**Software Design Document**

**for**

**Hound Army Checkers**

**Version 1.3 approved**

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

**CS3398 Fall 2018: Software Engineering, Section 001**

**September 12, 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

# Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Primary Author(s) | Description of Version | Date Completed |
| v1.1 | Taylor Moralez | Changed extra features from king mode and avatar selection to shot clock and hint square. | 10-23-18 |
| V1.2 | Samantha Coyle | Updated AI feature functionality. | 10-24-18 |
| V1.3 | Samantha Coyle | Added test button. | 10-29-18 |

# Introduction

## Purpose

The purpose of this document is to provide a detailed description of the design specifications for the Hound Army Checker Board game. This document will cover the details mostly regarding the design specifications, but will also touch on the functionality, features, and requirements for the game. Hound Army shall provide the user a functional, high-quality checkers game. This document is version 1.0. The SDD scope will include a full description of the design requirements and functional deliverables that the Hound Army shall produce.

## System Overview

HAC will be utilize a web-based interface. The main components will consist of a web browser and the GE, whose subcomponents will be discussed in the following sections. The web browser will allow viewability and functionality of the front-end components of the GE to allow the user to provide input for the back-end components of the GE.

## Definitions, Acronyms and Abbreviations

The naming standards included within this document are as stated below. The acronym is on the left and the full-description of the acronym is on the right-hand side.

|  |  |
| --- | --- |
| **Acronym** | ***Definition*** |
| HAC | *Hound Army Checkers* |
| *GUI* | *Graphical User Interface.* |
| *AI* | *Artificial Intelligence* |
| *GE* | *Game Engine* |

## Supporting Materials

Ceta, Noel. “All You Need to Know About UML Diagrams: Types and 5 Examples.” *Tallyfy*,

Tallyfy, 14 Sept. 2018, tallyfy.com/uml-diagram/.

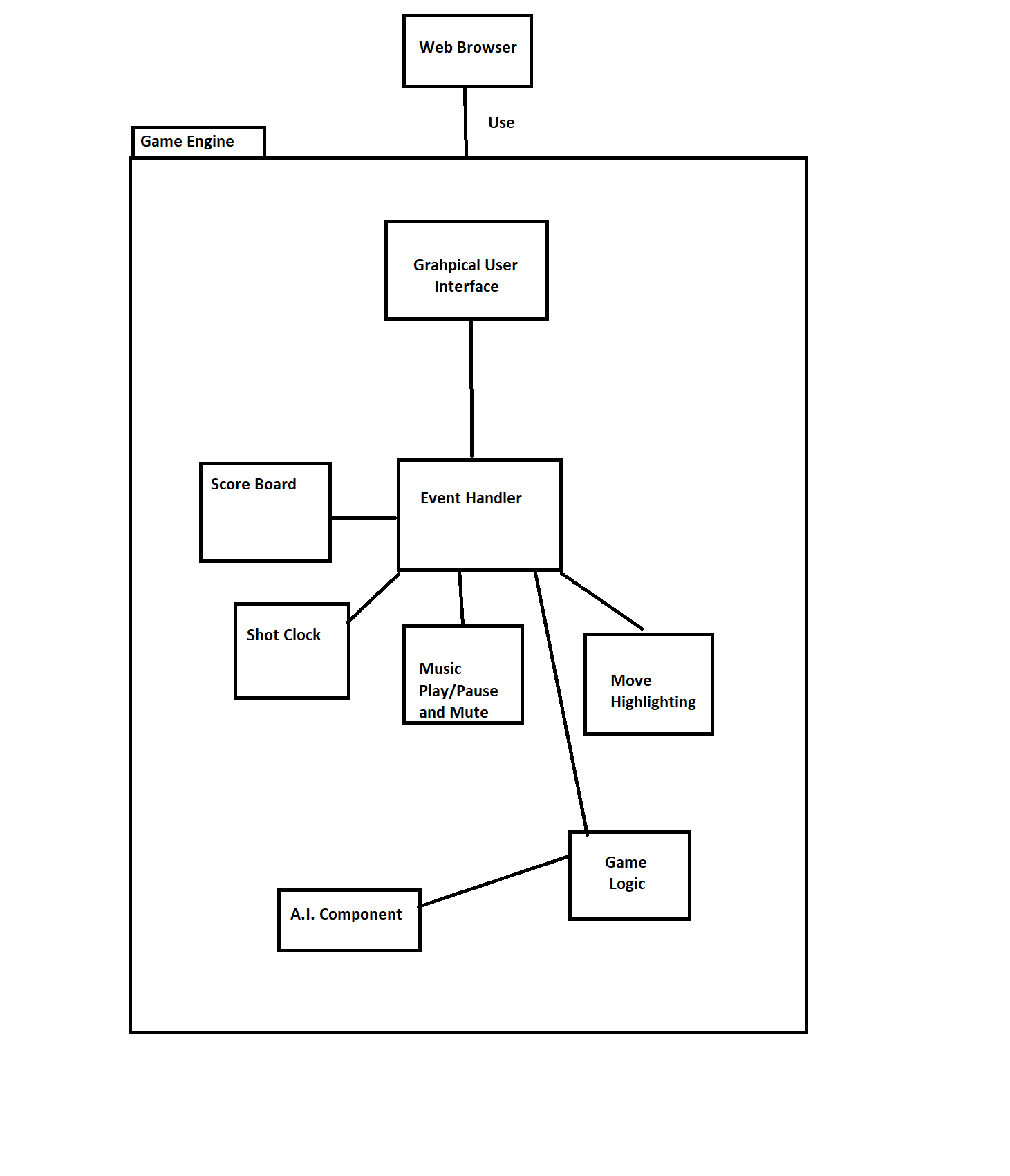
“UmLet.” *UMLetino - Free Online UML Tool for Fast UML Diagrams*, GNU General Public

License, www.umlet.com/umletino/umletino.html.

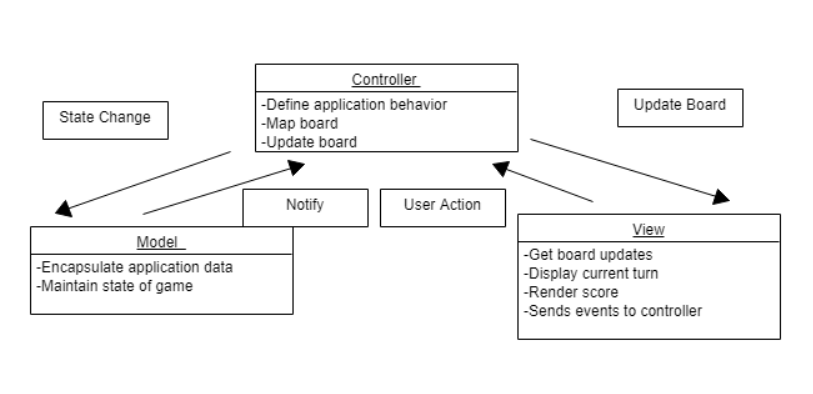
## Document Overview

The purpose of this document is to provide a description of the design implementation for HAC. Section 1 gives the overview of the system structure and functionality at a basic level. Section 2 describes the top-level architectural view of the system and provides the foundation for detailed design work for the main components that will make up the HAC system. Section 3 will present the high-level design view for the sub-components of the system and algorithm descriptions.

# Architecture



# Overview



The above diagram represents an MVC model architecture to represent a simplified design for the HACG. The game logic and rules will be kept in the controller area, along with the functionality to map the board moves and logic to update the board. The view will get updates from the controller and display UI information to the user on whose turn it is, the current score, and send events from the board to the controller so that the board may be kept up-to-date. The model will contain the game data and contain the current state of the game. It will notify the controller of any state changes in the board game.

HAC will use DOM manipulation to draw and update all the different components of the game, including the user manual, scoreboard, shot clock, and the game board itself. DOM manipulation allows the game engine to directly manipulate the elements of the HTML file (representative of the View component of the above model) and update the screen based on the user's inputs.

# UI Component Description

The UI component includes the game board, the user information (scoreboard, shot clock, etc.) below the board as well as the user manual to the right of the game board which come together to make up the game window. There will be a test button at the bottom right of the UI to test the functionality of the checker board. The board will be rendered/updated from Board View, the various user options below the board will be handled by various subcomponents, and the user manual will be handled by the User Manual. The Board View will interact with the Game Engine to get the information needed to update the Board after each turn.

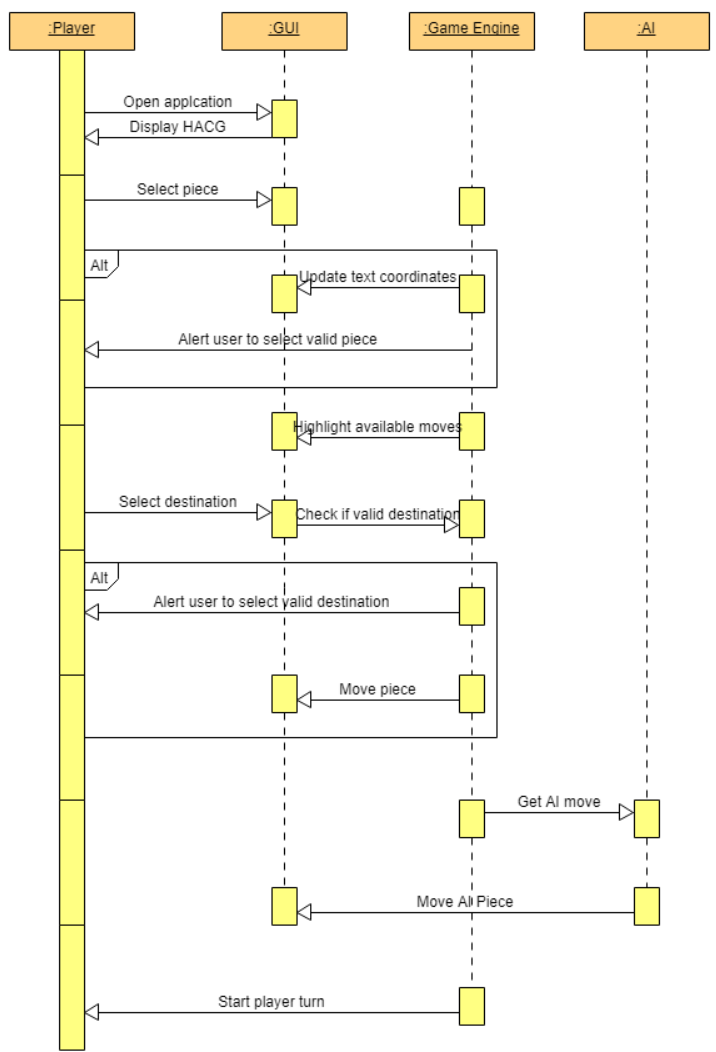
The Game Prompt will interact with the Game Engine to display the current player’s turn and the player’s selected piece. The Score Board will interact with the Game Engine as well to update the score after each piece being taken. The shot clock will interact with the GE to keep an accurate time consumption per player. The music play/pause functionality will interact with the event handlers in that a click will turn on/off the music. The move highlighting component shall highlight optional destinations for a user once a piece is selected. Because it will highlight squares only if a piece is selected, this component will interact with the event handlers.

# Game Engine

The game engine component includes the rules and guidelines for a player’s turn to ensure that the HAC will be a fair and legal checkers game. It will ensure players play for their own game pieces, validate destinations for game pieces, and ensure that players jump over multiple game pieces if possible. The game engine will interact with the board, player and AI to allow for the AI to play with an actual human player.

The game engine acts as both the controller and the model in the MVC model; the game engine takes in and validates the user’s inputs, as well as maintaining the overall state of the game at any given point.

Additionally, the game engine will update a few components of the UI, namely the Board View, the Scoreboard, and the Game Prompt. These components are updated as information in the game changes, such as updating the board view at the end of each turn, and will be detailed in section 3.



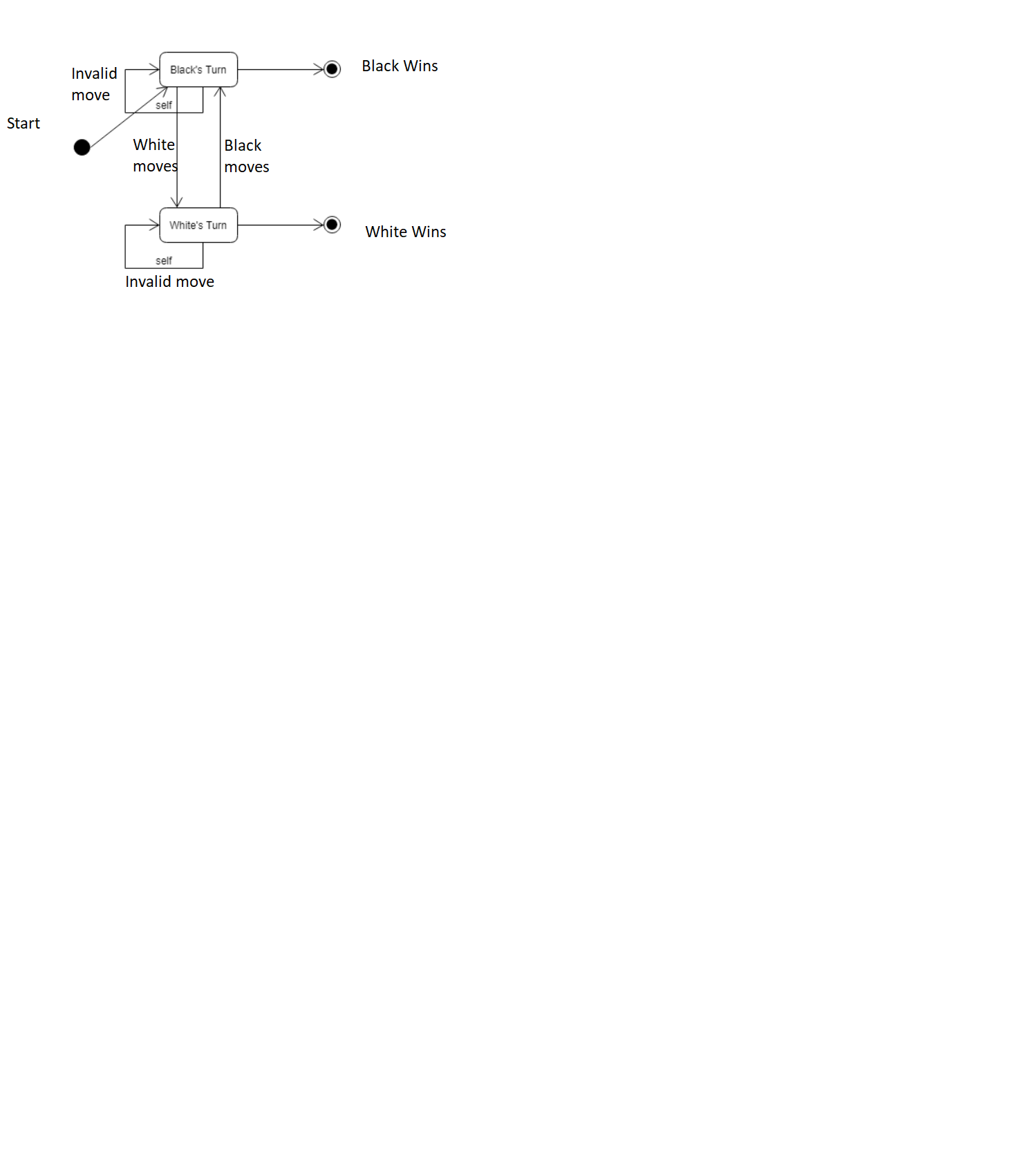
# AI

The AI component will include the same functionality as the player object, except being able to start a new game. Being a player technically, it will have the same functionality as the normal user (player), but it will include an algorithm fit to work with the current implementation of the HACG that will choose a random piece to move to a random valid destination. The AI component will be treated as a player in order to keep the scoreboard up-to-date and keep consistency with the game engine.

# High-Level Design

Below are listed state and sequence diagrams that represent the HACG from a high level.

This diagram represents the UML state machine diagram for the flow of states within the game. The game starts out with black’s turn, as per the traditional checkers guidelines, and will alternate back and forth between the state of white’s turn, versus black’s turn as each player makes a move and progresses the game state. If a player picks an invalid piece or tries to move a piece to an invalid destination, then the state will remain the current player’s turn as they have not made a valid move. When a player makes its last move, and the other player has no more checker pieces to move, then the current player has won the game.



# Game Engine

The user, or player of the game, will be able to select a piece from the UI that they want to move. This will be implemented using an event handler for on click events through the select piece function. When a user selects a piece by clicking on a piece that is valid for their player, then the game engine will update the UI coordinates information to inform the user of the coordinates for the piece they have selected using an id for the selected piece and displaying its id (which will include row and column chosen). If the player selected a piece that is not their own, then the UI will inform the player with an alert that they have chosen an invalid piece and will have the user reselect a piece. Within the select piece function, there will be the additional feature to highlight target destinations for a selected piece. This will be implemented using a hint square function to handle the event of selecting a piece and showing to the user the optional destination squares.

If a valid piece is chosen and a valid target destination square is chosen to move the piece to, then the inner HTML method will be used to swap contents of the squares to move the piece to the proper square. When a piece is moved, the yellow highlighted target squares will be un-highlighted to their original color. If a user selects an invalid destination to move a piece, and it is not one of the yellow highlighted optional move squares, then the player will be alerted that they are trying to move to an invalid square and the squares will be un-highlighted. The player will then have to reselect a piece to move.

# AI

The AI component of the game algorithm will be responsible for moving a game piece after the player has moved their game piece. The AI component will keep track of its pieces and will keep track of when it is its turn to go by calling the getAIMove() method from within the move piece function for the user player. This will ensure that after an actual player has moved their piece, the AI method will get called to move only after a player has completed their turn. After a player has moved their piece, the move piece function will be called to move the AI player piece so that the player can take their next turn next. The AI functionality will be to take a random piece on the board using the Math.random library built in with JavaScript. This function will set a random row and column to choose for the AI piece selection and move the piece after its destination is validated. It will update the scoreboard if needed after completing an AI turn.

# Player

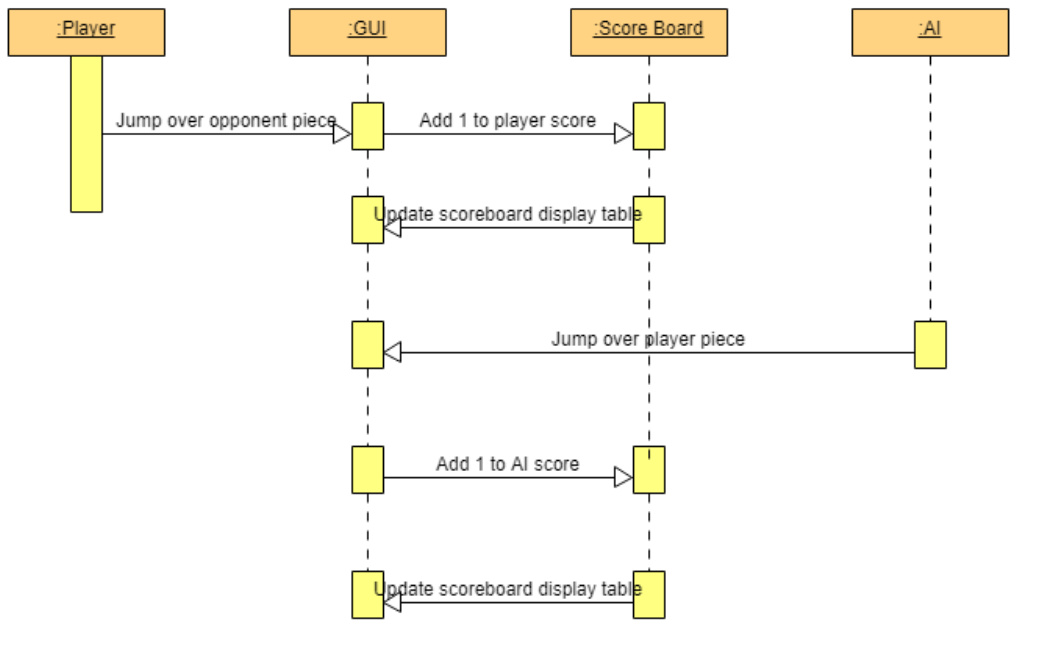
The player component of the game algorithm is responsible for keeping track of which player’s turn it currently is. It will be the entity with which the game algorithm will attribute scores to for the player and AI component. The player functionality will include keeping a variable to maintain the current turn status (currentPlayer), as well as the number of pieces each player has captured (player1Score and player2Score).

# Board View

The board view is a component of the UI, and is responsible for drawing the game board, based on the current game state. It has one main method; the drawWorld() method, which prints the board to the screen based on the positions of the pieces stored in the game engine.

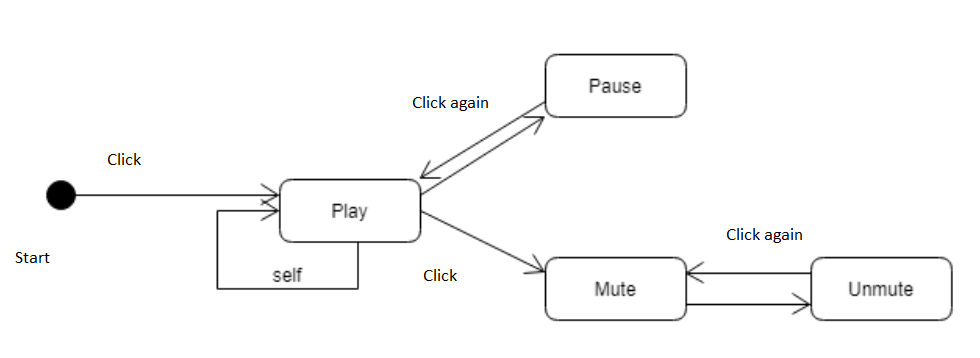
# Score Board

The Score Board is a component of the UI, located underneath the game board and the Game Prompt. It is responsible for showing the score to the user. This score is updated by updating the HTML elements corresponding to the scoreboard, referenced by p1Score and p2Score, and updated using the player1Score and player2Score variables discussed in section (3.3 Player).



# Music Feature

The music toggle component is a component of the UI and is responsible for playing music during the checkers game. It will be an HTML audio element, which embeds a music player into the application. The player may control the toggle to turn the music on/off, change the volume of the music, or mute/unmute the music.

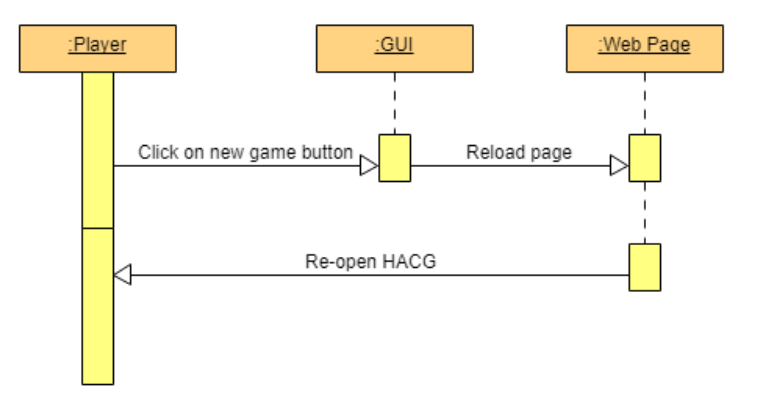


# Game Prompt/ Alerts

The Game Prompt/ Alerts component is a component of the UI and is responsible for showing simple messages to the player. This message will be a string, and it is updated in different various methods. The “Current Players Turn” prompt is updated in the movePiece() method and the “Selected Piece” prompt is updated in the selectPiece() method. The Alert for selecting a non-selectable piece is updated in the selectPiece() method and the Alert for selecting a non-selectable destination is updated in the movePiece() method.

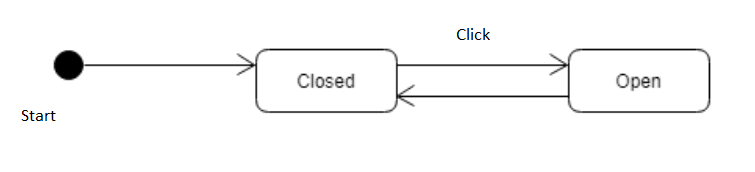
# New Game Button

The New Game button is a component of the UI, located under the game board and the music toggle. Once the button is clicked on, the refreshPage() function will be called, which reloads the page. HAC will refresh and a new game will begin.



# User Manual

The User Manual is a component of the UI, located to the right of the gameboard. Upon the game opening, the User Manual will show the user a closed drop-down button with a set of directions for playing checkers. When the user clicks on the button, the toggle will be expanded, and the user will be shown the directions. The user can then click on the top of the drop-down button to close the drop-down button again. This is done by a small method that gets the contents of the User Manual from the HTML document and shows it on the game board when the User Manual button is expanded.



# Test Button

The test button is a component of the UI, located at the bottom right of the screen. Upon opening the game, the test button may be clicked. This click will then cause the test functionality to be ran. The test function will create a simulated click and assign that click on a hard-coded black-piece on the board. This test function will then send the piece ‘clicked’ through the movePiece function to highlight the optional target destinations and allow the end user to move the piece to a desired destination. The piece will undergo regular game validation.

