**Test Plan**

for

**Hound Army Checkers**

Version 1.1 approved

Prepared by Samantha Coyle, Spencer Albert, Taylor Moralez, and Rowan Stone

CS3398 Fall 2018: Software Engineering, Section 001

October 14, 2018

# Contents

[Revisions ii](#_Toc117484243)

[1 Introduction 1](#_Toc117484244)

[1.1 Purpose 1](#_Toc117484245)

[1.2 System Overview 1](#_Toc117484246)

[1.3 Definitions, Acronyms and Abbreviations 1](#_Toc117484247)

[1.4 Supporting Materials 1](#_Toc117484248)

[1.5 Document Overview 1](#_Toc117484249)

[2 Architecture 2](#_Toc117484250)

[2.1 Overview 2](#_Toc117484251)

[2.2 Component 1..n 2](#_Toc117484252)

[3 High-Level Design 3](#_Toc117484253)

[4 Low-Level Design 4](#_Toc117484254)

[4.1 Modules Overview 4](#_Toc117484257)

[4.2 Module Specifications 4](#_Toc117484258)

[4.2.1 Module X1 4](#_Toc117484259)

[4.2.2 Module X2 5](#_Toc117484260)

[Appendix A – Group Log 6](#_Toc117484261)

# Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| V1.1 | Samantha Coyle | Included test function. | 10-29-18 |

# Introduction

This is the Test Plan document for Hound Army Checkers, a stand-alone browser application that simulates the classic board game of the same name. Checkers is a turn-based strategy game in which the objective for both players is to capture all their opponent’s pieces or leave their opponent without any playable moves. Legal moves are constrained to only forward, diagonal moves for standard pieces, which can include single or multi-jumps to capture pieces. For king pieces, diagonal moves backward are also allowed. This Test Plan has been developed in order to provide a description of the means and purpose for verifying and validating the operation and functionality of the game.

### Test Plan Objectives

The purpose of this document is to verify and validate that the code in HAC is mostly free of bugs or performance issues, and that the user requirements have been met and tested. Further unit testing will be done on the object's methods, and negative testing will be done to rule out invalid user input.

* HAC shall allow the user to reference the player manual during the game.
* HAC shall allow the user to turn music/sound on or off at will.
* HAC shall display a shot clock for the players during the game.
* HAC shall display whose turn it is during the game.
* HAC shall only allow each player to move their own pieces during their turn.
* HAC shall highlight optional moves for a piece when selected.
* HAC shall allow the user to move a piece to their selected destination as long as the move is valid.
* HAC shall not move a piece for the user if the user’s chosen move for that piece is invalid.
* HAC shall provide an accurate and up-to-date score on the scoreboard.
* HAC shall remove captured pieces from the board.
* HAC shall king a player’s piece as soon as their piece reaches king’s row on their opponent’s side.
* HAC shall notify the user when the game has ended and then stop gameplay.
* HAC shall notify the user if they have won or lost the game.
* HAC shall allow the user to test the functionality of the game if desired.

# Test Strategy

### System Test

The test approach for system testing is to test that all the product functions, components and subcomponents of HAC are working properly, as listed in the HAC SRS. The product functions and features, along with the required steps for basic testing are listed below.

1. HAC shall allow the user to reference the player manual during the game.
   1. The tester will open up HAC and click on the player manual.
   2. The tester will ensure that the manual opens and displays the rules for HAC.
2. HAC shall allow the user to toggle music/sound on or off at will.
   1. The tester will open HAC and click on the play button for the music, and check that the music starts playing.
   2. The tester will then, while the music is playing, press the pause button, and check that the music stops playing.
   3. The tester will then click on the volume button to check that the music mutes.
   4. The tester will then, while the music is muted, press the volume button again, and check that the music starts playing again.
3. HAC shall notify the user when it is their turn to move.
   1. The tester will start a game of HAC, upon which they should ensure that underneath ‘Current Players Turn’ reads ‘black.’
   2. The tester will then select and move one of their pieces, and check that the screen now reads ‘white’ under ‘Current Players Turn.’
   3. Once the AI has moved, the tester will ensure that the text returns to displaying ‘black.’

1. HAC shall only allow each player to move their own pieces during their turn.
   1. The tester will start a game of HAC and select and move one of the player’s pieces.
   2. The tester, once it is their turn again, will attempt to select a piece that is not theirs, and check that the game engine will not let them.
2. HAC shall allow the user to move a piece, to the selected space, as long as the move is valid.
   1. The tester will start a game of HAC and select and move one of the player’s pieces to a valid location.
   2. The tester, once it is their turn again, will attempt to select a piece and move it to an invalid location, and ensure that the game engine does not let them.
3. HAC shall provide an accurate and up-to-date score on the scoreboard.
   1. The tester will start a game of HAC and start playing a game, moving pieces until they have an opportunity to capture an enemy piece.
   2. The tester will then opt to jump over an enemy piece, and check that once they do, the scoreboard increments player 1’s score by 1.
4. HAC shall remove captured pieces from the board.
   1. The tester will start a game of HAC and start playing a game, moving pieces until they have an opportunity to capture an enemy piece.
   2. The tester will then opt to jump over an enemy piece, and check that once they do, the enemy piece is cleared from the board.
5. HAC shall king a player’s piece as soon as their piece reaches king’s row on their opponent’s side.
   1. The tester will start a game of HAC and move their pieces until one has reached the row before the back row of the opposing player.
   2. The tester will move their piece into the back row, and check that the game makes the piece a king.
   3. The tester will verify that the newly kinged piece is able to move both forwards and backwards.
6. HAC shall notify the user when the game has ended and then stop gameplay.
   1. The tester will start a game of HAC and play the game as normal.
   2. The tester will continue until the game has finished, and they have either won or lost.
   3. The tester will then check that HAC has notified them that the game is over.
   4. The tester will then attempt to move their pieces after the game has ended and ensure that HAC does not allow them to continue gameplay.
7. HAC shall notify the user if they have won or lost the game.
   1. The tester will start a game of HAC and play the game as normal.
   2. The tester will continue until the game has finished, and they have won.
      1. If the tester finishes the game and has lost, they will go back to step (10.1) and repeat until a game has been won.
   3. The tester will check that HAC has notified them that they have won the game.
   4. The tester will start a new game of HAC and play the game as normal.
   5. The tester will throw the game and attempt to lose.
      1. In the event that the tester wins the game, the tester will go back to step (10.4) and try harder to lose the game.
   6. The tester will ensure that once the game is finished, HAC notifies them that they have lost the game.
8. Shot Clock
   1. The tester will start a game of HAC and check that both entries in the shot clock start at 0, and that player 1’s shot clock starts counting upwards.
   2. The tester will then move a piece, and once they have done so, ensure that player 1’s shot clock stops counting upwards (but retains its last number) and that player 2’s shot clock starts counting upwards.
   3. The tester will then wait for the AI to make its move, at which point the tester should check that player 2’s shot clock stops counting and that player 1’s shot clock starts counting up from the number that it last reached during the previous turn.
   4. The tester will then make a move, and ensure that player 2’s shot clock acts in a similar manner to the conditions in step (11.3).
9. Hint Squares
   1. The tester will start a game of HAC and select a piece to move.
   2. The tester will check that the correct possible squares to move to have been highlighted.
   3. Without moving that piece, the tester will select another piece and check that the original highlighted squares are now un-highlighted, and the correct squares for the new piece are highlighted.
   4. The tester will then move the currently selected piece and ensure that the previously highlighted squares have now been un-highlighted.
10. Test Button
    1. The tester will start a game of HAC and select the test button.
    2. The tester will click “ok” on the alert.
    3. The tester will click on a valid placement for the checker piece to ensure that valid input works.
    4. The tester will repeat steps 13.1-13.2, and then try to move the piece to an invalid sqaure to ensure that invalid moves do not work.

### Stress/Performance Test

Negative testing will be done on HAC to test how the game will handle unexpected scenarios and bad user input.

1. Play a regular game and attempt to move pieces to invalid locations.
2. Attempt to stress the system by attempting to move to invalid locations multiple times in a row.
3. Attempt to stress the system by clicking on different pieces in a semi-random fashion over and over.
4. Attempt to stress the system by repeatedly clicking the start over button.

### Automated Test

Due to the nature of the application, there will be no automated testing for HAC.

# Environment Requirements

### Environment 1

The first environment to be tested in is the Google Chrome internet browser.

### Environment 2

The second environment to be tested in is the Mozilla Firefox internet browser.

# Functions To Be Tested

**Initialization/ Misc.**

*drawWorld()*

*startTheClock()*

*init()*

*refreshPage()*

*anonymous manual function*

**Piece Selection**

*selectPiece()*

*isSelectableChecker()*

*hintSquare()*

*unhintSquare()*

**Piece Movement**

*movePiece()*

**Testing**

*test()*