Arimaa 2.0

Project Documentation

Jesse Shellabarger
Trevor Burch
Tayler How

Table of Contents

Introduction

Our team has developed a computer based version of the board game Arimaa. A description of the game as given by Wikipedia is:

"Arimaa was invented in 2003 by Omar Syed, an Indian American computer engineer trained in artificial intelligence. Syed was inspired by Garry Kasparov's defeat at the hands of the chess computer Deep Blue to design a new game which could be played with a standard chess set, would be difficult for computers to play well, but would have rules simple enough for his then four-year-old son Aamir to understand."
[https://en.wikipedia.org/wiki/Arimaa]

Our version of the game strives to follow all of the rules as outlined by Omar Syed. Our game has piece placing functionality on load and is turn and timer based. If a player uses all of their allotted time for a move without completing the move they forfeit the game and their opponent is declared the winner. Also, all standard Arimaa rules for winning the game are implemented in our version of the game.

The standard rules of Arimaa can be found at: http://arimaa.com/arimaa/learn/rulesIntro.html.

Our version of the game is aimed at players of any age who would just like to enjoy this new and innovative board game. Our implementation is runnable on any computer configuration as long as the standard Java JRE is installed on that system, as the game is Java dependent.

This document contains all Software Documents, User Documents, and Administrator Documents. The documents in the same categories are placed together in this document and the categories are organized in the order given above. For exact locations of all documents in the document please see the Table of Contents on page 1.

User Guide

Introduction

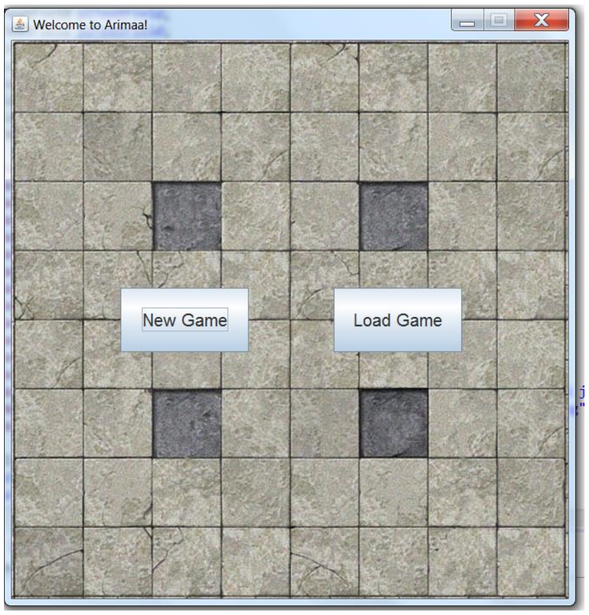
“Arimaa was invented in 2003 by Omar Syed, an Indian American computer engineer trained in artificial intelligence. Syed was inspired by Garry Kasparov's defeat at the hands of the chess computer Deep Blue to design a new game which could be played with a standard chess set, would be difficult for computers to play well, but would have rules simple enough for his then four-year-old son Aamir to understand.”
[https://en.wikipedia.org/wiki/Arimaa]

This product is a Java based implementation of the Arimaa board game which can be ran by anyone with a Java installation.

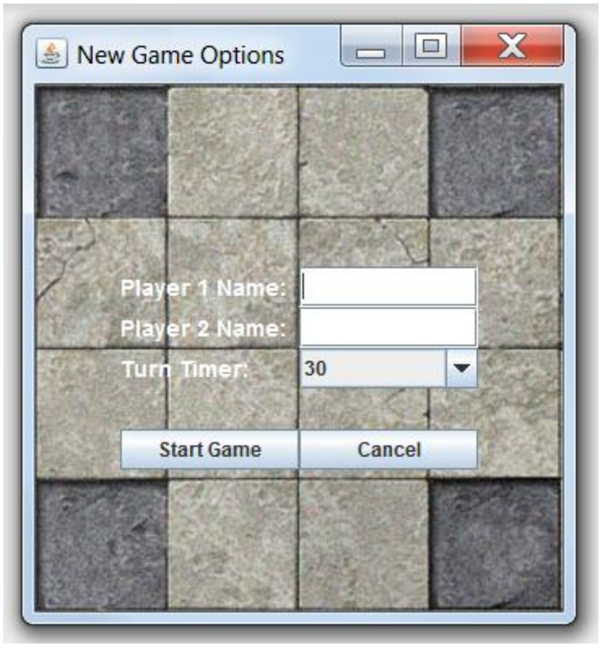
Starting a Game

New Game

A new game may be started by clicking on the “New Game” button that shows up on the startup screen.

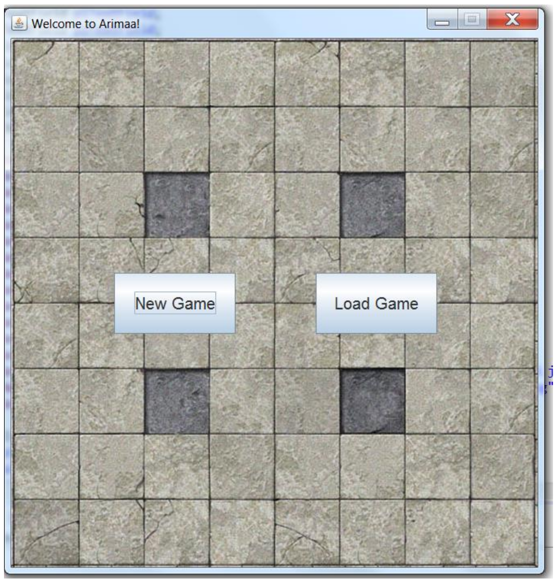


After clicking on the “New Game” button the player’s names should be entered into the text fields that are displayed. The desired turn time also is set via this window. This changes the amount of time that a player has to complete his/her moves without forfeiting the game. When these actions are completed the user may select “Start Game” to begin the game. If the user decides not to start a new game they may select “Cancel” in order to return to the start screen.

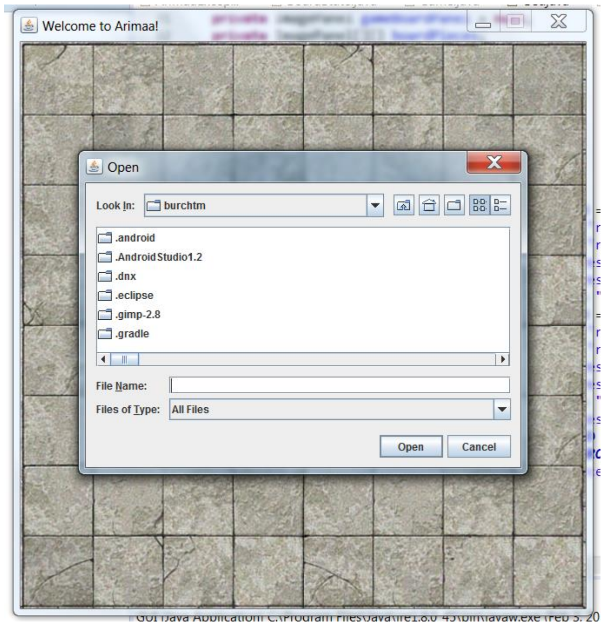


Loading a Game

A user can load a previously saved game by selecting “Load Game” on the start screen.



The save file for the previously saved game should then be selected using the file picker that is displayed and then click “Open” and the game should start from the saved state.
Note: The turn timer will be set to the full time allotted for a turn no matter what it was when the save occurred.



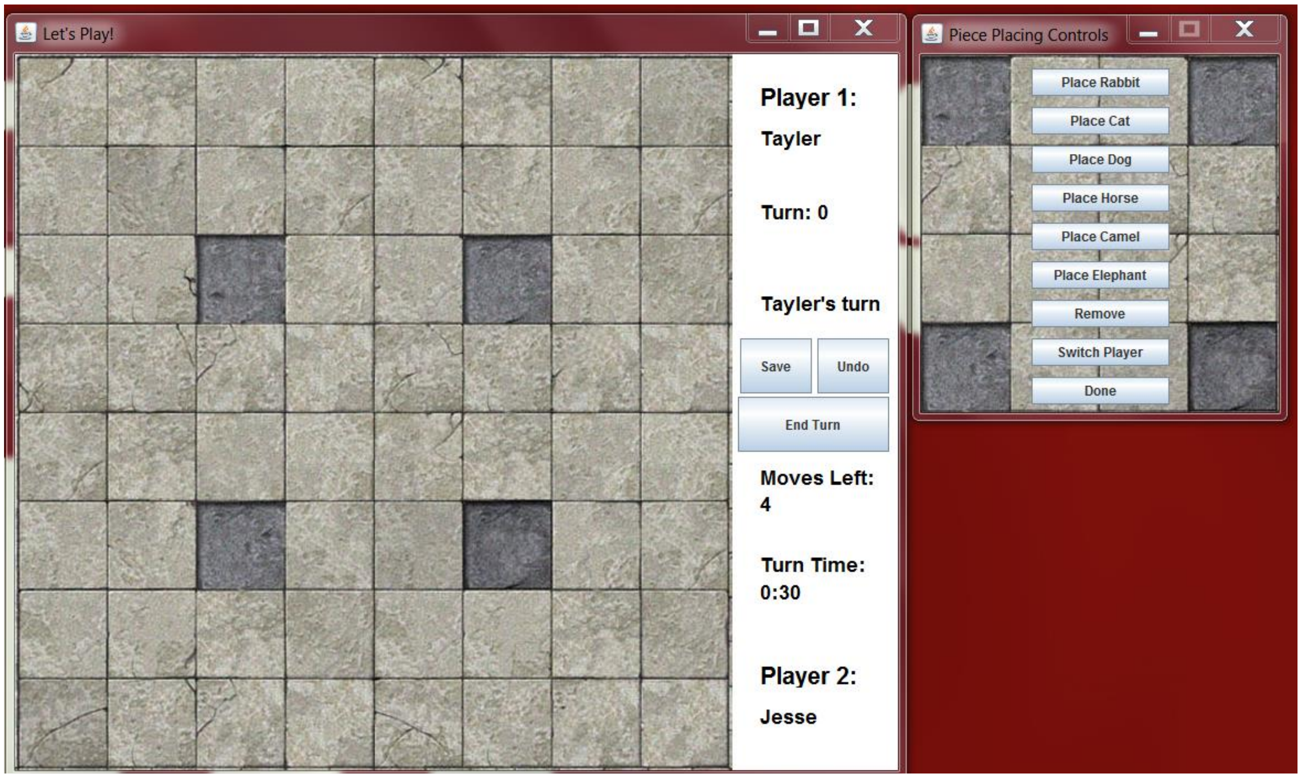
Placing Pieces

A player has the following pieces when they start the game:

- 8 Rabbits
- 2 Cats
- 2 Dogs
- 2 Horses
- 1 Camel
- 1 Elephant

When the game begins, each player will be allowed to place all of their pieces in their two home rows. Player 1 will be allowed to place all of their pieces first, followed by Player 2.

After a new game is started, a piece placement window will be displayed along with the main game board. To place a piece, the “Place…” button should be clicked for the desired piece and then the square that the piece is to be placed on should be clicked to place the piece. If the square that is clicked is not valid then the piece will not be placed.



If a player wishes to remove a piece they may select “Remove” and then click on a placed piece in order to remove it from the game board.

Once all of Player 1’s pieces have been placed, Player 1 should hit the “Switch Player” button to change the piece placement to Player 2. This will not be functional until all of Player 1’s piece have been placed on the game board. Once Player 2 has completed their placement the “Done” button should be clicked and the game will start automatically.

Playing the Game

Basics

Game play alternates between Player 1 and Player 2 with each player receiving 4 moves per turn. A player must make all of their moves within the set time limit, which was set during setup, or they forfeit the game.

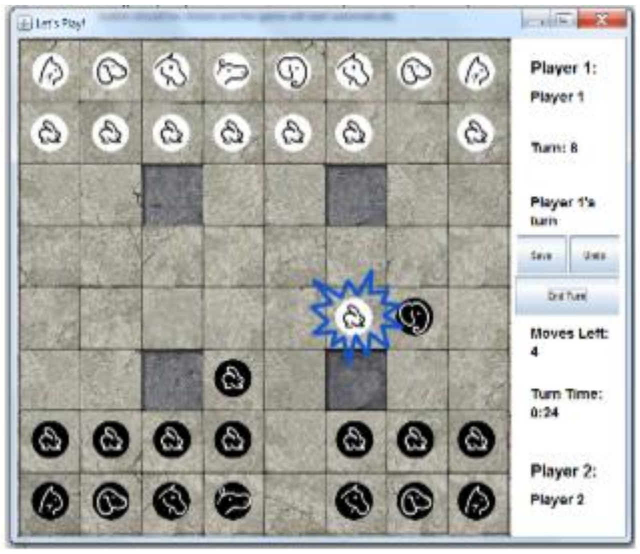
The goal of the game is for a player to move their rabbit to the last row on the opposite side of the board. If a player loses all of their rabbit pieces before achieving this goal they forfeit the game.

Moving

To move a piece, a player must first click on the piece they would like to move and then click on the space that they desire to move the piece to. This space must be adjacent to the space that the piece currently occupies.

Freezing a Piece

A piece is considered “frozen” if an enemy piece of strictly greater strength is occupying a space adjacent to it. If a piece is frozen then a player cannot move this piece unless another one of the player’s pieces is adjacent to the frozen piece.



Pushing and Pulling

During play, a player may “push” or “pull” an enemy piece that it has become adjacent to as long as the enemy piece is of a strictly lower strength than the player’s piece. Piece strengths are, from lowest to highest:

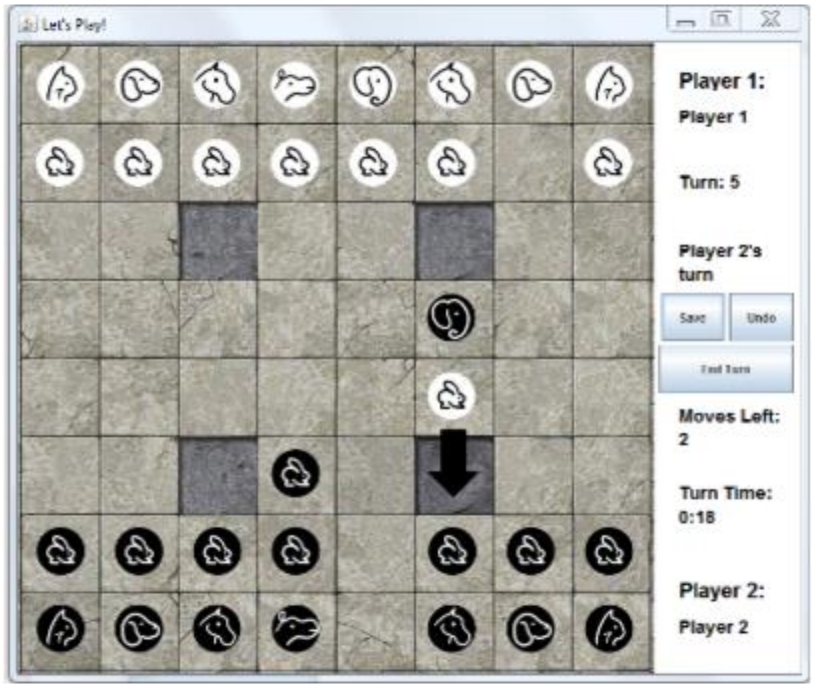
- Rabbits
- Cats
- Dogs
- Horses
- Camel
- Elephant

In order to push or pull a piece the player must first click on their stronger piece, the weaker enemy piece, and then the space that they would like to use to complete the push or pull.

Pushing or pulling an enemy piece consumes two of the player’s moves for the given turn.

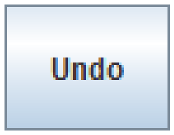
Killing A Piece

To kill an enemy's piece the enemy's piece must be pushed into one of the four darker spaces on the game board. If there is another enemy piece adjacent to the target piece then the target piece will not be killed and can be moved out of the trap space during the opponent's next turn.



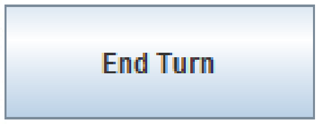
Undoing a Move

A player may undo their previous move by clicking the "Undo" button while still in their turn. The move will revert on the game board and the player will be given back the move in their move counter; however, a player will not receive the time spent making the move back. A player may not undo a move after the "End Turn" button has been clicked.



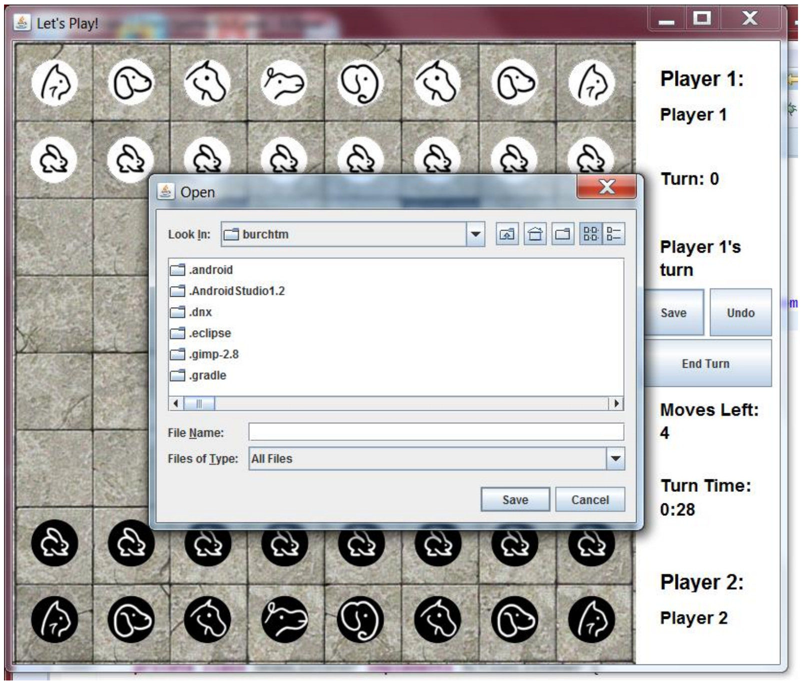
Ending a Turn

Once a player has made all of their move, the player must hit the “End Turn” button in order to officially end their turn and advance play to the other player. If a player fails to click the “End Turn” button before the turn timer runs out they will forfeit the game.



Saving a Game

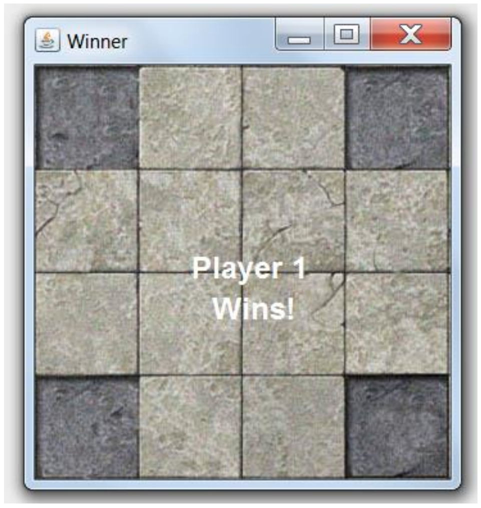
At any time the players may choose to pause and save the game by clicking the “Save Game” button. This will open a file prompt where the user may choose the file location for the save file and giving the save file a name. Once this is done, the user must hit “Save” to save the game.



Winning The Game

A player wins the game if:

- The player advances one of their rabbits to the last row on the opponent's starting side of the board
- The other player has no surviving rabbits
- The other player does not complete their turn in the time allotted



Contact Information

For further information regarding the use of this software, please contact our customer care center, available 8 a.m. - 5 p.m. EST, Monday through Friday. Email shellajt@rose-hulman.edu or call 260-438-7918. Please allow 1-2 business days (3-4 business days during the holiday season) for our customer care team to review your feedback and help provide more information.

Installation/Configuration Guide

To install and configure Arimaa:

1. Download the latest version of the game, available at: https://github.com/CSSE375-Arimaa/Arimaa-2.0/raw/master/Arimaa-2.0.zip
   You may also email our customer care representative at shellajt@rose-hulman.edu to receive a copy of the game. Please allow 1-2 business days for our customer care representative to handle your request.
2. Open the Arimaa.zip file and extract the Arimaa folder to a desired location on your computer.
3. Inside the extracted Arimaa folder, you will find two files: “Arimaa.jar” and the “resources” folder. Make sure to leave these two items in the same folder. The game cannot run correctly if the resources file is not available. A shortcut may be created to the Arimaa.jar file, if desired.
4. To start the game, simply launch the Arimaa.jar runnable.
5. Have fun playing Arimaa!

Maintenance Guide

If you experience any issues while playing Arimaa, we're very sorry. Included is a list of the most commonly reported bugs, along with some probable solutions. Please take a moment to see if the issue you are experiencing is addressed below.

- When I launch the game, the window is super tiny and the background image won't load!
  - This issue is likely due to the fact that the "Arimaa.jar" file cannot locate the "resources" folder that I was originally packaged with. Locate the .jar file and ensure that the "resources" folder can be found in the same parent directory. If this is not the case, simply move them into the same parent directory. If you cannot locate either "Arimaa.jar" or the resources folder, simply download a new copy of the game at: https://github.com/CSSE375-Arimaa/Arimaa-2.0/raw/master/Arimaa-2.0.zip
- I can't move my piece! What's going on?
  - Is your piece frozen? If your piece is adjacent to a strong enemy piece, it could be frozen. This game mechanic is described in detail in the User Guide.
  - Is it your turn? Check the turn indicator on the side menu. It will show which player is currently taking their turn. If it's not your turn, you won't be able to move your pieces. If the other player has already expended their four moves, make sure they've clicked the "End Turn" button!
- I was in the middle of taking my turn, and the game suddenly ended! It even said my opponent won!
  - Chances are you let the turn timer expire before finishing your turn. If the turn timer expires during a player's turn, the game is ended and their opponent is declared the winner. Make sure you click the "End Turn" button after making your moves to officially end your turn!
- I'm trying to switch players/start the game while placing pieces, but the game won't let me!
  - Player 1 must have all of their pieces placed before Player 2 can begin placing pieces. Similarly, both players must have all of their pieces placed before the game can begin. This game mechanic is described in detail in the User Guide.
- I moved one of my pieces and it disappeared! Why is this happening to me?!
  - Shhh.. everything will be okay.You probably moved your piece onto a trap square! If a piece is moved onto a trap square and is not adjacent to another friendly piece, it dies. Sad, huh? Luckily, you can undo your accidental suicide move by clicking the "Undo Move" button (provided that you haven't ended your turn).

If the issue you are experiencing has not been addressed, try closing and relaunching the game. If this does not resolve the issue, please contact our customer care center.

Submit any questions, comments, or bug reports our customer care center, available 8 a.m. - 5 p.m. EST, Monday through Friday. Email shellajt@rose-hulman.edu or call 260-438-7918. Please allow 1-2 business days (3-4 business days during the holiday season) for our customer care team to review your feedback and help provide a solution.

<u>SRS & SADS</u>

Software Requirements

In addition to the basic implementation of the game described in the Introduction, several requirements were specified for the development team. These features were determined to be essential to the game and were required to be implemented with the game.

Loading/Saving Games

The state of the game must be able to be saved in a platform-independent format. This record must encapsulate everything about the game, including where the players' pieces are, whose turn it is, how many moves are remaining on that turn, the names the players have chosen for themselves, and the selected turn timer. Users must be able to load this record to continue their games at a later date. The format of the save file must be platform independent, so that the record would function if it were saved on one platform, then loaded on another.

Variable Turn Timers

Players must be able to choose the length of the turn timer for their game during game setup. This timer would determine how long the player has to make their four allotted moves before forfeiting the game. The player must be able to choose from a comprehensive list of options containing a variety of standard timer lengths

Ability to Undo Moves

A player must be able to "undo" moves he has made. Moves should be "undone" one at a time. A player should be able to undo moves until he reaches the beginning of his current turn. Any further attempts to undo moves should have no effect.

Pushes and pulls must be undone as one move. If a push or a pull is undone, then both pieces must move back to their original spaces. The player who made the push or pull should be refunded his two moves for that turn.

Placing of Pieces During Setup

During game setup, players must be able to strategically place their pieces anywhere within their first two rows. Because Player 1 makes the first move, Player 1 should also place his pieces first. Once Player 1 has placed all of his pieces, Player 2 may place his. This serves to help balance the advantage Player 1 gains from making the first move.

Software Architecture and Design

Complete confidence in the game’s functionality was determined to be the critical component of the design. The development team kept this in mind as the system was designed, to ensure that the system was easily testable. Because board games are fairly static, the game’s extendability was not much of a concern.

To facilitate the testing of the game, the development team separated the game’s logic from the user interface. The game’s logic is encapsulated in the Game class. This class manages things like moving, pushing, and pulling pieces by manipulating the other classes and their information.

The BoardState class to store different states of the game. It tracks where the pieces are and the turn number. This class is used by the Game class to keep track of the board. Specifically, undoing moves and saving the game make use of this class.

The Piece class is used to store information regarding each piece. It stores which type of piece it is, along with which player owns it. This information is used to facilitate various game logic procedures.

ImagePanel and TimePanel are custom user interface components that were abstracted out of the GUI class to clean up the code. ImagePanel is used to create the various backgrounds as well as the individual piece tokens. TimePanel is used to manage the game’s turn timer and display it to the players.

The separation of the game’s logic from the user interface allows the development team to test the system by simulating input given to the game by the user interface, rather than managing the user interface directly in their tests.

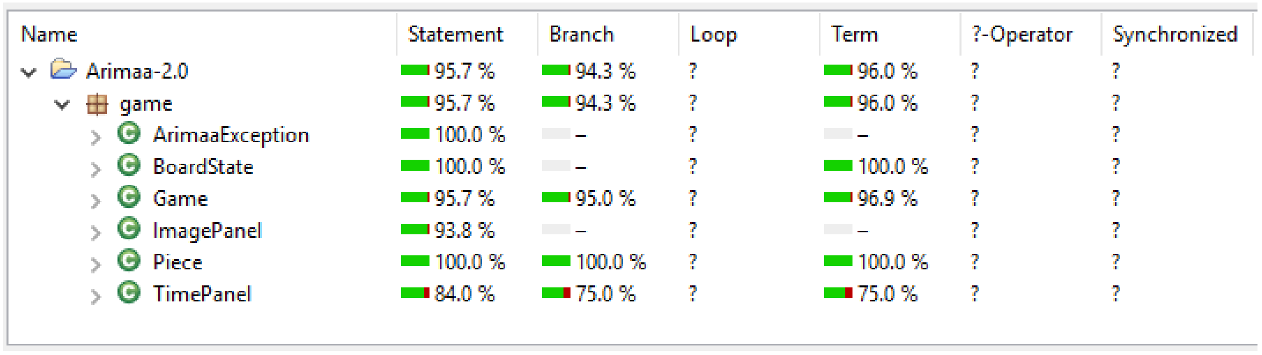
Test Suite

Several types of testing were used throughout the game’s development. Unit tests were used during the initial development phase. Integration tests were used during the secondary development phase, to integrate the graphics and IO modules with the game module. Acceptance testing was used after the full application was developed, during the improvement and refactoring phases.

Unit Testing

All initial software development was done using Test-Driven Development. As a result, a set of scripted automated unit tests was defined for each class as it was being developed. The only exception to this was the GUI class, as graphics functionally is not unit testable. These unit tests also served as a comprehensive suite of regression tests. This was particularly helpful in the refactoring phase of this project, as it allowed the developers to confidently refactor and improve the code while being sure to preserve the intended functionality.

CodeCover was used to evaluate the quality of the automatic testing suite by analyzing several code coverage metrics. CodeCover’s evaluation verified that the testing was very rigorous and thorough. The evaluation showed that the testing suite had 95.7% statement coverage, 94.3% branch coverage, and 96.0% term coverage. For a full summary of test cases see Appendix A on page 18.



Name	Statement	Branch	Loop	Term	?-Operator	Synchronized
Arimaa-2.0	95.7 %	94.3 %	?	96.0 %	?	?
game	95.7 %	94.3 %	?	96.0 %	?	?
ArimaaException	100.0 %	–	?	–	?	?
BoardState	100.0 %	–	?	100.0 %	?	?
Game	95.7 %	95.0 %	?	96.9 %	?	?
ImagePanel	93.8 %	–	?	–	?	?
Piece	100.0 %	100.0 %	?	100.0 %	?	?
TimePanel	84.0 %	75.0 %	?	75.0 %	?	?

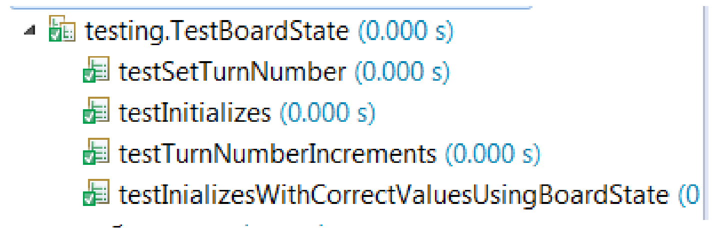
Integration Testing

Integration testing was done using Big-Bang integration testing. There were only three parts to integrate: the game model, the GUI, and the .txt files for saving and loading games. Although Big integration testing is not typically used, the development team felt that it was the correct approach because the complexity of the integrations was very low. It was very easy for the development team to identify and debug any issues that occurred during integration testing.

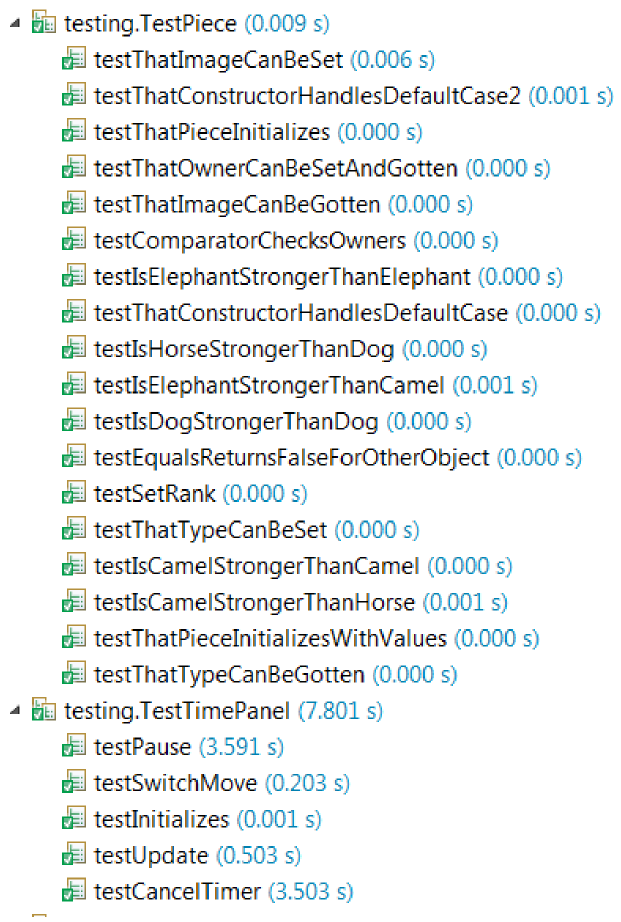
Acceptance Testing

The ultimate goal of the development team was to provide their customers with an outstanding user experience while playing the game. To ensure the quality of this user experience, the application was subjected to rigorous acceptance testing. In particular, the team focused on exploratory manual black-box acceptance testing. Testing the game from a user's perspective allowed them to identify and address any bugs that may have been overlooked during earlier phases of development. To perform this testing, the developers played through whole games countless times. The team made sure to test all game model and GUI functionality, focusing on edge and corner cases. After the development phases had been completed, the game was further tested through independent verification and validation. Several third party users were asked to play the game and provide feedback on their experience. Any bugs or issues that were reported were addressed during the refactoring phase. Users were also given the opportunity to provide suggestions for possible improvements in the game. These were evaluated by the development team and some were implemented as new features during the improvement phase.

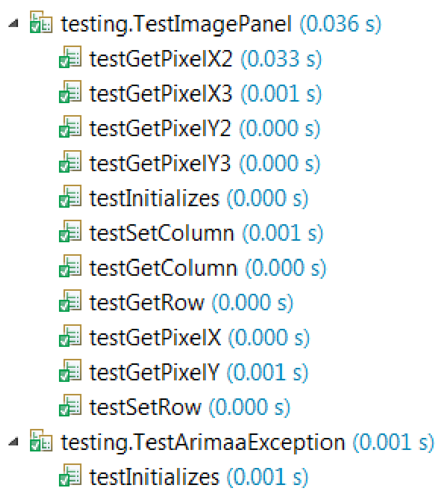
Appendix A - Summary Of Test Cases



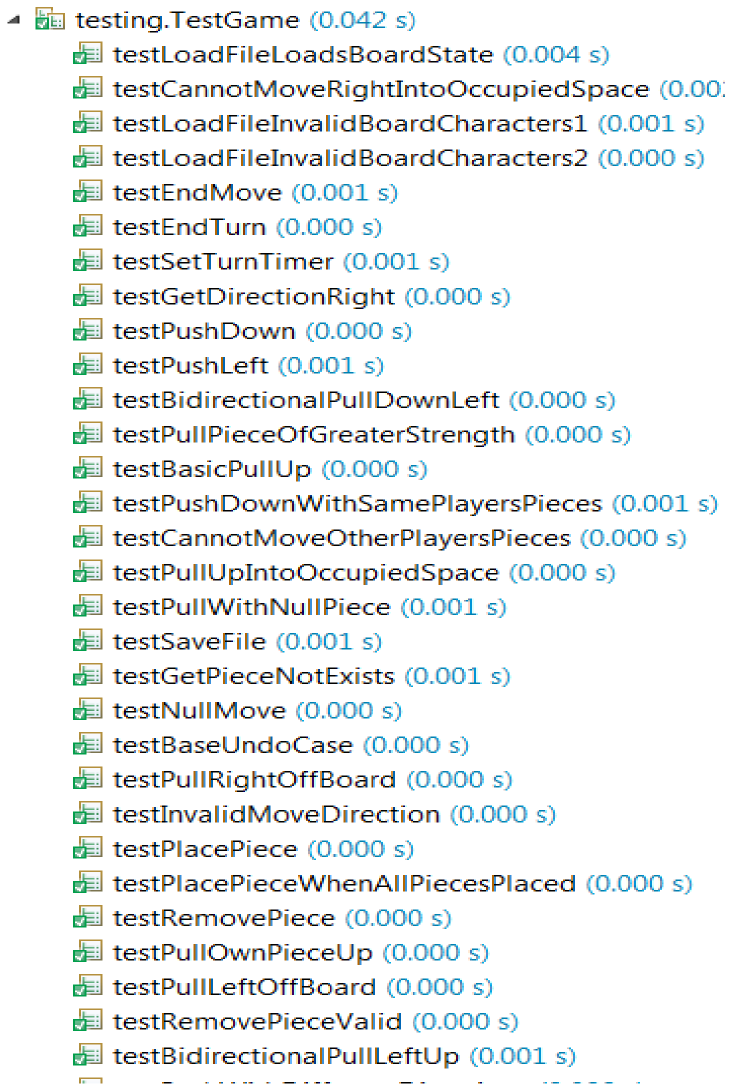
```
testing.TestBoardState (0.000 s)
    testSetTurnNumber (0.000 s)
    testInitializes (0.000 s)
    testTurnNumberIncrements (0.000 s)
    testInializesWithCorrectValuesUsingBoardState (0
```



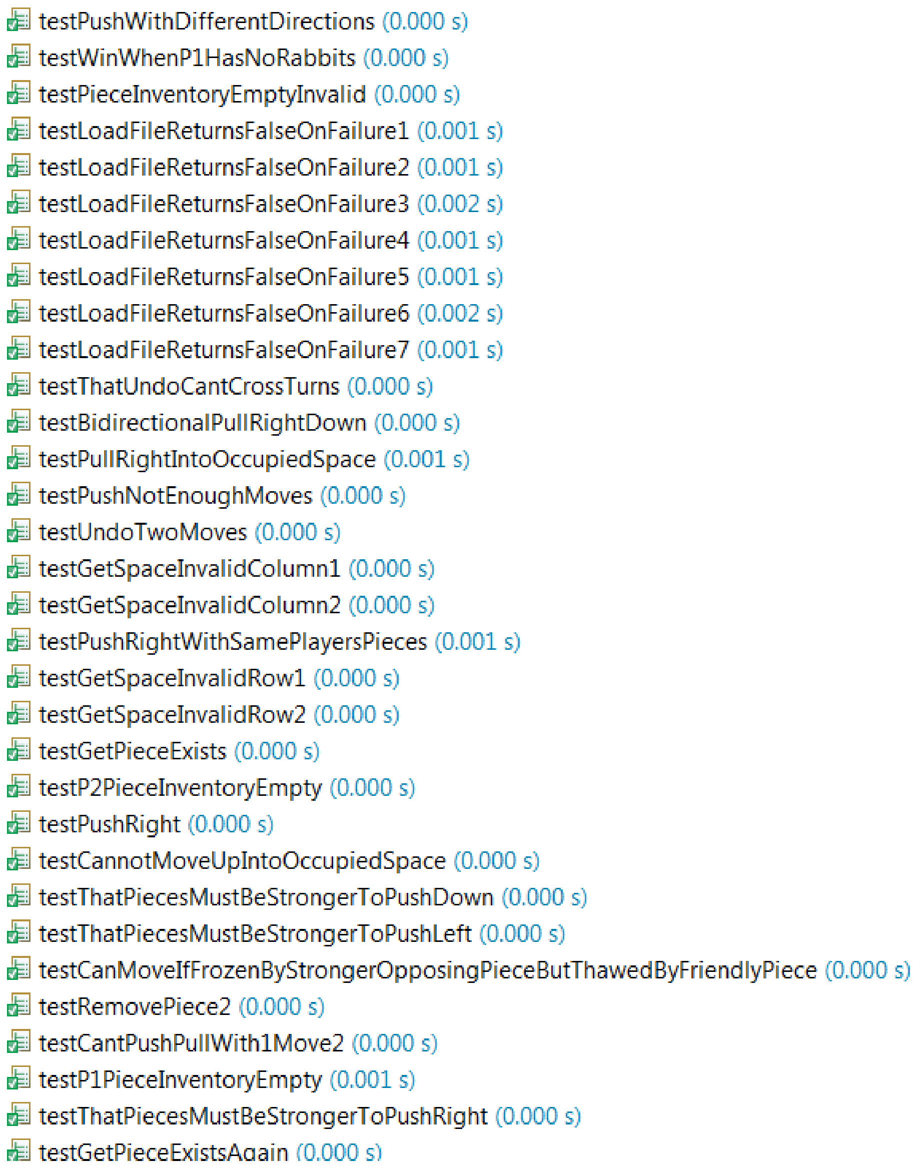
```
testing.TestPiece (0.009 s)
    testThatImageCanBeSet (0.006 s)
    testThatConstructorHandlesDefaultCase2 (0.001 s)
    testThatPieceInitializes (0.000 s)
    testThatOwnerCanBeSetAndGotten (0.000 s)
    testThatImageCanBeGotten (0.000 s)
    testComparatorChecksOwners (0.000 s)
    testIsElephantStrongerThanElephant (0.000 s)
    testThatConstructorHandlesDefaultCase (0.000 s)
    testIsHorseStrongerThanDog (0.000 s)
    testIsElephantStrongerThanCamel (0.001 s)
    testIsDogStrongerThanDog (0.000 s)
    testEqualsReturnsFalseForOtherObject (0.000 s)
    testSetRank (0.000 s)
    testThatTypeCanBeSet (0.000 s)
    testIsCamelStrongerThanCamel (0.000 s)
    testIsCamelStrongerThanHorse (0.001 s)
    testThatPieceInitializesWithValues (0.000 s)
    testThatTypeCanBeGotten (0.000 s)
testing.TestTimePanel (7.801 s)
    testPause (3.591 s)
    testSwitchMove (0.203 s)
    testInitializes (0.001 s)
    testUpdate (0.503 s)
    testCancelTimer (3.503 s)
```



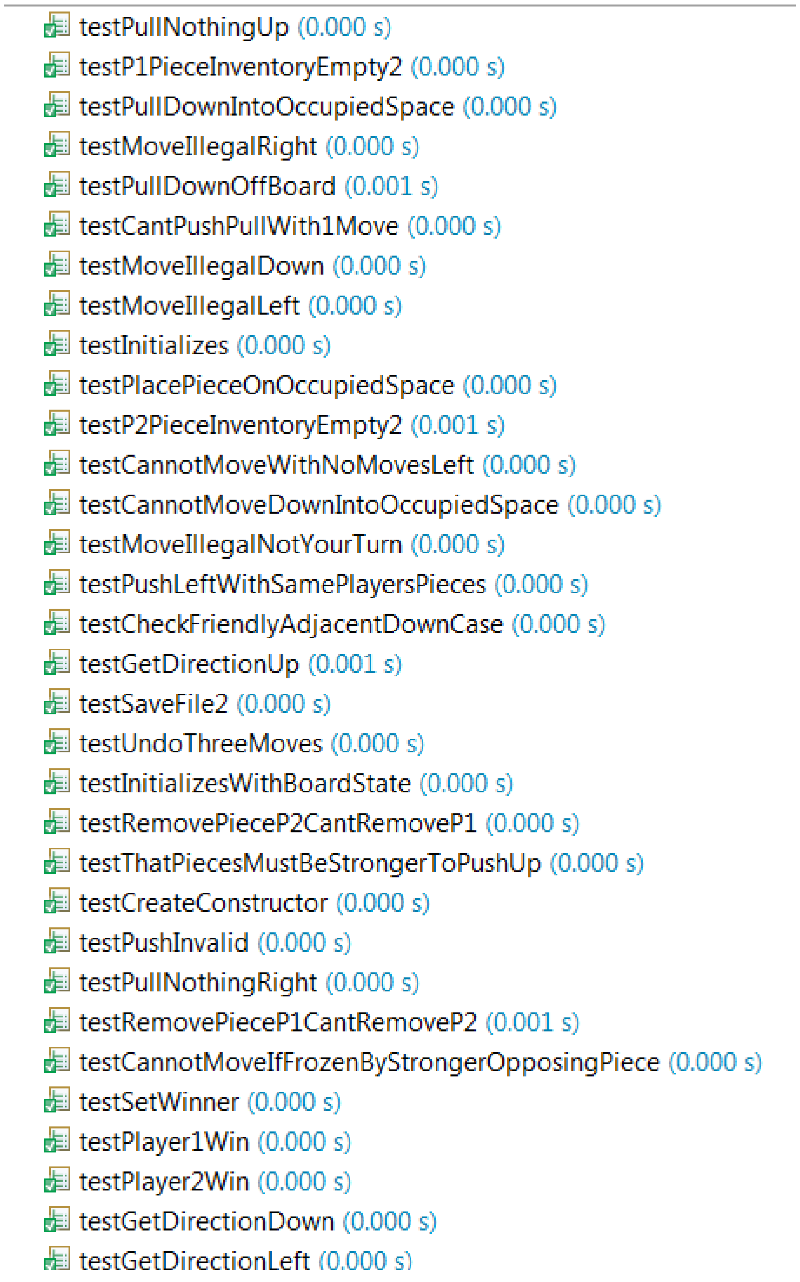
```
testing.TestImagePanel (0.036 s)
    testGetPixelX2 (0.033 s)
    testGetPixelX3 (0.001 s)
    testGetPixelY2 (0.000 s)
    testGetPixelY3 (0.000 s)
    testInitializes (0.000 s)
    testSetColumn (0.001 s)
    testGetColumn (0.000 s)
    testGetRow (0.000 s)
    testGetPixelX (0.000 s)
    testGetPixelY (0.001 s)
    testSetRow (0.000 s)
testing.TestArimaaException (0.001 s)
    testInitializes (0.001 s)
```



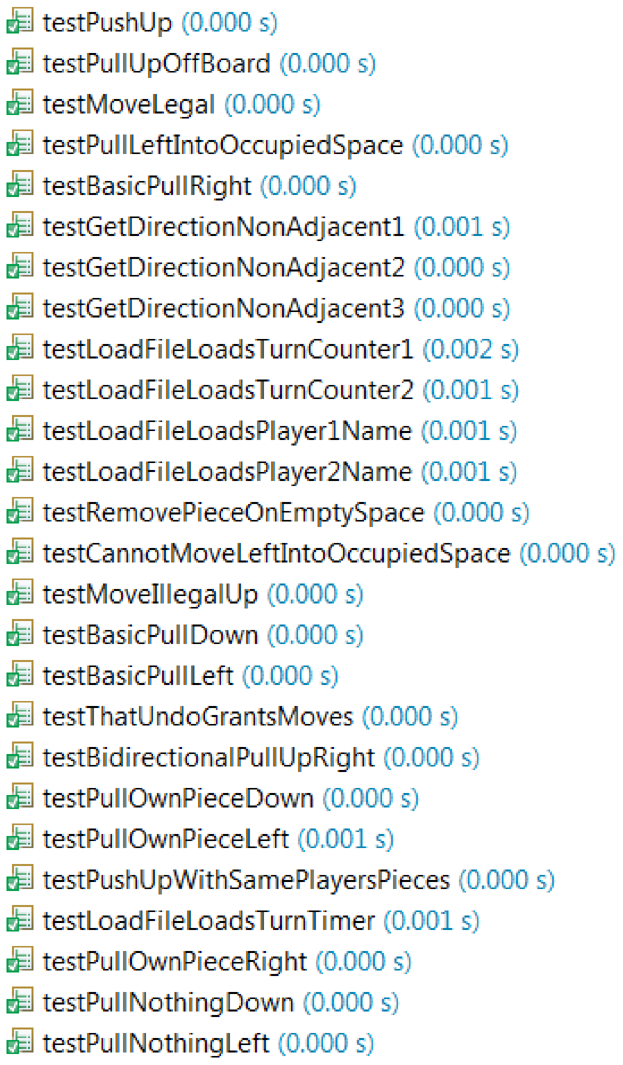
```
testing.TestGame (0.042 s)
    testLoadFileLoadsBoardState (0.004 s)
    testCannotMoveRightIntoOccupiedSpace (0.00
    testLoadFileInvalidBoardCharacters1 (0.001 s)
    testLoadFileInvalidBoardCharacters2 (0.000 s)
    testEndMove (0.001 s)
    testEndTurn (0.000 s)
    testSetTurnTimer (0.001 s)
    testGetDirectionRight (0.000 s)
    testPushDown (0.000 s)
    testPushLeft (0.001 s)
    testBidirectionalPullDownLeft (0.000 s)
    testPullPieceOfGreaterStrength (0.000 s)
    testBasicPullUp (0.000 s)
    testPushDownWithSamePlayersPieces (0.001 s)
    testCannotMoveOtherPlayersPieces (0.000 s)
    testPullUpIntoOccupiedSpace (0.000 s)
    testPullWithNullPiece (0.001 s)
    testSaveFile (0.001 s)
    testGetPieceNotExists (0.001 s)
    testNullMove (0.000 s)
    testBaseUndoCase (0.000 s)
    testPullRightOffBoard (0.000 s)
    testInvalidMoveDirection (0.000 s)
    testPlacePiece (0.000 s)
    testPlacePieceWhenAllPiecesPlaced (0.000 s)
    testRemovePiece (0.000 s)
    testPullOwnPieceUp (0.000 s)
    testPullLeftOffBoard (0.000 s)
    testRemovePieceValid (0.000 s)
    testBidirectionalPullLeftUp (0.001 s)
```



```
testPushWithDifferentDirections (0.000 s)
testWinWhenP1HasNoRabbits (0.000 s)
testPieceInventoryEmptyInvalid (0.000 s)
testLoadFileReturnsFalseOnFailure1 (0.001 s)
testLoadFileReturnsFalseOnFailure2 (0.001 s)
testLoadFileReturnsFalseOnFailure3 (0.002 s)
testLoadFileReturnsFalseOnFailure4 (0.001 s)
testLoadFileReturnsFalseOnFailure5 (0.001 s)
testLoadFileReturnsFalseOnFailure6 (0.002 s)
testLoadFileReturnsFalseOnFailure7 (0.001 s)
testThatUndoCantCrossTurns (0.000 s)
testBidirectionalPullRightDown (0.000 s)
testPullRightIntoOccupiedSpace (0.001 s)
testPushNotEnoughMoves (0.000 s)
testUndoTwoMoves (0.000 s)
testGetSpaceInvalidColumn1 (0.000 s)
testGetSpaceInvalidColumn2 (0.000 s)
testPushRightWithSamePlayersPieces (0.001 s)
testGetSpaceInvalidRow1 (0.000 s)
testGetSpaceInvalidRow2 (0.000 s)
testGetPieceExists (0.000 s)
testP2PieceInventoryEmpty (0.000 s)
testPushRight (0.000 s)
testCannotMoveUpIntoOccupiedSpace (0.000 s)
testThatPiecesMustBeStrongerToPushDown (0.000 s)
testThatPiecesMustBeStrongerToPushLeft (0.000 s)
testCanMoveIfFrozenByStrongerOpposingPieceButThawedByFriendlyPiece (0.000 s)
testRemovePiece2 (0.000 s)
testCantPushPullWith1Move2 (0.000 s)
testP1PieceInventoryEmpty (0.001 s)
testThatPiecesMustBeStrongerToPushRight (0.000 s)
testGetPieceExistsAgain (0.000 s)
```



```
testPullNothingUp (0.000 s)
testP1PieceInventoryEmpty2 (0.000 s)
testPullDownIntoOccupiedSpace (0.000 s)
testMoveIllegalRight (0.000 s)
testPullDownOffBoard (0.001 s)
testCantPushPullWith1Move (0.000 s)
testMoveIllegalDown (0.000 s)
testMoveIllegalLeft (0.000 s)
testInitializes (0.000 s)
testPlacePieceOnOccupiedSpace (0.000 s)
testP2PieceInventoryEmpty2 (0.001 s)
testCannotMoveWithNoMovesLeft (0.000 s)
testCannotMoveDownIntoOccupiedSpace (0.000 s)
testMoveIllegalNotYourTurn (0.000 s)
testPushLeftWithSamePlayersPieces (0.000 s)
testCheckFriendlyAdjacentDownCase (0.000 s)
testGetDirectionUp (0.001 s)
testSaveFile2 (0.000 s)
testUndoThreeMoves (0.000 s)
testInitializesWithBoardState (0.000 s)
testRemovePieceP2CantRemoveP1 (0.000 s)
testThatPiecesMustBeStrongerToPushUp (0.000 s)
testCreateConstructor (0.000 s)
testPushInvalid (0.000 s)
testPullNothingRight (0.000 s)
testRemovePieceP1CantRemoveP2 (0.001 s)
testCannotMoveIfFrozenByStrongerOpposingPiece (0.000 s)
testSetWinner (0.000 s)
testPlayer1Win (0.000 s)
testPlayer2Win (0.000 s)
testGetDirectionDown (0.000 s)
testGetDirectionLeft (0.000 s)
```



```
testPushUp (0.000 s)
testPullUpOffBoard (0.000 s)
testMoveLegal (0.000 s)
testPullLeftIntoOccupiedSpace (0.000 s)
testBasicPullRight (0.000 s)
testGetDirectionNonAdjacent1 (0.001 s)
testGetDirectionNonAdjacent2 (0.000 s)
testGetDirectionNonAdjacent3 (0.000 s)
testLoadFileLoadsTurnCounter1 (0.002 s)
testLoadFileLoadsTurnCounter2 (0.001 s)
testLoadFileLoadsPlayer1Name (0.001 s)
testLoadFileLoadsPlayer2Name (0.001 s)
testRemovePieceOnEmptySpace (0.000 s)
testCannotMoveLeftIntoOccupiedSpace (0.000 s)
testMoveIllegalUp (0.000 s)
testBasicPullDown (0.000 s)
testBasicPullLeft (0.000 s)
testThatUndoGrantsMoves (0.000 s)
testBidirectionalPullUpRight (0.000 s)
testPullOwnPieceDown (0.000 s)
testPullOwnPieceLeft (0.001 s)
testPushUpWithSamePlayersPieces (0.000 s)
testLoadFileLoadsTurnTimer (0.001 s)
testPullOwnPieceRight (0.000 s)
testPullNothingDown (0.000 s)
testPullNothingLeft (0.000 s)
```