**Test Plan**

for

Game of Chess

Version 1.0 approved

Prepared by Condor Army

3/3/18

# Contents

[1.1 Contents i](#_Toc507944157)

[1.2 Revisions i](#_Toc507944158)

[1.3 Introduction 1](#_Toc507944159)

[1.3.1 Test Plan Objectives 1](#_Toc507944160)

[2.1 Test Strategy 1](#_Toc507944162)

[2.1.1 System Test 1](#_Toc507944166)

[2.1.2 User Interface Test 1](#_Toc507944167)

[2.1.3 Documentation Test 2](#_Toc507944168)

[2.1.4 Beta Test 2](#_Toc507944169)

[2.1.5 User Acceptance Test 2](#_Toc507944170)

[3.1 Environment Requirements 3](#_Toc507944172)

[3.1.1 Environment 1 3](#_Toc507944173)

[3.1.2 Environment 2 3](#_Toc507944174)

[4.1 Functions To Be Tested 3](#_Toc507944176)

[5.1 Test Results 3](#_Toc507944178)

[5.1.1 System Test 3](#_Toc507944179)

[5.1.2 User Interface Test 3](#_Toc507944180)

[5.1.3 Documentation Test 3](#_Toc507944181)

[5.1.4 Beta Test 3](#_Toc507944182)

[5.1.5 User Acceptance Test 3](#_Toc507944183)

# Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| Init 1 | Dustin Rowe | Completed Sections 1 and 2, describing the game of chess and the test plan, and describing, in detail, the test strategy for this application. | 03/04/18 |

# Introduction

The Game of Chess is an application that allows a player to play chess on their computer against the computer as their opponent. The application also allows the user to set up the color of each of the team’s pieces as well as add a move timer to the game that gives the player an extra challenge to pick a move before a certain amount of time.

### Test Plan Objectives

This Test Plan will be used to test and validate and verify that The Game of Chess has all functionality and features requirements according to the Software Requirements Specification. These tests will also execute test scripts to ensure no defects remain within the application, and if defects are to be found, will be fixed and retested. By reaching the end of this Test Plan with no defects one can be assured that the Game of Chess is stable for release and production-ready.



# Test Strategy



### System Test

System testing will be performed to verify that the Game of Chess meets all specified requirements from the Software Requirements Specification. This test will also verify that all classes and functions are in working condition together and will also act as a means of smoke testing. This test will be performed by playing through the game several times, testing different settings and scenarios that would be typical for any player to perform. This will include testing the changing of piece colors, the implementation of the move timer, and performing a checkmate against the computer, as well as having the computer checkmate you. This test can be considered accepted if no defects are found through the above processes and scenarios, in which case they shall move to User Interface Testing. In the case that the application fails this test, immediate debugging should be completed, and defective code should be altered.

### User Interface Test

User Interface testing will be performed to test how the Game of Chess performs under multiple user flows of input as well as test all User Interface components within the application. This includes the pieces on the board, the board itself, and the main menu. This test will be performed first by attempting to do any and everything to each component of the User Interface individually, including trying to move a piece to an invalid area, pressing a button multiple times, and clicking any and every part of the board for any major defects. After this is accepted than the testing goes over to user flow, where multiple paths through the application will be tested. This includes having the tester play through the game multiple different ways, seeing if the system acts any different when playing a knight first other than a pawn. This will also go through the menu as well, testing to observe if any defects are found in the gameplay by changing setting to a certain way. This test can be considered accepted if no defects are found through the above user flows and User Interface components, in which case they shall move to Documentation Testing. In the case that the application fails this test, debugging and testing shall be completed on the corresponding component of set of components if defect found in user flow.

### Documentation Test

Documentation testing will be performed to verify that the previously written documents, as well as this document itself, carries out the specifications needed within the Game of Chess application to perform as intended. This includes the testing of the Software Requirements Specification, the Software Design Document, and this Test Plan. This test will be performed by looking towards the Software Requirements Specification and looking through each requirement, observing if that requirement is intended and works within the Game of Chess program, as well as looking through the rest of the document for any defects regarding the application. Once no defects can be found, and each requirement can be considered necessary, then the Software Requirements Specification is considered accepted and testing moves on to the Software Design Document. This test will be performed by looking through the UML designs that code was implemented from and whether there are any defects with the design itself, as well as looking if any valuable functions and/or classes are missing or defective within the designs. Once no defects are found, then the Software Design Document can be considered accepted and testing moves on to the Test Plan. This test will be performed by observing the test strategies and whether they cover the full application and will be able to detect the maximum number of defects that can be found, as well as checking over the test results to see if they matched the original test strategy. Once no defects are found, then the Test Plan can be considered accepted and testing moves on to the Beta Test. In the case that any of these documents fail in this test, any defects regarding the document should be rewritten and in any cases that may affect other documents or code, should have those defects removed (and replaced if needed) as well.

### Beta Test

Beta testing will be performed to more thoroughly test the Game of Chess under more normal scenarios. This test will allow users outside the development team to test the application on their own computers and observe any possible defects that may be occur within the game. Since no testers will have any experience with the inner code of this application, this test will also serve as our black box testing. This test will be performed by distributing an executable version of the Game of Chess to a select number of people, each of which will be asked to test the application however they wish and in the case of any defects to provide information about the defect such as what the player was doing at the time and the symptoms of the defect, including no user input being allowed, the piece going to the wrong position, etc. This test can be considered accepted if no defects are found by the beta testers, in which case they shall move to User Acceptance Testing. In the case that a defect is found, the application will go back to developers to attempt to recreate the problem as the tester described and to fix the defect.

### User Acceptance Test

User Acceptance testing will be performed to completely test the Game of Chess under scrutinized testing by the Condor Army organization. This test will have testers go through the application multiple times to find any other possible defects within the application, as well as verify that all requirements are still met after all other tests and defects have been fixed. Since those testing the application have helped in developing the Game of Chess, this test will also server as our white box testing. This test will be performed by having the Condor Army team meet up and openly test all aspects of the application together, communicating on what they are testing and working on. If any defects are found during this process, then the development team should start working with whoever it was that found the defect to reproduce it and fix the defect. This test can be considered accepted if no more defects can be found within the application after the last defect was fixed, in which case the Game of Chess is ready to go into production.



# Environment Requirements

### Environment 1

Windows 10 PC: Intel i7-1790K CPU, Radeon RX 480 GPU, 24 GB of RAM.

Runs extremely well, consistent FPS. No exclusive defects.

### Environment 2



# Functions To Be Tested

### GetMoves()

GetMoves() is expected to take nothing as its arguments, and is expected to return a list of “Coordinates” which each have their own property x and z. These coordinates should correspond to positions in which the piece calling the function is available to move to. This was tested by naturally using the HighlightTiles() function to highlight every tile that is contained from the list of Coordinates returned by the GetMoves() function.

### Move()

Move() is expected to take a Coordinate as its argument, and is expected to actively move the piece on the board as well as update the two-dimensional array containing all pieces, as well as reset the board and deactivate any pieces that the piece may have captured. The function itself, however, does not return anything. This function was tested by looking at the two-dimensional array, as well as the board itself to ensure they match what is expected when a tile corresponding to a possible tile is clicked.

### IsCheck()

IsCheck() is expected to take a Coordinate as its argument, and is expected to return a list of “ChessPieces” which each corresponds to a piece on the board that is causing the piece which called the function’s king. This was tested by implementing a debug logging statement within the IsCheck() function at each possible point a piece could be added to the return list. The application is then played through multiple scenarios in which a check could be possible, as well as where a check should not be possible, like for instance if another piece is blocking the king.

### IsCheckmate()

IsCheckmate() is expected to take a Chesspiece, which should be the king of the piece which called the function. The function is also expected to return a Boolean of true if there is a checkmate on the king given in the argument, and false otherwise. This was tested by implementing a debug logging statement after each move to test if there is checkmate for either team1, team2, both, or neither. The application is then played through multiple scenarios in which a checkmate is made for team1, then team2, then neither.

### HighlightTiles()

HighlightTiles() is expected to take a list of Coordinates to highlight as its argument, and is expected to return nothing. This was tested by naturally clicking on multiple pieces on the application to observe if tiles are highlighted corresponding to their possible moves.

### SetColor()

SetColor() does not expect any arguments nor is it expected to return anything. Rather, SetColor() is merely used to set color of pieces on the board to colors set up within the settings menu before the start of the game. This is tested by naturally playing through the application, changing the settings to multiple possible colors for both teams, and playing the game to test that the pieces are in fact the color chosen.