make Die()
        roll() {
            move
            moveReverse
        }
    }
}
```

The Players Block

```
Players {
  NumPieces
  StartOn
  setupPlayers() {
    NumPlayers
    Player Names
  }
}
```

```
Game Chutes {
    Players {
        NumPieces
        StartOn
        setupPlayers() {
            NumPlayers
            Player Names
        }
    }
    Board {
        NumTiles
        make Tile()
        landsOn() {
            Jump
        }
        goalCheck() {
            declareWinner
        }
    }
    Dice {
        make Die()
        roll() {
            move
            moveReverse
        }
    }
}
```

The Board Block

```
Board {
  NumTiles
  make Tile()
  landsOn() {
    Jump
  }
  goalCheck() {
    declareWinner
  }
}
```

```
Game Chutes {
    Players {
        NumPieces
        StartOn
        setupPlayers() {
            NumPlayers
            Player Names
        }
    }
    Board {
        NumTiles
        make Tile()
        landsOn() {
            Jump
        }
        goalCheck() {
            declareWinner
        }
    }
    Dice {
        make Die()
        roll() {
            move
            moveReverse
        }
    }
}
```

The Dice Block

```
Dice {
  make Die(faces: 6);
  roll() {
    move
    moveReverse
  }
}
```

1

Overview

John Graham (System Architect)

2

Outline of a ROLL Program

Jesse Bentert (Tools & Language Guru)

3

ROLL Syntax

Daniel Wilkey (System Integrator)

4

Compiler Architecture & Testing

Yipeng Huang (Tester & Validator)

5

Conclusion & Demo

Lauren Pully (Project Manager)

GUI = 1;

```
Players {
    GUI = 1;
    MaxPlayers = 6;
    MinPlayers = 2;
    NumPieces = 1;
    StartOn = {0,0,0,0,0,0};
    define setupPlayers() {
        print("How many people are playing this game?");
        promptRange(NumPlayers, MinPlayers, MaxPlayers);
        for(int i : {1 ~ NumPlayers}) {
            print("What is player" | i | "'s name?");
            promptText(PlayerList[i-1].name);
        }
    }
}
```

make Tile(id: 1, next: 2, prev: 0, accessible:{5~9}, landsOn: ladder);

```
Board {
    NumTiles = 10;

    make Tile(id: 1, next: 2, prev: 0, accessible:{5~9}, landsOn: ladder);
    make Tile(id: 3, next: 4, prev: 2, accessible:{8}, landsOn: ladder);
    make Tile(id: 6, next: 7, prev: 5, accessible:{0}, landsOn: chute);
    make Tile(id: 7, next: 8, prev: 6, accessible:{4}, landsOn: chute);

    function ladder = define landsOn(int playerID, int pieceID, int tileID) {
        print("Hooray! " | PlayerList[playerID].name | "'s piece number " | pieceID | "
        went up the ladder to tile " | TileList[tileID].accessible[0]);
        jump(playerID, pieceID, TileList[tileID].accessible[0]);
    }

    …
```

make Tile(id: 1, next: 2, prev: 0, accessible:{5~9}, landsOn: ladder);

```
Board {
    NumTiles = 10;

    make Tile(id: 1, next: 2, prev: 0, accessible:{5~9}, landsOn: ladder);
    make Tile(id: 3, next: 4, prev: 2, accessible:{8}, landsOn: ladder);
    make Tile(id: 6, next: 7, prev: 5, accessible:{0}, landsOn: chute);
    make Tile(id: 7, next: 8, prev: 6, accessible:{4}, landsOn: chute);

    function ladder = define landsOn(int playerID, int pieceID, int tileID) {
        print("Hooray! " | PlayerList[playerID].name | "'s piece number " | pieceID | "
        went up the ladder to tile " | TileList[tileID].accessible[0]);
        jump(playerID, pieceID, TileList[tileID].accessible[0]);
    }

    …
```

```
make Tile(id: 1, next: 2,
prev: 0, accessible:{5~9},
landsOn: ladder);
```

```
Board {
    NumTiles = 10;

    make Tile(id: 1, next: 2, prev: 0, accessible:{5~9}, landsOn: ladder);
    make Tile(id: 3, next: 4, prev: 2, accessible:{8}, landsOn: ladder);
    make Tile(id: 6, next: 7, prev: 5, accessible:{0}, landsOn: chute);
    make Tile(id: 7, next: 8, prev: 6, accessible:{4}, landsOn: chute);

    function ladder = define landsOn(int playerID, int pieceID, int tileID) {
        print("Hooray! " | PlayerList[playerID].name | "'s piece number " | pieceID | "
        went up the ladder to tile " | TileList[tileID].accessible[0]);
        jump(playerID, pieceID, TileList[tileID].accessible[0]);
    }

    …
```

jump(playerID, pieceID, TileList[tileID].accessible[0]);

```
Board {
    NumTiles = 10;

    make Tile(id: 1, next: 2, prev: 0, accessible:{5~9}, landsOn: ladder);
    make Tile(id: 3, next: 4, prev: 2, accessible:{8}, landsOn: ladder);
    make Tile(id: 6, next: 7, prev: 5, accessible:{0}, landsOn: chute);
    make Tile(id: 7, next: 8, prev: 6, accessible:{4}, landsOn: chute);

    function ladder = define landsOn(int playerID, int pieceID, int tileID) {
        print("Hooray! " | PlayerList[playerID].name | "'s piece number " | pieceID | "
        went up the ladder to tile " | TileList[tileID].accessible[0]);
        jump(playerID, pieceID, TileList[tileID].accessible[0]);
    }

    …
```

1

Overview

John Graham (System Architect)

2

Outline of a ROLL Program

Jesse Bentert (Tools & Language Guru)

3

ROLL Syntax

Daniel Wilkey (System Integrator)

4

Compiler Architecture & Testing

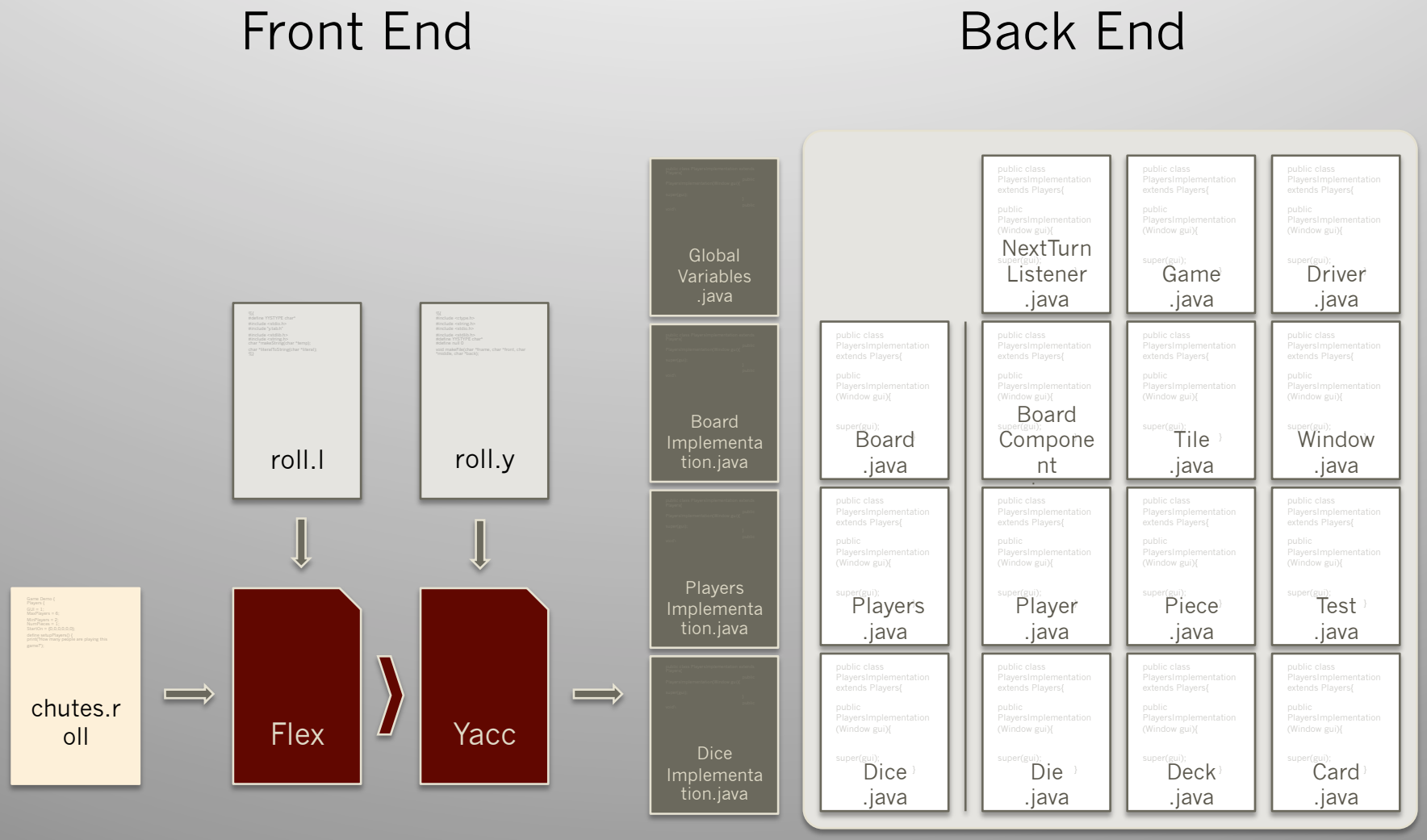
Yipeng Huang (Tester & Validator)

5

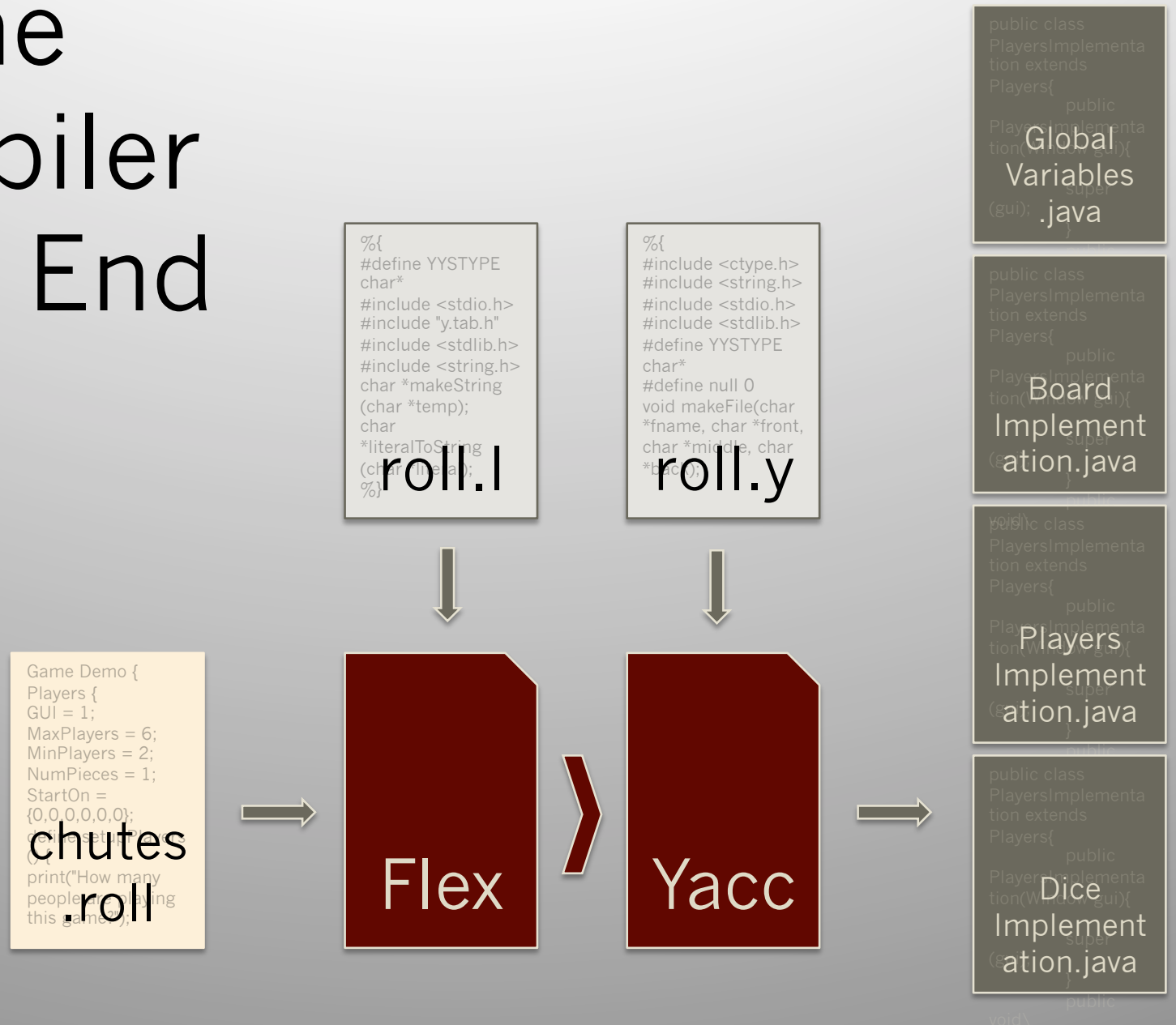
Conclusion & Demo

Lauren Pully (Project Manager)

Compiler Architecture



The Compiler Front End



The Compiler Back End

Overrides

Global Variables .java

Board Implementation.java

Players Implementation.java

Dice Implementation.java

Generalized Game Framework

NextTurn Listener .java

Game .java

Driver .java

Board .java

Board Component .java

Tile .java

Window .java

Players .java

Player .java

Piece .java

Test .java

Dice .java

Die .java

Deck .java

Card .java

The Compiler Back End

Overrides

Global Variables .java

Board Implementation.java

Players Implementation.java

Generalized Game Framework

NextTurn Listener .java

Game .java

Driver .java

Board .java

Board Component .java

Tile .java

Window .java

Players .java

Player .java

Piece .java

Test .java

Dice .java

Die .java

Deck .java

Card .java

1

Overview

John Graham (System Architect)

2

Outline of a ROLL Program

Jesse Bentert (Tools & Language Guru)

3

ROLL Syntax

Daniel Wilkey (System Integrator)

4

Compiler Architecture & Testing

Yipeng Huang (Tester & Validator)

5

Conclusion & Demo

Lauren Pully (Project Manager)

Environment and Tools

Flex + Bison Yacc

Conclusions

- Learned how to make a compiler using Flex and Bison Yacc
- Project management and planning
- Group collaborations


ROLL