Milestone 2, Sprint 1
CSSE 375
Team E
Luke Miller, Sean McPherson, Philip Ross, Zach Haloski
3/30/16

Table of Contents

Design Documentation

Extracted logic associated with move() in Game out into a number of classes using the Command pattern. This simplified the overall logic within move() and its helper methods, and simply passes the position and directional data to the MoveCommand object which handles everything from there. Each MoveCommand stores the state of the board before it calculates a move as well as the state after the move (though this probably can be removed later). Additionally, in Game we store these MoveCommands in an ArrayList (should be a queue) so that we can easily keep track of each move made and also to make undo'ing moves easier to accomplish.

To remove some lines of code from the huge Gui class, we moved the large listeners into their own classes, and refactored the gui class to use these. This also allows us to reuse these listeners if we make other panels while adding features. In addition, it insures that we separate the listener logic from the other logic and allows our code to be much more modular and maintainable.

We have several features that were large, and we laid the groundwork for these. Adding network connectivity to a game designed around only local play is very difficult. We have the basic connection code in place in a feature branch that is being kept in sync with main so that we can merge that feature into our master branch once it is done, without breaking all of the rest of our refactorings. This feature will be completed in milestone 3, and mostly just needs to gui code to be written to support connecting to another client.

We also have the basic framework to add an ai opponent for the use to play singleplayer against. This will be using our new piece interface, and will be able to generate move objects, so it won't have to interact with our gui at all. This feature will also be finished in milestone 3

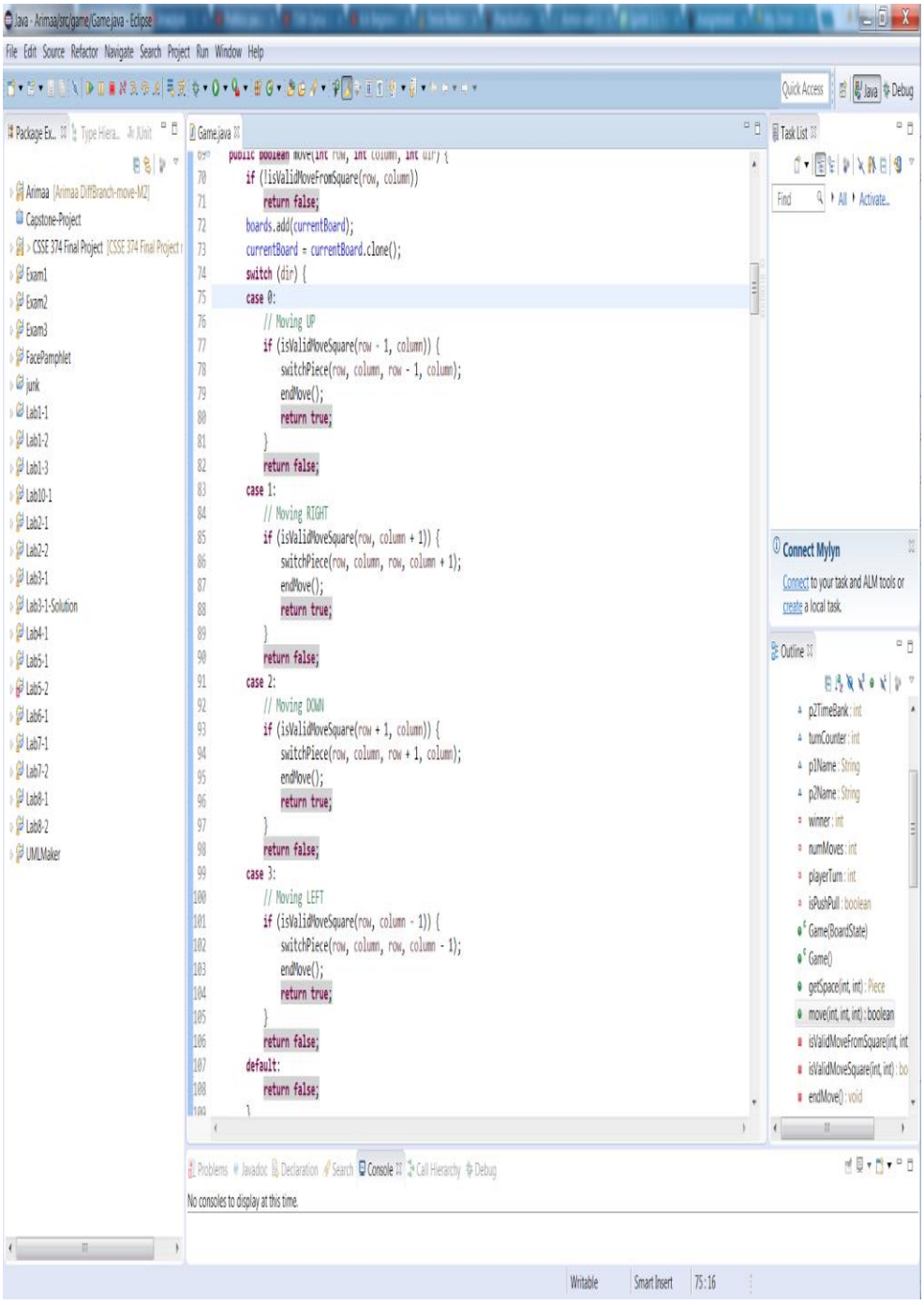
Code smells:

Long Method (Sean, Luke)
Extracted code out of the move() method and its helpers into several classes arranged in a Command pattern style, thus shortening the overall length and complexity of the method.
Switch Statements (Sean)
Very much in line with the previous code smell, we were able to simplify the method by pulling logic out of it and replacing them with method calls in such a way that the methods or objects knew what to do internally rather than needing to be explicitly told using arguments. Additionally, simplified some logic in the undo() method by removing integer checks and replacing them with a simple algebraic expression based on the original purpose and structure of the method.
Comments (Sean)
All throughout our codebase, and particularly in the Game and GUI classes, we have numerous comments strewn about, indicating what fields are, explaining methods, and giving context to otherwise ambiguous code segments. These have since been removed and all ambiguities have been remedied, mostly with changes in the names of methods or fields.
Duplicated Code (Luke)
A large portion of our GUI class contained multiple places where redundant or extremely similar code was being run. As a result, we refactored them out into sensible and smaller methods to be called, rather than have the same ~15-20 lines of code occurring multiple times to do similar tasks.
Long Class (Luke, Phil)
Similarly with the duplicated code bad smell, we split up the responsibilities of the GUI into a number of different Listener classes, all according to their function. This drastically reduced the overall length of our GUI class.
Shotgun Surgery
In various places throughout our code, we found that refactoring or pulling out certain methods was extremely tricky due to their strong reliance on various method calls in places like the Game class.
Data Clumping (Sean)
Row and column where being passed around to many different functions for moving pieces around. We eliminated this by encapsulating the move logic and parameters into a command object. This command object is created, then passed to game, which then executes the move.

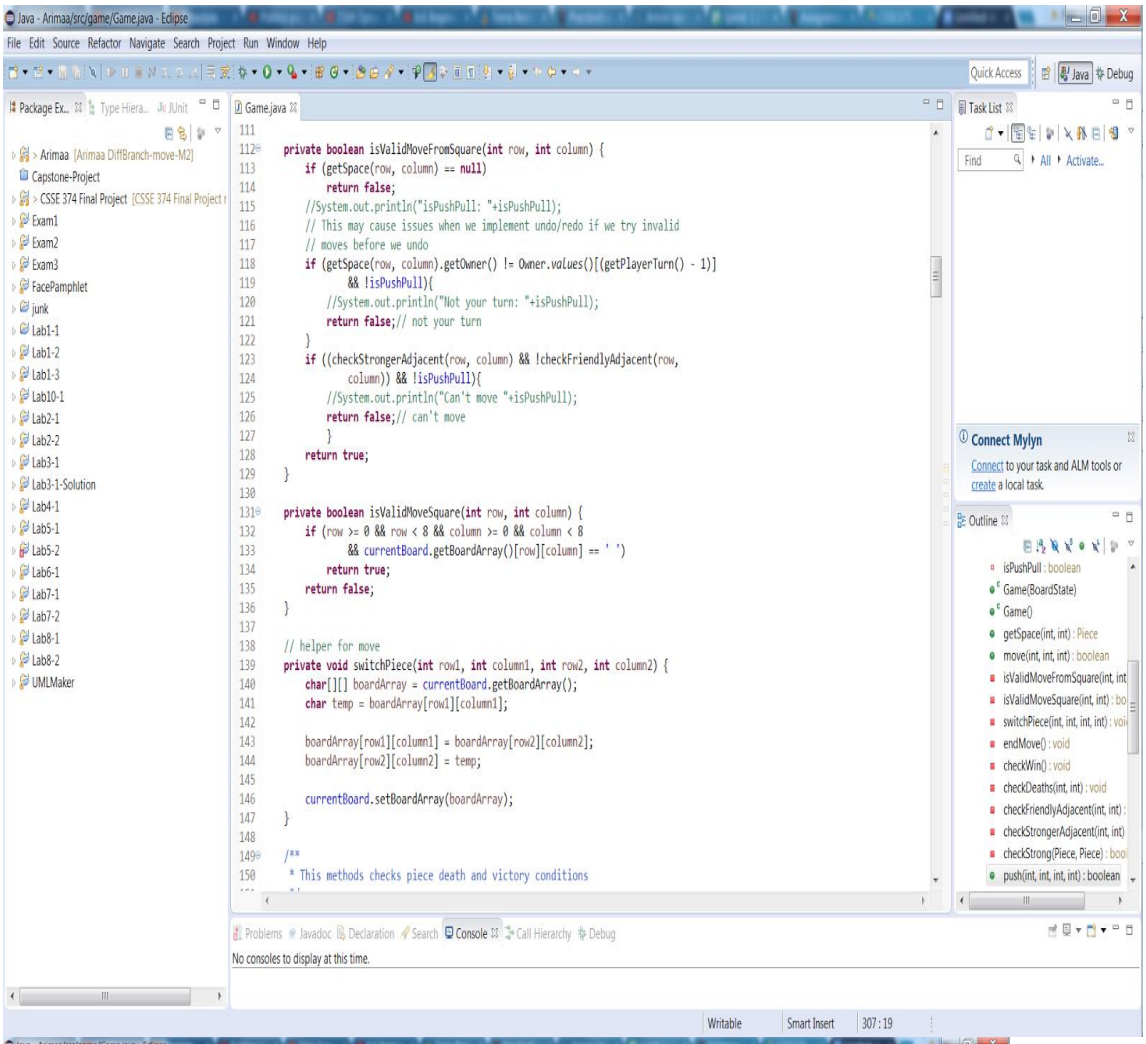
Primitive Obsession
In several cases, we replaced many of our various fields and variables with concrete objects which contained the data that these fields originally contained, but instead stored within separate classes. One specific example of this was our addition of the Piece classes, which we are using now in place of simply parsing out characters from a 2D array. We also often used integers in place of enumerations, like with our move logic in Game.
Inappropriate Intimacy (GUI and listeners getting GUI object)

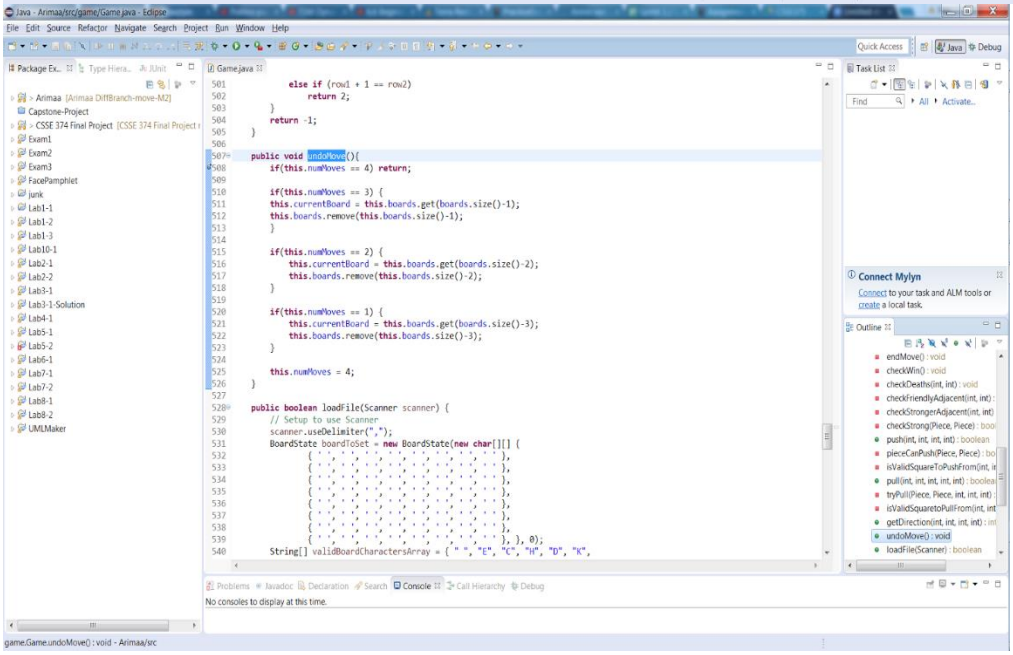
Before and After Code Snippets:

Move and Related Methods: Before

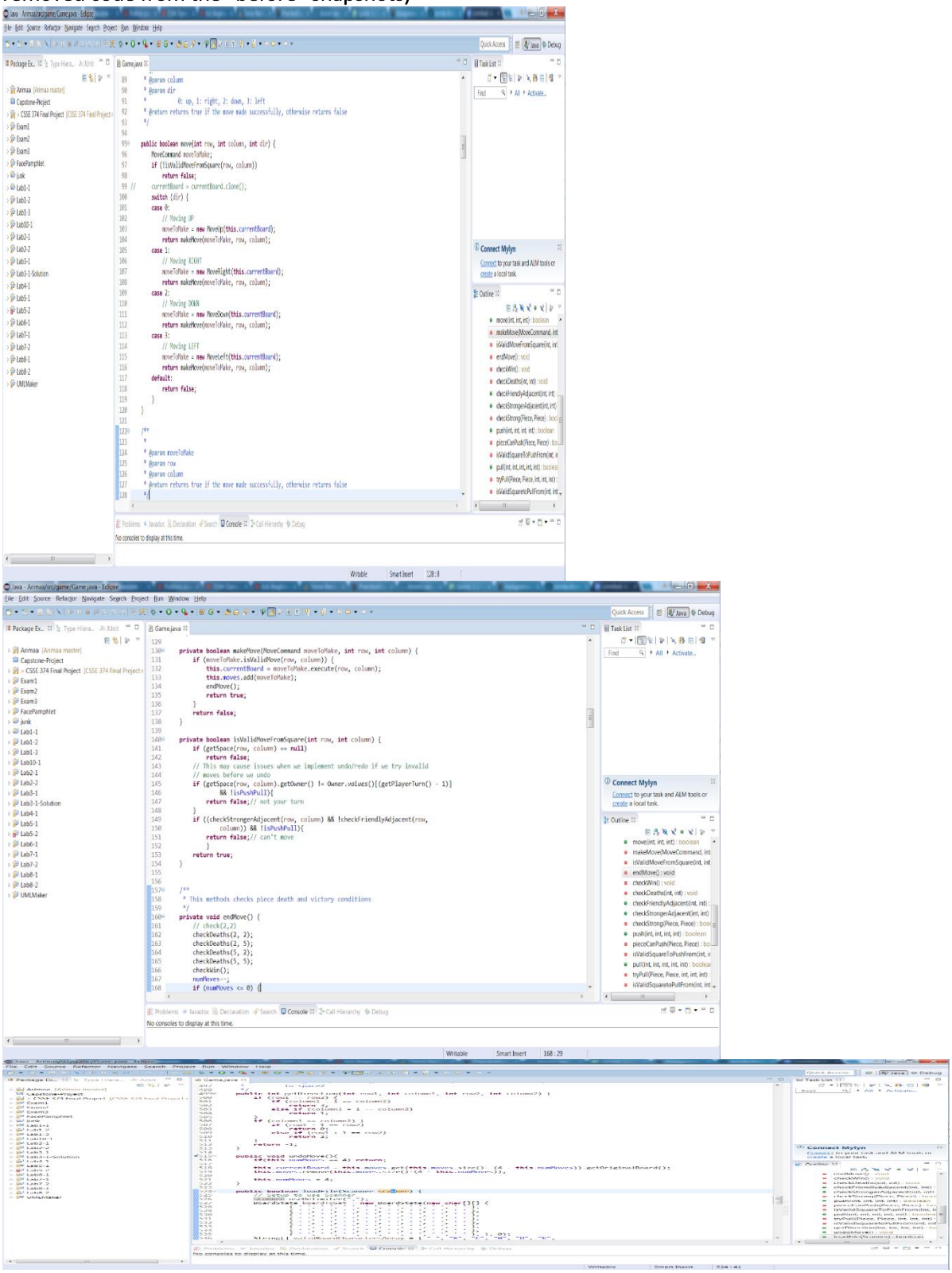



```
70      if (!isValidMoveFromSquare(row, column))
71          return false;
72      boards.add(currentBoard);
73      currentBoard = currentBoard.clone();
74      switch (dir) {
75      case 0:
76          // Moving UP
77          if (isValidMoveSquare(row - 1, column)) {
78              switchPiece(row, column, row - 1, column);
79              endMove();
80              return true;
81          }
82          return false;
83      case 1:
84          // Moving RIGHT
85          if (isValidMoveSquare(row, column + 1)) {
86              switchPiece(row, column, row, column + 1);
87              endMove();
88              return true;
89          }
90          return false;
91      case 2:
92          // Moving DOWN
93          if (isValidMoveSquare(row + 1, column)) {
94              switchPiece(row, column, row + 1, column);
95              endMove();
96              return true;
97          }
98          return false;
99      case 3:
100         // Moving LEFT
101         if (isValidMoveSquare(row, column - 1)) {
102             switchPiece(row, column, row, column - 1);
103             endMove();
104             return true;
105         }
106         return false;
107     default:
108         return false;
```


Java - Arimaa/src/game/Game.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Package Ex...
Type Hiera...
JUnit
Game.java
Task List
Find
All
Activate...
> Arimaa [Arimaa DiffBranch-move-M2]
Capstone-Project
> CSSE 374 Final Project [CSSE 374 Final Project
Exam1
Exam2
Exam3
FacePamphlet
junk
Lab1-1
Lab1-2
Lab1-3
Lab10-1
Lab2-1
Lab2-2
Lab3-1
Lab3-1-Solution
Lab4-1
Lab5-1
Lab5-2
Lab6-1
Lab7-1
Lab7-2
Lab8-1
Lab8-2
UMLMaker
111
112 private boolean isValidMoveFromSquare(int row, int column) {
113 if (getSpace(row, column) == null)
114 return false;
115 //System.out.println("isPushPull: "+isPushPull);
116 // This may cause issues when we implement undo/redo if we try invalid
117 // moves before we undo
118 if (getSpace(row, column).getOwner() != Owner.values()[(getPlayerTurn() - 1)]
119 && !isPushPull){
120 //System.out.println("Not your turn: "+isPushPull);
121 return false;// not your turn
122 }
123 if ((checkStrongerAdjacent(row, column) && !checkFriendlyAdjacent(row,
124 column)) && !isPushPull){
125 //System.out.println("Can't move "+isPushPull);
126 return false;// can't move
127 }
128 return true;
129 }
130
131 private boolean isValidMoveSquare(int row, int column) {
132 if (row >= 0 && row < 8 && column >= 0 && column < 8
133 && currentBoard.getBoardArray()[row][column] == ' ')
134 return true;
135 return false;
136 }
137
138 // helper for move
139 private void switchPiece(int row1, int column1, int row2, int column2) {
140 char[][] boardArray = currentBoard.getBoardArray();
141 char temp = boardArray[row1][column1];
142
143 boardArray[row1][column1] = boardArray[row2][column2];
144 boardArray[row2][column2] = temp;
145
146 currentBoard.setBoardArray(boardArray);
147 }
148
149 /**
150 * This methods checks piece death and victory conditions
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
isPushPull : boolean
Game(BoardState)
Game()
getSpace(int, int) : Piece
move(int, int, int) : boolean
isValidMoveFromSquare(int
isValidMoveSquare(int, int) : bo
switchPiece(int, int, int, int) : voi
endMove() : void
checkWin() : void
checkDeaths(int, int) : void
checkFriendlyAdjacent(int, int) :
checkStrongerAdjacent(int, int)
checkStrong(Piece, Piece) : bool
push(int, int, int, int) : boolean
Problems
Javadoc
Declaration
Search
Console
Call Hierarchy
Debug
No consoles to display at this time.
Writable
Smart Insert
307 : 19


Java - Arimaa/src/game/Game.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Package Ex...
Type Hiera...
JUnit
Game.java
Task List
Find
All
Activate...
> Arimaa [Arimaa DiffBranch-move-M2]
Capstone-Project
> CSSE 374 Final Project [CSSE 374 Final Project
Exam1
Exam2
Exam3
FacePamphlet
junk
Lab1-1
Lab1-2
Lab1-3
Lab10-1
Lab2-1
Lab2-2
Lab3-1
Lab3-1-Solution
Lab4-1
Lab5-1
Lab5-2
Lab6-1
Lab7-1
Lab7-2
Lab8-1
Lab8-2
UMLMaker
501 else if (row1 + 1 == row2)
502 return 2;
503 }
504 return -1;
505 }
506
507 public void undoMove(){
508 if(this.numMoves == 4) return;
509
510 if(this.numMoves == 3) {
511 this.currentBoard = this.boards.get(boards.size()-1);
512 this.boards.remove(this.boards.size()-1);
513 }
514
515 if(this.numMoves == 2) {
516 this.currentBoard = this.boards.get(boards.size()-2);
517 this.boards.remove(this.boards.size()-2);
518 }
519
520 if(this.numMoves == 1) {
521 this.currentBoard = this.boards.get(boards.size()-3);
522 this.boards.remove(this.boards.size()-3);
523 }
524
525 this.numMoves = 4;
526 }
527
528 public boolean loadFile(Scanner scanner) {
529 // Setup to use Scanner
530 scanner.useDelimiter(",");
531 BoardState boardToSet = new BoardState(new char[][] {
532 { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },
533 { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },
534 { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },
535 { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },
536 { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },
537 { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },
538 { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' },
539 { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' }, }, 0);
540 String[] validBoardCharactersArray = { " ", "E", "C", "H", "D", "K",
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
endMove() : void
checkWin() : void
checkDeaths(int, int) : void
checkFriendlyAdjacent(int, int) :
checkStrongerAdjacent(int, int)
checkStrong(Piece, Piece) : bool
push(int, int, int, int) : boolean
pieceCanPush(Piece, Piece) : bo
isValidSquareToPushFrom(int, i
pull(int, int, int, int, int) : boolea
tryPull(Piece, Piece, int, int, int)
isValidSquaretoPullFrom(int, int
getDirection(int, int, int, int) : int
undoMove() : void
loadFile(Scanner) : boolean
Problems
Javadoc
Declaration
Search
Console
Call Hierarchy
Debug
No consoles to display at this time.
game.Game.undoMove() : void - Arimaa/src

Move and Related Methods: After (also added several MoveCommand classes which contain much of the removed code from the “before” snapshots)



Piece refactoring: before

```java
package testing;

import static org.junit.Assert.*;

import java.awt.Image;
import java.awt.image.BufferedImage;
import java.util.ArrayList;

import game.Piece;
import game.Piece.Owner;
import game.Piece.PieceType;

import org.junit.Test;

public class TestPiece {

	@Test
	public void testThatPieceInitializes() {
		Piece p = new Piece(PieceType.Camel, null, Owner.Player1);
		assertNotNull(p);
	}

	@Test
	public void testThatPieceInitializesWithValues() {
		Image img = new BufferedImage(1, 1, 1);
		Piece p = new Piece(PieceType.Camel, img, Owner.Player1);
		assertNotNull(p);
		assertEquals(PieceType.Camel, p.getType());
		assertEquals(img, p.getImage());
	}

	@Test
	public void testThatTypeCanBeGotten() {
		Image img = new BufferedImage(1, 1, 1);
		Piece p = new Piece(PieceType.Camel, img, Owner.Player1);
		assertEquals(PieceType.Camel, p.getType());
	}

	@Test
	public void testThatImageCanBeGotten() {
		Image img = new BufferedImage(1, 1, 1);
		Piece p = new Piece(PieceType.Camel, img, Owner.Player1);
		assertEquals(img, p.getImage());
	}

	@Test
	public void testThatTypeCanBeSet() {
		Image img = new BufferedImage(1, 1, 1);
		Piece p = new Piece(PieceType.Camel, img, Owner.Player1);
		p.setType(PieceType.Elephant);
		assertEquals(PieceType.Elephant, p.getType());
	}
```

```
53
54          @Test
55          public void testThatImageCanBeSet() {
56                  Image img = new BufferedImage(1, 1, 1);
57                  Image img2 = new BufferedImage(50, 50, 2);
58                  Piece p = new Piece(PieceType.Camel, img, Owner.Player1);
59                  p.setImg(img2);
60                  assertEquals(img2, p.getImage());
61          }
62
63          @Test
64          public void testThatOwnerCanBeSetAndGotten() {
65                  Image img = new BufferedImage(1, 1, 1);
66                  Piece p = new Piece(PieceType.Camel, img, Owner.Player1);
67                  p.setOwner(Owner.Player2);
68                  assertEquals(Owner.Player2, p.getOwner());
69          }
70
71          @Test
72          public void testComparatorChecksOwners() {
73                  assertNotEquals(new Piece(PieceType.Rabbit, null, Piece.Owner.Player1),
74                                  new Piece(PieceType.Rabbit, null, Piece.Owner.Player2));
75                  assertEquals(new Piece(PieceType.Rabbit, null, Piece.Owner.Player2),
76                                  new Piece(PieceType.Rabbit, null, Piece.Owner.Player2));
77          }
78
79          @Test
80          public void testIsElephantStrongerThanCamel() {
81                  Piece p1 = new Piece('E');
82                  Piece p2 = new Piece('c');
83                  assertTrue(p1.isStrongerThan(p2));
84          }
85
86          @Test
87          public void testIsElephantStrongerThanElephant() {
88                  Piece p1 = new Piece('E');
89                  Piece p2 = new Piece('e');
90                  assertFalse(p1.isStrongerThan(p2));
91          }
92
93          @Test
94          public void testIsCamelStrongerThanCamel() {
95                  Piece p1 = new Piece('C');
96                  Piece p2 = new Piece('c');
97                  assertFalse(p1.isStrongerThan(p2));
98          }
99
100         @Test
101         public void testIsCamelStrongerThanHorse() {
102                 Piece p1 = new Piece('C');
103                 Piece p2 = new Piece('h');
104                 assertTrue(p1.isStrongerThan(p2));
105         }
```

```
105         }
106
107         @Test
108         public void testIsHorseStrongerThanDog() {
109                 Piece p1 = new Piece('H');
110                 Piece p2 = new Piece('d');
111                 assertTrue(p1.isStrongerThan(p2));
112         }
113
114         @Test
115         public void testIsDogStrongerThanDog() {
116                 Piece p1 = new Piece('D');
117                 Piece p2 = new Piece('d');
118                 assertFalse(p1.isStrongerThan(p2));
119         }
120
121         @Test
122         public void testThatConstructorHandlesDefaultCase(){
123                 Piece p = new Piece('g');
124                 assertEquals(null, p.getOwner());
125                 assertEquals(null, p.getType());
126         }
127
128         @Test
129         public void testThatConstructorHandlesDefaultCase2(){
130                 Piece p = new Piece('G');
131                 assertEquals(null, p.getOwner());
132                 assertEquals(null, p.getType());
133         }
134
135         @Test
136         public void testEqualsReturnsFalseForOtherObject(){
137                 Piece p = new Piece('E');
138                 ArrayList<Integer> notAPiece = new ArrayList<Integer>();
139                 assertFalse(p.equals(notAPiece));
140         }
141
142         @Test
143         public void testSetRank(){
144                 Piece p = new Piece('C');
145                 p.setRank(1);
146                 assertEquals(1, p.getRank());
147         }
148 }
```

```java
package game;

import java.awt.Image;

public class Piece {
	private PieceType type;
	private Image image;
	private Owner owner;
	private int rank;

	public enum Owner {
		Player1, Player2
	}

	public enum PieceType {
		Elephant, Camel, Horse, Dog, Cat, Rabbit
	}

	public Piece(char c) {
		if (Character.isUpperCase(c))
			createP1Piece(c);
		else
			createP2Piece(c);
	}

	private void createP2Piece(char c) {
		switch (c) {
		case 'e':
			this.type = PieceType.Elephant;
			this.owner = Owner.Player2;
			this.rank = 5;
			break;
		case 'c':
			this.type = PieceType.Camel;
			this.owner = Owner.Player2;
			this.rank = 4;
			break;
		case 'h':
			this.type = PieceType.Horse;
			this.owner = Owner.Player2;
			this.rank = 3;
			break;
		case 'd':
			this.type = PieceType.Dog;
			this.owner = Owner.Player2;
			this.rank = 2;
			break;
		case 'k':
			this.type = PieceType.Cat;
			this.owner = Owner.Player2;
			this.rank = 1;
			break;
```

```
53                  case 'r':
54                          this.type = PieceType.Rabbit;
55                          this.owner = Owner.Player2;
56                          this.rank = 0;
57                          break;
58                  default:
59                          System.err.println("Invalid char supplied");
60                  }
61          }
62
63          private void createP1Piece(char c) {
64                  switch (c) {
65                  case 'E':
66                          this.type = PieceType.Elephant;
67                          this.owner = Owner.Player1;
68                          this.rank = 5;
69                          break;
70                  case 'C':
71                          this.type = PieceType.Camel;
72                          this.owner = Owner.Player1;
73                          this.rank = 4;
74                          break;
75                  case 'H':
76                          this.type = PieceType.Horse;
77                          this.owner = Owner.Player1;
78                          this.rank = 3;
79                          break;
80                  case 'D':
81                          this.type = PieceType.Dog;
82                          this.owner = Owner.Player1;
83                          this.rank = 2;
84                          break;
85                  case 'K':
86                          this.type = PieceType.Cat;
87                          this.owner = Owner.Player1;
88                          this.rank = 1;
89                          break;
90                  case 'R':
91                          this.type = PieceType.Rabbit;
92                          this.owner = Owner.Player1;
93                          this.rank = 0;
94                          break;
95
96                  default:
97                          System.err.println("Invalid char supplied");
98                  }
99          }
100
101         public Piece(PieceType t, Image i, Owner o) {
102                 this.type = t;
103                 this.image = i;
104                 this.owner = o;
105         }
```

```
106
107         public PieceType getType() {
108                 return this.type;
109         }
110
111         public void setType(PieceType type) {
112                 this.type = type;
113         }
114
115         public Image getImage() {
116                 return this.image;
117         }
118
119         public void setImg(Image img) {
120                 this.image = img;
121         }
122
123         public Owner getOwner() {
124                 return this.owner;
125         }
126
127         public void setOwner(Owner owner) {
128                 this.owner = owner;
129         }
130
131         public int getRank() {
132                 return this.rank;
133         }
134
135         public void setRank(int rank) {
136                 this.rank = rank;
137         }
138
139         public boolean equals(Object p2) {
140                 if ((p2 instanceof Piece)) {
141                         if (((Piece) p2).getType() == this.getType()
142                                         && (((Piece) p2).getOwner() == this.getOwner())) {
143                                 return true;
144                         }
145                 }
146                 return false;
147
148         }
149
150         public boolean isStrongerThan(Piece p2) {
151                 return (this.getRank() > p2.getRank());
152         }
153 }
```

Piece Refactor After:

Piece.java is not modified to maintain backwards compadibility, we will substitute the new piece class in other refactors

```java
package testing;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertTrue;

import java.awt.Image;
import java.awt.image.BufferedImage;
import java.util.ArrayList;

import javax.swing.ImageIcon;

import org.junit.Test;

import piece.AbstractPiece;
import piece.Camel;
import piece.Dog;
import piece.Elephant;
import piece.Horse;
import piece.Owner;
import piece.Piece;
import piece.Rabbit;

public class TestPiece {

	@Test
	public void testThatPieceInitializes() {
		AbstractPiece p = new Camel(Owner.Player1);
		assertNotNull(p);
	}

	@Test
	public void testThatTypeCanBeGotten() {
		AbstractPiece p = new Camel(Owner.Player1);
		assertTrue(p instanceof Camel);
	}

	@Test
	public void testThatImageCanBeGotten() {
		Image img = new ImageIcon("resources/White camel.png").getImage();
		AbstractPiece p = new Camel(Owner.Player1);
		assertEquals(img, p.getImage());
	}

	@Test
	public void testThatImageCanBeSet() {
		Image img = new BufferedImage(1, 1, 1);
		AbstractPiece p = new Camel(Owner.Player1);
		p.setImage(img);
		assertEquals(img, p.getImage());
	}
```

```
55          @Test
56          public void testThatOwnerCanBeGotten() {
57                  AbstractPiece p = new Camel(Owner.Player1);
58                  assertEquals(Owner.Player1, p.getOwner());
59          }
60
61          @Test
62          public void testGetRank() {
63                  AbstractPiece p = new Elephant(Owner.Player1);
64                  assertEquals(5, p.getRank());
65          }
66
67          @Test
68          public void testComparatorChecksOwners() {
69                  assertNotEquals(new Rabbit(Owner.Player1), new Rabbit(Owner.Player2));
70                  assertEquals(new Rabbit(Owner.Player2), new Rabbit(Owner.Player2));
71          }
72
73          @Test
74          public void testIsElephantStrongerThanCamel() {
75                  AbstractPiece p1 = new Elephant(Owner.Player1);
76                  AbstractPiece p2 = new Camel(Owner.Player2);
77                  assertTrue(p1.isStrongerThan(p2));
78          }
79
80          @Test
81          public void testIsElephantStrongerThanElephant() {
82                  AbstractPiece p1 = new Elephant(Owner.Player1);
83                  AbstractPiece p2 = new Elephant(Owner.Player2);
84                  assertFalse(p1.isStrongerThan(p2));
85          }
86
87          @Test
88          public void testIsCamelStrongerThanCamel() {
89                  AbstractPiece p1 = new Camel(Owner.Player1);
90                  AbstractPiece p2 = new Camel(Owner.Player2);
91                  assertFalse(p1.isStrongerThan(p2));
92          }
93
94          @Test
95          public void testIsCamelStrongerThanHorse() {
96                  AbstractPiece p1 = new Camel(Owner.Player1);
97                  AbstractPiece p2 = new Horse(Owner.Player2);
98                  assertTrue(p1.isStrongerThan(p2));
99          }
100
101         @Test
102         public void testIsHorseStrongerThanDog() {
103                 AbstractPiece p1 = new Horse(Owner.Player1);
104                 AbstractPiece p2 = new Dog(Owner.Player2);
105                 assertTrue(p1.isStrongerThan(p2));
106         }
107
```

```
107
108         @Test
109         public void testIsDogStrongerThanDog() {
110                 AbstractPiece p1 = new Dog(Owner.Player1);
111                 AbstractPiece p2 = new Dog(Owner.Player2);
112                 assertFalse(p1.isStrongerThan(p2));
113         }
114
115         @Test
116         public void testEqualsReturnsFalseForOtherObject() {
117                 AbstractPiece p = new Elephant(Owner.Player1);
118                 ArrayList<Integer> notAPiece = new ArrayList<Integer>();
119                 assertFalse(p.equals(notAPiece));
120         }
121 }
```

39 lines (28 sloc) | 616 Bytes

Raw | Blame | History

```java
package piece;

import java.awt.Image;

public abstract class AbstractPiece {
	private Image image;
	private Owner owner;
	private int rank;

	public AbstractPiece(Image image, Owner owner, int rank) {
		this.image = image;
		this.owner = owner;
		this.rank = rank;
	}

	public Image getImage() {
		return image;
	}

	public void setImage(Image image) {
		this.image = image;
	}

	public Owner getOwner() {
		return owner;
	}

	public int getRank() {
		return rank;
	}

	abstract public boolean equals(Object obj);

	public boolean isStrongerThan(AbstractPiece p2) {
		return (this.getRank() > p2.getRank());
	}

}
```

21 lines (17 sloc) | 471 Bytes

```java
package piece;

import javax.swing.ImageIcon;

public class Camel extends AbstractPiece {
	public Camel(Owner owner) {
		super(null, owner, 4);
		String color = owner.equals(Owner.Player1) ? "White" : "Black";
		this.setImage(new ImageIcon("resources/" + color + " camel.png").getImage());

	}

	@Override
	public boolean equals(Object obj) {
		if (obj instanceof Camel) {
			Camel e = (Camel) obj;
			return this.getOwner().equals(e.getOwner());
		}
		return false;
	}
}
```

21 lines (17 sloc) | 459 Bytes

```java
package piece;

import javax.swing.ImageIcon;

public class Cat extends AbstractPiece {
	public Cat(Owner owner) {
		super(null, owner, 1);
		String color = owner.equals(Owner.Player1) ? "White" : "Black";
		this.setImage(new ImageIcon("resources/" + color + " cat.png").getImage());

	}

	@Override
	public boolean equals(Object obj) {
		if (obj instanceof Cat) {
			Cat e = (Cat) obj;
			return this.getOwner().equals(e.getOwner());
		}
		return false;
	}
}
```

```java
package piece;

import java.awt.Image;

import javax.swing.ImageIcon;

public class Dog extends AbstractPiece {
	public Dog(Owner owner) {
		super(null, owner, 2);
		String color = owner.equals(Owner.Player1) ? "White" : "Black";
		this.setImage(new ImageIcon("resources/" + color + " dog.png").getImage());

	}

	public Dog(Image image, Owner owner, int rank) {
		super(image, owner, rank);
	}

	@Override
	public boolean equals(Object obj) {
		if (obj instanceof Dog) {
			Dog e = (Dog) obj;
			return this.getOwner().equals(e.getOwner());
		}
		return false;
	}
}
```

```java
package piece;

import javax.swing.ImageIcon;

public class Elephant extends AbstractPiece {
	public Elephant(Owner owner) {
		super(null, owner, 5);
		String color = owner.equals(Owner.Player1) ? "White" : "Black";
		this.setImage(new ImageIcon("resources/" + color + " elephant.png").getImage());

	}

	@Override
	public boolean equals(Object obj) {
		if (obj instanceof Elephant) {
			Elephant e = (Elephant) obj;
			return this.getOwner().equals(e.getOwner());
		}
		return false;
	}
}
```

21 lines (17 sloc) | 471 Bytes

```java
package piece;

import javax.swing.ImageIcon;

public class Horse extends AbstractPiece {
	public Horse(Owner owner) {
		super(null, owner, 3);
		String color = owner.equals(Owner.Player1) ? "White" : "Black";
		this.setImage(new ImageIcon("resources/" + color + " horse.png").getImage());

	}

	@Override
	public boolean equals(Object obj) {
		if (obj instanceof Horse) {
			Horse e = (Horse) obj;
			return this.getOwner().equals(e.getOwner());
		}
		return false;
	}
}
```

5 lines (4 sloc) | 55 Bytes

```java
package piece;

public enum Owner {
        Player1, Player2
}
```

21 lines (17 sloc) | 477 Bytes

```java
package piece;

import javax.swing.ImageIcon;

public class Rabbit extends AbstractPiece {
        public Rabbit(Owner owner) {
                super(null, owner, 0);
                String color = owner.equals(Owner.Player1) ? "White" : "Black";
                this.setImage(new ImageIcon("resources/" + color + " rabbit.png").getImage());

        }

        @Override
        public boolean equals(Object obj) {
                if (obj instanceof Rabbit) {
                        Rabbit e = (Rabbit) obj;
                        return this.getOwner().equals(e.getOwner());
                }
                return false;
        }
}
```

The internal classes of gui: LoadGameListener, MovementListener, NewGameListener, StartGameListener, where all refactored into external classes, and take a reference to a gui object