**Software Design  
Document**

for

Flame Checkers

Version 1.3 approved

Zachary Golla, Connor Oldmixon,

Cameron Stevenson, Sergio Cardoza-Molina, Md Masum Billah

Flame Army

23 October 2018

# Table of Contents

Table of Contents i

Revisions ii

1. Introduction 1

1.1 Purpose 1

1.2 System Overview 1

1.3 Definitions, Acronyms and Abbreviations 1

1.4 Supporting Materials 1

1.5 Document Overview 1

2. Architecture 1

2.1 Overview 1

2.2 Component 1..N 1

3. High-Level Design 2

3.1 View / Model Component 1..n 2

3.2 A.I IMPLEMENTATION……………………………………………………………………………………2

4. USE CASES 2

4.1 USE CASE POSSIBILITIES 1..n 2

# Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Primary Author(s) | Description of Version | Date Completed |
| 1.0 | Zachary Golla | Initial Creation | 10/15/2018 |
| 1.1 | Zachary Golla | Class Diagram/Sequence/High-Level/Use Case Skeletons/Starts | 10/17/2018 |
| 1.2 | Connor Oldmixon | Inserted correct class diagrams for FlameCheckersGUI and OptionWindow classes | 10/23/2018 |
| 1.3 | Zachary Golla | Inserted Use Cases and correct class diagrams for all remaining classes | 10/29/2018 |

# Introduction

**1.1 Purpose**

This document contains the software design description for the Flame Checkers App. Within this document includes Flame Checker’s software architecture, interactions and communication between classes of the application’s design, and descriptions of how those classes will properly work with each other to execute Flame Checkers.

**1.2 System Overview**

Flame Checkers will be written in Java following an object-oriented design using the UML. Using an object-oriented design will allow us to design object classes that will provide separate functionalities of Flame Checkers and work in conjunction with each other to provide a complete executable. Each object class will then define the objects in the system and their interactions. This design will allow us to easier make change compared to other functional approaches, providing us an ability to change the implementation of one object without affecting another’s.

**1.3 Definitions, Acronyms, and Abbreviations**

|  |  |
| --- | --- |
| **Abbreviation** | **Term** |
| FC | Flame Checkers |
| GB | Game Board |
| AI | Artificial Intelligence |
| JRE | Java Runtime Environment |
| UI | User Interface |

**1.4 Supporting Materials**

“Learn to Play Checkers.” *The Spruce Crafts.com*, Chess.com, www.thesprucecrafts.com/play-checkers-using-standard-rules-409287

<https://www.thesprucecrafts.com/play-checkers-using-standard-rules-409287>

\*\*\*\*\*\* Need to find a checkers AI not Chess \*\*\*\*\*\*

Hartikka, Lauri. “A Step-by-Step Guide to Building a Simple Chess AI.” *FreeCodeCamp.org*, Medium, 30 Mar. 2017

Description and guide using minimax on how to implement FC AI.

<https://medium.freecodecamp.org/simple-chess-ai-step-by-step-1d55a9266977>

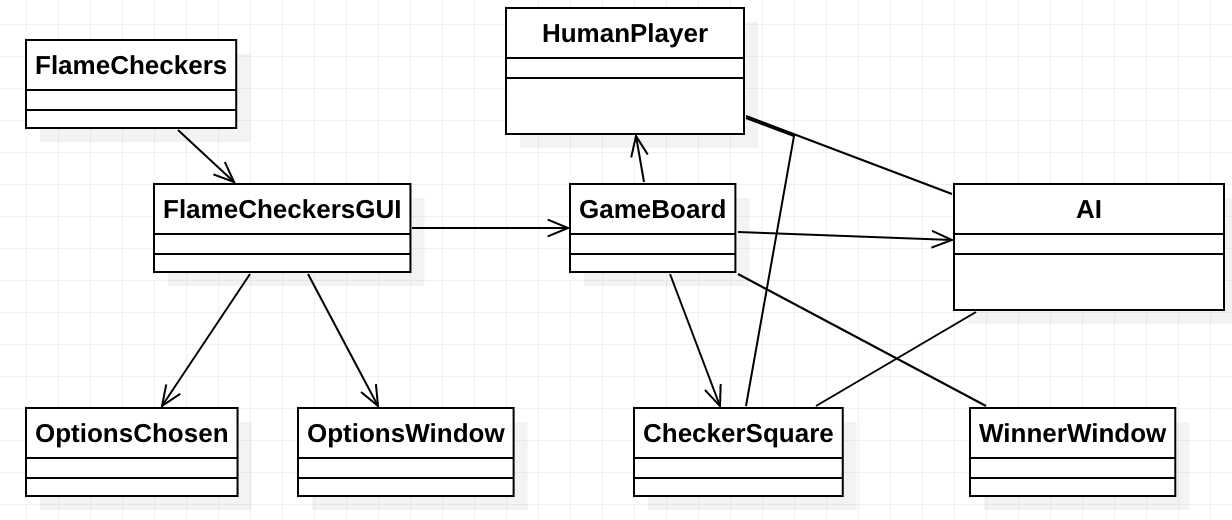
**1.5 Document Overview**

This document will contain information relative to the design and creation of Flame Checkers. The first section of this section, Section 1, contains an introduction to the document, its purpose, and provides an overall structure to direction of its design. The second section of the document will include the object-oriented architecture of Flame Checkers. This section will include the blueprint of Flame Checkers as-well-as the relationships between the different object classes and the objects they include. Section 3 contains an over-view of high-level design including different states of Flame Checkers and the communication of the data between a user’s decisions and the system. Finally, the fourth section will display a series of use cases pivotal to the execution of Flame Checkers that feature user’s progression through the application when each action is performed.

# Architecture

# Overview

Flame Checkers High-Level design Component Diagram:

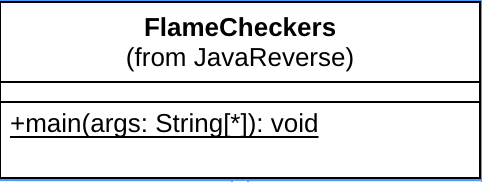


This is a test design and will need to be updated\*\*

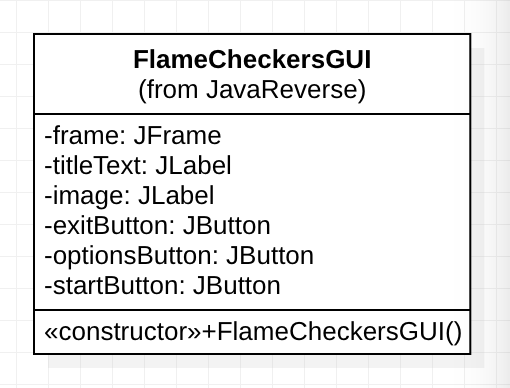
From the High Level design Component Diagram Flame Checkers will have a central application, that when run, will trigger a GUI. From the GUI a user will be able to chose from a variety of option windows: one to exit the application entirely, one to allow a user to chose options for their upcoming game, and finally an option to start the game. We did this to allow features of the game to be set before the game starts. That way once it does, the game knows how to initially setup the board and continue with the game. This should allow for future implementation of new features in an object-oriented manner if changes or upgrades to the game are needed in the future. Once the start game option is chosen a board window will appear. At the same time a board appears, player data is built, implemented, and then also placed onto the board. As a player or players continue to make choices, features within the system will trigger, until a game is complete. Once completed a winner window will appear, signifying the game is over, and allowing the user to chose which choice they would like next.

# Component 1..n

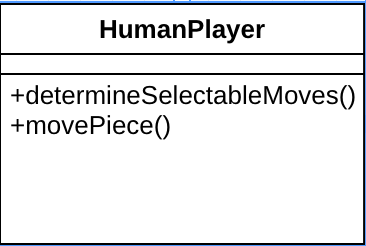
**2.2.1 FlameCheckers Class**



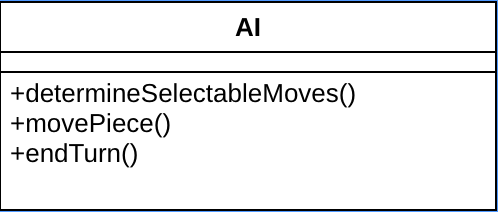
**2.2.2 GUI Class**

****

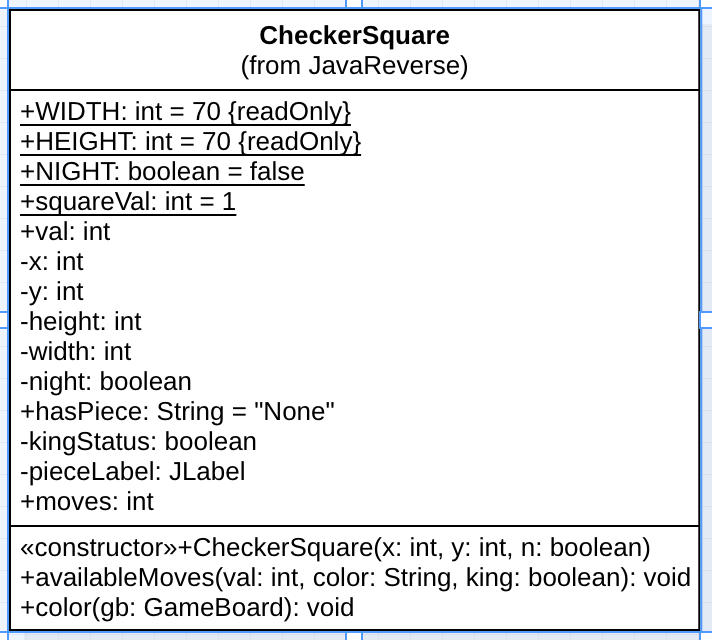
**2.2.3 HumanPlayer Class**



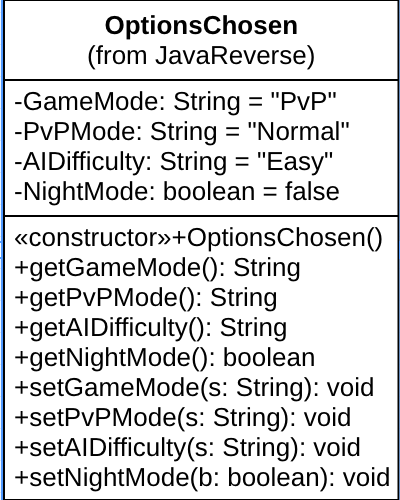
**2.2.4 AI Class**



**2.2.5 CheckerSquare Class**

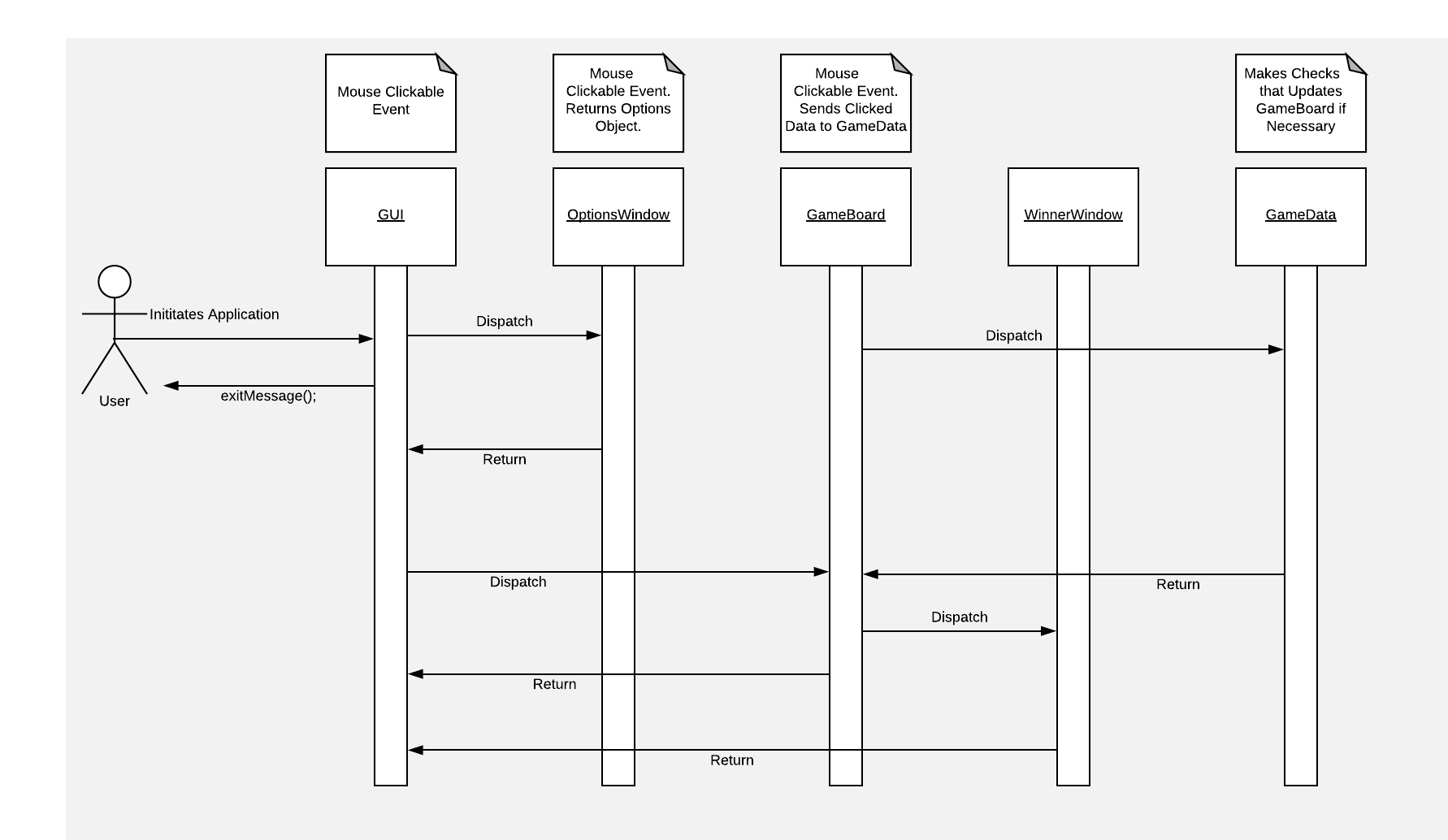


**2.2.6 OptionsChosen Class**



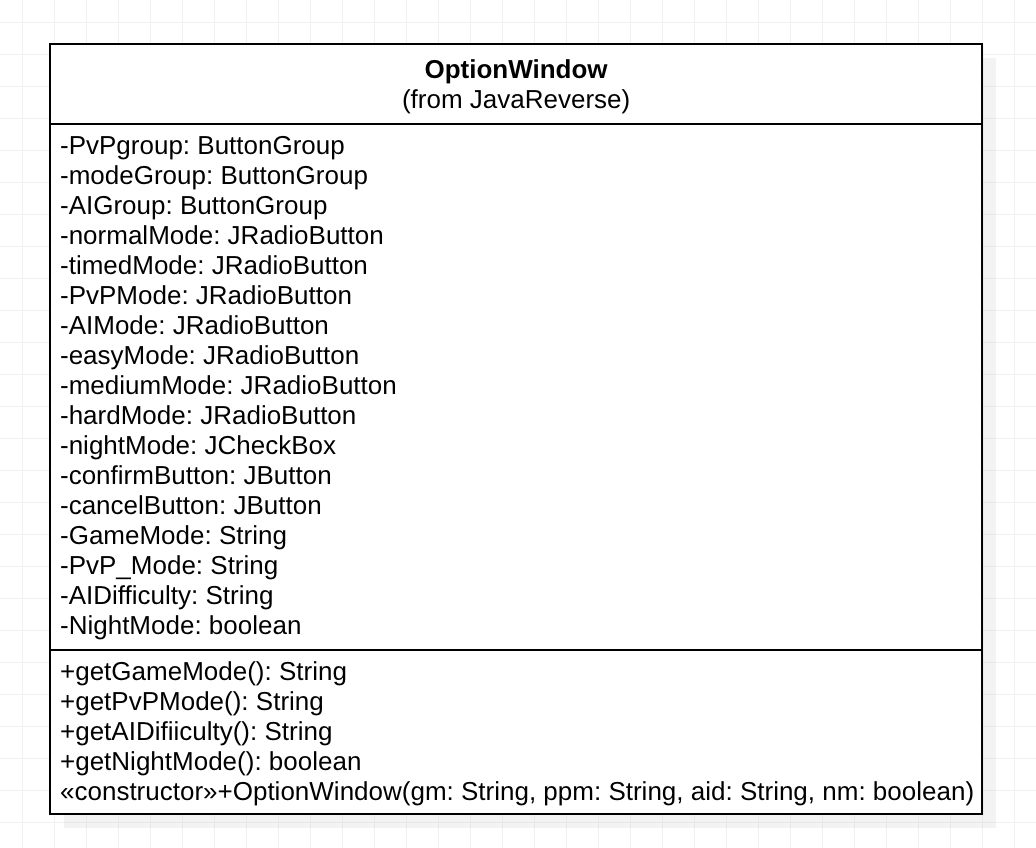
# High-Level Design

Flame Checkers shall accept input form the user through mouse clicks on buttons in the menus and on the cells of the chess board during the game. The following sequence diagram shows how input shall be handled by the view components and the flow of data back to the Model. From there, the observable Board class shall update the view directly the newest version of the board state, which then highlights its cells accordingly.

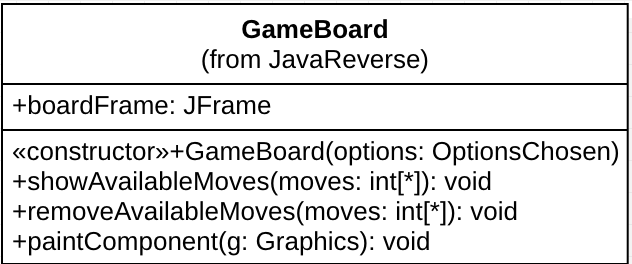


## View / Model Component 1..n

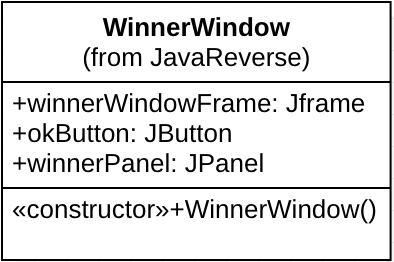
* + 1. **OptionWindow Class**

****

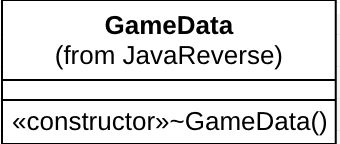
* + 1. **GameBoard Class**

****

* + 1. **WinnerWindow Class**

****

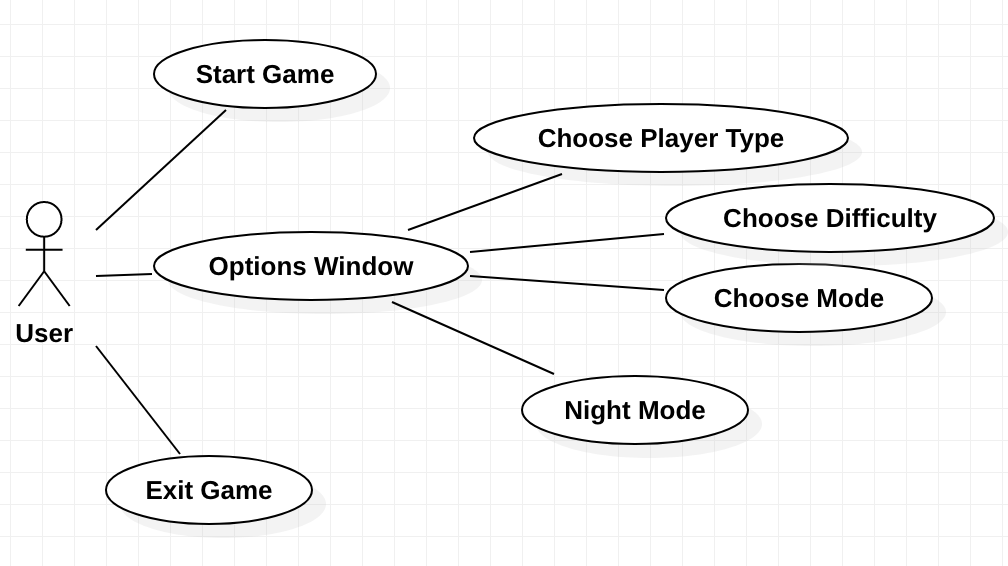
* + 1. **GameDataWindow Class**

****

# Use Cases

## Use Case Possiblities 1..n

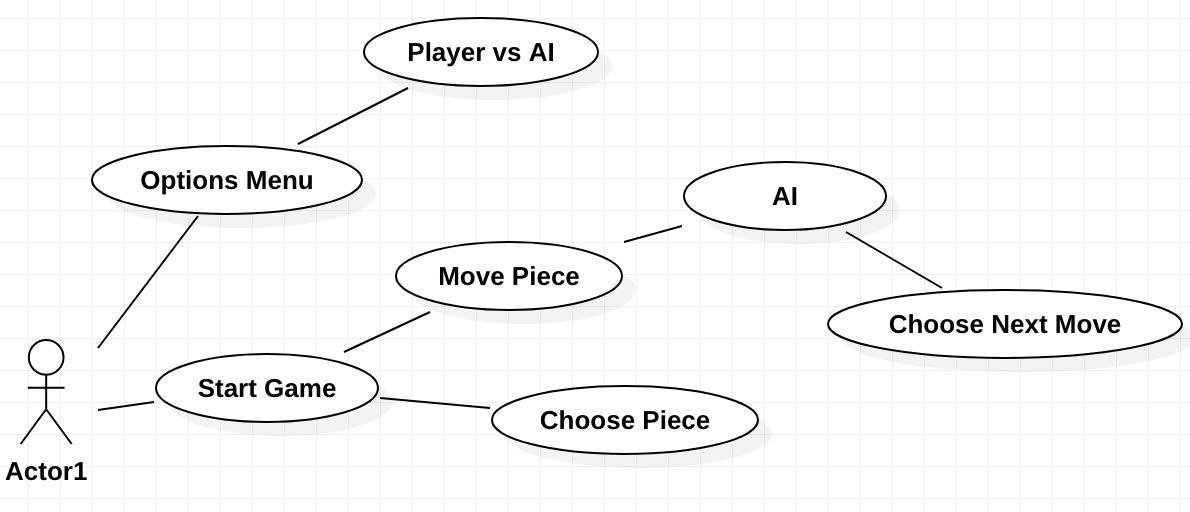
* + 1. **Start Game/Options/Exit Game**

****

Preconditions: None

Main Flow: One the user starts the game, they will have the option to exit the game, open the options window, or start the game

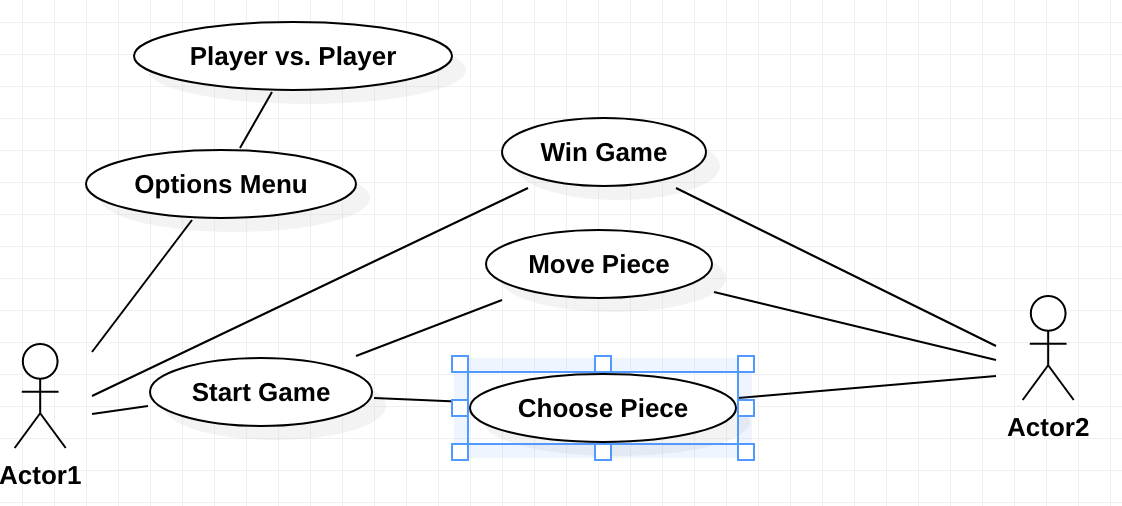
* + 1. **Player vs. AI**

****

Preconditions: Must be in game

Main Flow: User will start the game, open the options menu and select AI options as-well-as difficulty. The user will then start the game and move their first piece. After moving first piece the AI will trigger and will move its own piece accordingly.

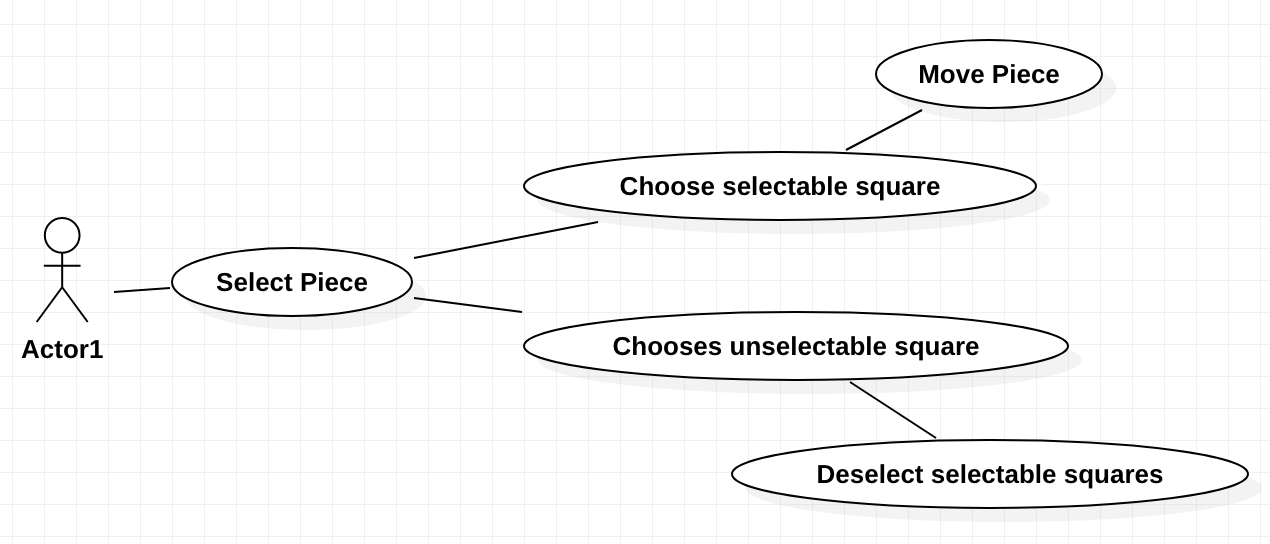
* + 1. **Player vs. Player**

****

Preconditions: Must be in game

Main Flow: A player will open the options menu and choose player vs player. The user will then start the game, choose their first piece, and move it. The second user will repeat the same process until one of the users has won the game.

* + 1. **Move Piece**



Preconditions: Must be in game

Main Flow: A user will select a piece he/she will want to move. The selected piece will highlight selectable squares the selectable piece can move to. If the user selects a square it can move to, the piece will move; otherwise, the highlighted will become no longer highlighted and the user will have to select a piece again.