Arimaa-2.0

Trevor Burch

Tayler How

Jesse Shellabarger

Milestone 2

December 16, 2015

Final Deliverable Functionality

- **Undo Button –**Previously, the undo button would return the player to the start of their turn. This should be changed to allow for single moves being undone.
- **End Turn Button –**Previously, the player's turn would automatically end when their moves remaining was 0 and not before, this should not be the case. Player's should be able to end their turn whenever they want and their turns should not be ended automatically when they run out of moves.
- **Placing Pieces On Start –** Players should be able to assign their pieces to spaces before the start of the game. This is in accordance with the Arimaa rules and was not implemented in the initial version of the game.

Software Artifacts

- UML is located on the last page of this document.

New Features Worked On

- **Undo Button –** We changed the functionality of the undo button to only affect the last move made. Previously, the undo button would return the player to the start of their turn
- **End Turn Button –** We added a new "End Turn Button" which ends the player's turn even if they have moves left. Previously, the player's turn would automatically end when their moves remaining was 0 and not before, this is no longer the case.
- **Placing Pieces On Start –** Work was started on getting piece placement implemented. This functionality allows players to assign their pieces to spaces before the start of the game. Work was not finished on this feature and will be resumed in a future iteration.

Bad Code Smells

- **Large Class –** The GUI class is excessively long. This class has many subclasses in it that need to be extracted into their own class files. ~ **Team**
- **Long Method/Switch Statement** – The *renderInitialBoard* method of the GUI class is very long. This method is extremely long because of a switch statement in the method that handles a lot of the rendering in similar ways. This switch statement will be changed to handle things in a more efficient manner and most of the logic will be extracted into a method that can be called with parameters for drawing different things. ~ **Jesse**
- **Duplicate Code –** The *loadGame* method and *actionPerformed* method have almost identical code. This code should be pulled out into a method that can be used in both places for whenever a game needs to be created and ran. ~ **Tayler**
- **Duplicate Code –** Every time we create a button we do the exact same thing. This code could be extracted into a method that creates any button that we want on the drawing canvas. This would also fix a Long Method bad code smell in the *main* method of the GUI class. ~ **Tayler**

- **Long Method –** The *createWinWindow* method is long and could have some of its functionality extracted into new methods that do specialized units of work. ~ **Trevor**
- **Long Method –** The *mousePressed* method has a lot of complicated logic that makes the method long and hard to follow. We will extract some of this logic into methods that handle these cases more clearly ~ **Jesse**
- **Long Class –** The Game class is very long. However, this class is also more contained and some of the functionality is important to stay how it is. However, there are some switch statements and repeated logic which could be extracted into methods. ~ **Team**
- **Long Method –** The *checkWin* method has multiple places where logic could be extracted into methods. These methods will be extracted. ~ **Trevor**
- **Long Method/Switch Statement –** The *push* method has a switch statement causes the method to be extremely long. The logic of the switch statement branches will be extracted into a method that can be called from the switch statement. ~ **Trevor**

<u>Testing</u>

- This project was part of the Software Quality Assurance (CSSE 376) class in the Spring term of the 2014-15 school year. Because of this, the project has a full test suite of test cases. These test cases currently have 98+% code coverage and will be used to verify that refactoring has not broken any functionality accidentally. These test cases may be viewed using GitHub.

```
game::GUI
-p1Name: String
-p2Name: String
-activeFrames: ArrayList<JFrame>
-game: Game
-gameBoardPanel = null: ImagePanel
-boardPieces: ImagePanel[][]
-p1TextField: JTextField
-p2TextField: JTextField
-timerComboBox: JComboBox<Integer>
-moveCountLabel: JLabel
-turnCountLabel: JLabel
-turnIndicatorLabel: JLabel
-timerLabel: JLabel
-timer: TimePanel
+GUI(): ctor
+main(String[] args): void
+getP1name(): String
+setP1name(String p1name): void
+getP2name(): String
+setP2name(String p2name): void
+getActiveFrames(): ArrayList<JFrame>
+setActiveFrames(ArrayList<JFrame> frames): void
-renderInitialBoard(): void
#renderBoard(): void
+createWinWindow(): void
```

```
game::Game
~boards = new ArrayList<BoardState>(): ArrayList<BoardState>
+currentBoard = null: BoardState
~moveTimer = 0: int
~p1TimeBank = 0: int
~p2TimeBank = 0: int
~turnCounter = 0: int
~p1Name = "Player1": String
~p2Name = "Player2": String
-winner = 0: int
-numMoves = 4: int
-playerTurn = 1: int
-isPushPull: boolean
~p1Pieces: HashMap<Character, Integer>
~p2Pieces: HashMap<Character, Integer>
+Game(BoardState b): ctor
+Game(): ctor
+configurePieceInventory(): void
+placePiece(int row, int column, char piece): boolean
+removePiece(int row, int column): boolean
+getSpace(int row, int column): Piece
+move(int row, int column, int dir): boolean
-isValidMoveFromSquare(int row, int column): boolean
-isValidMoveSquare(int row, int column): boolean
-endMove(): void
-checkWin(): void
-checkDeaths(int row, int col): void
-checkFriendlyAdjacent(int row, int col): boolean
-checkStrongerAdjacent(int row, int col): boolean
-checkStrong(Piece one, Piece two): boolean
-switchPiece(int row1, int column1, int row2, int column2): void
+push(int row, int column, int dir1, int dir2): boolean
-pieceCanPush(Piece pushingPiece, Piece pushedPiece): boolean
-isValidSquareToPushFrom(int row, int column): boolean
+pull(int row1, int column1, int row2, int column2, int direction1): boolean
-tryPull(Piece space, Piece space2, int row1, int column1, int direction1): boolean
-isValidSquaretoPullFrom(int row1, int column1, int row2, int column2): boolean
+getDirection(int row1, int column1, int row2, int column2): int
+undoMove(): void
+loadFile(Scanner scanner): boolean
+saveFile(FileWriter fw): boolean
+getTurnCounter(): int
+getP1Name(): String
+getP2Name(): String
+setWinner(int winner): void
+getNumMoves(): int
+getTurnTimer(): int
+getWinner(): int
+getPlayerTurn(): int
+setPlayerTurn(int playerTurn): void
+getP1Pieces(): HashMap<Character, Integer>
+getP2Pieces(): HashMap<Character, Integer>
```

```
game::BoardState
-boardArray = new char[8][8]: char[][]
-turnNumber: int
+BoardState(char[][] map, int turnNumber): ctor
+getBoardArray(): char[][]
+setBoardArray(char[][] boardArray): void
+setBoardSpace(int row, int column, String piece): void
+getTurnNumber(): int
+setTurnNumber(int turnNumber): void
+incrementTurn(): void
+clone(): BoardState
```

```
game::ImagePanel
-serialVersionUID = -7315460075240330922L: long
-img: Image
-row: int
-column: int
+ImagePanel(String img): ctor
+ImagePanel(Image img): ctor
+createWinWindow(): void
+paintComponent(Graphics g): void
+setRow(int row): void
+getRow(): int
+setColumn(int column): void
+getColumn(): int
+getPixelX(): int
+getPixelY(): int
```

```
game::TimePanel
-timerLabel: JLabel
~timer: Timer
~playerTurn: int
+TimePanel(GUI gui, Game game, int startTime, JLabel label): ctor
+update(int s, int minute): void
+getTimerLabel(): JLabel
+setTimerLabel(JLabel timerLabel): void
```

```
game::Piece
-type: PieceType
-image: Image
-owner: Owner
-rank: int
+Piece(char c): ctor
+Piece(PieceType t, Image i, Owner o): ctor
-createP2Piece(char c): void
-createP1Piece(char c): void
+getType(): PieceType
+setType(PieceType type): void
+getImage(): Image
+setImg(Image img): void
+getOwner(): Owner
+setOwner(Owner owner): void
+getRank(): int
+setRank(int rank): void
+equals(Object p2): boolean
+isStrongerThan(Piece p2): boolean
```

```
MouseListener
+mouseClicked(MouseEvent): void
+mouseReleased(MouseEvent): void
+mouseEntered(MouseEvent): void
+mouseExited(MouseEvent): void
+mousePressed(MouseEvent): void
```

```
ActionListener
+actionPerformed(ActionEvent arg0): void
```

```
MovementListener
-selectedPiece: ImagePanel
-secondSelectedPiece: ImagePanel
+mouseClicked(MouseEvent): void
+mouseReleased(MouseEvent): void
+mouseEntered(MouseEvent): void
+mouseExited(MouseEvent): void
+mousePressed(MouseEvent): void
-moveDirectionTwoPush(ImagePanel, int, int): int
-moveDirectionOnePush(ImagePanel, ImagePanel): int
-twoPieceSelectedAndEmptySpaceClicked(int, int): boolean
-pieceSelectedAndSecondPieceClicked(int, int): boolean
-moveDirection(ImagePanel, int, int): int
-noSelectedPieceAndEmptySpaceClicked(int, int): int
-noPieceSelectedAndPieceClicked(int, int): boolean
-clockOnBoard(int, int): boolean
-checkForPush(int, int): boolean
-checkForPull(int, int): boolean
```

```
newGameListener
+actionPerformed(ActionEvent arg0): void
```

```
loadGameListener
+actionPerformed(ActionEvent arg0): void
-loadGame(File file): void
```

```
cancelListener
+actionPerformed(ActionEvent arg0): void
```

```
startGameListener
+actionPerformed(ActionEvent arg0): void
```

```
saveGameListener
+actionPerformed(ActionEvent arg0): void
```

```
undoListener
+actionPerformed(ActionEvent arg0): void
```

```
endTurnListener
+actionPerformed(ActionEvent arg0): void
```