**Test Plan**

for

CentipedeArmy Chess Game

Version 1.0 approved

**Prepared by Emmanuel (Tobi) Afolayan, Drew Grubb, Jed Hutto, Jesse Miara, Ryan Weeks**

CS 3398.264

February 16, 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 |
| --- | --- | --- | --- |
| 1.0 | Drew Grubb Jed H. Ryan W. Jesse M. Tobi A. | Basic implementation of the Test Plan for Centipede Chess | 03/02/2018 |

# 1 Introduction

The following pages serve to give an overview of a Test Plan for the chess game we designed.

# Test Plan Objectives

Upon running the program, users should be able to navigate the various menus, from starting up the program to completing a game with another player or an AI, without facing substantial difficulties, such as bugs or runtime errors. Should errors occur during test runs, our code shall be inspected and modified based on the magnitude of the error. This test plan is to help ensure that any bugs or issues in code that we find can be identified and fixed in upcoming patches.

The Verification of centipede chess was successful. The program runs and navigates through every game state without crashing. Everything is rendered where it is supposed to be, and is buffered correctly. However, on extremely slow machines, if the machine cannot update the game at 60TPS, the timers will slow down.

Our Validation of this project works as intended. The user can navigate through a Menu State and a Play State, successfully setting up different game modes based on the choices they make. Once in the Play State, the user can successfully play chess until a winner is determined, either by Checkmate, by forfeit, or by running out of time in timed mode. Each piece moves as intended, and Check and Checkmate are found successfully. Every button, when pressed, performs the desired action to the best of our knowledge, and are reset with the creation of a new game or the changing of a menu state.

1. **Test Strategy**

# System Test

Centipede Chess is a stand-alone Java project. It is not a part of a system, therefore it does not require an extensive system test beyond testing the program itself.   
We will be testing Centipede with a variety of methods including manual testing and Unit Tests.

# Stress/Performance Test

To test how the game worked on different types and speeds of computers, this program was run on a variety of machines:

Fast Computers: We chose to run Centipede Chess on an Intel I7 Quad Core processor to test the performance of the game, and the game ran extremely quickly. Due to us creating the game loop to run off real time, a fast computer was capped out at what it could render. The game ran smoothly, and the timers ran at the appropriate speed.

Normal Computers: To test performance on the average computer, we chose to run centipede chess on an Intel I5 dual core processor. The game ran smoothly at 60TPS, and timers ran at appropriate speed.

Slow Computers: This game was not tested on a slow computer, but theoretically on a computer that cannot run the game at 60TPS, the timers will run slower and normal because they are updated based on the update speed of the game. In the future, we plan on running this program on a slower computer, and if necessary, updating the algorithm so that the timers do not depend on system updating.

# Automated Test

Due to the simplicity of Centipede Chess, automated tests are not a part of this Test Plan. Any testing will be done by hand or by Unit Testing.

# Recovery Test

Depending on the severity of the error, the game can either recover immediately or would crash entirely. Java automatically throws exceptions when it runs into an error, and if the error is expected we can catch it and work on it accordingly. However, if an error is not expected and is thrown, there is a possibility of the JVM crashing entirely. In this event, there is no fail safe, and the user will have to restart the program.

# Documentation Test

Eclipse has a built in Javadoc Documentation Generator, so testing is not required beyond the initial program.

# Beta Test

The extension of Beta Testing in this project will be testing by the developers. We will click each menu button in every possible combination to search for the desired result. We will also simply play a game of chess using every piece to test the success of the game.

# User Acceptance Test

Before releasing this project, we will test that everything works as intended with as few errors as possible. This will be done with a variety of manual testing and unit testing. Due to the GUI, most bug catching can be done manually by simply playing games of chess and recording any errrors that seem to appear or compromise the program entirely.

1. **Environment Requirements**

# Environment 1

Our first testing environment for this project was on Jed's Fedora (Linux) Environment, where the program ran as intended. Java's built in multi-environment support handles the bulk of these changes.  
  
Processor: Intel I7 Quad Core

# Environment 2

Our second testing environment for this project was on a Windows Machine, where the program ran as intended.  
  
Processor: Intel I5 Dual-Core

1. **Functions To Be Tested**

Update - The various update methods throughout the game loop need to be tested to ensure that the state of the game is constantly updated as intended.

Render – The various render methods throughout the game loop need to be tested to ensure that every piece, button, text, image, and board are being drawn where they are supposed to be drawn when they are supposed to be drawn.

Update Possible Moves – The bulk of the game algorithms, update possible moves needs to be extensively tested to make sure that the pieces can only move where they are supposed to when the game allows them to. Any failure in this testing will cause the game to be unplayable due to easy cheating or game breaking strategies.