1. **Introduction**
   1. Purpose  
      The purpose of this document is to communicate the minimum viable product features, design decisions, roadmap of enhancements, overview of architecture.
2. **Requirements**  
     
   Minimum Viable Product or MVP is a development technique in which a product contains basic features, but enough to add value to a customer. In other words, it is the most basic version of the product that is released.

Below mentioned are the functional requirements implemented for Sprint 1:

* **Player Sign-in**: As a Player I want to sign-in to play online checkers, so that I can play the game.
  + A player can navigate to the Sign-in page from the Home page.
  + A player can enter their name and create a gaming session.
  + The system must reject any sign-in using an existing name.
  + A player can sign-out, which ends that player's session and frees up the name.
  + The system will not persist player information. Furthermore, a given player name is first-come-first-serve. When the player signs-out the name is released for another player to use.
* **Start a Game**: As a Player I want to start a game, so I can challenge other online players.
  + The player can pick an opponent from a list of all players signed-in that are not currently playing a game.
  + This selection launches the Game view that includes a complete 8x8 grid with the initial piece layout.
  + Sends the player back to the Home page if the requested opponent is already engaged in a game.

Below mentioned are the functional requirements implemented for Sprint 2:

* **Player Create Jump**: As a Player I want to create 12 set of coins of a color, so that I can play the web checkers game.
  + Coins move diagonally in forward direction on the same colored block with opponent disc in the next position in the diagonal forward position
  + Moves multiple squares in the forward direction diagonally when there is an empty diagonal square
  + Moves only in forward direction diagonally (except the king) refer the user story player king identification
  + When the disc reaches the farthest position with respect to the player , it becomes the king
  + Eliminate only one opponent disc even when multiple discs are jumped over
* **Player Disc Movement**: As a Player I want to move my discs forward so that I can progress in the game.
  + Coins move diagonally in forward direction on the same colored block on which it is placed on start of the game.
  + Moves only one square in the forward direction diagonally when there is an empty diagonal square
  + Moves only in forward direction diagonally (except the king) refer the user story player king identification
  + When the disc reaches the farthest position with respect to the player, it becomes the king
  + Coins that do not move diagonally in forward direction on the same colored block when there is an empty diagonal square should not be allowed
* **Player King Identification**: As a Player I want to add a single coin upon my first farthest coin so that I can identify my king.
  + Player disc has reached the farthest end of the opponent side
* **King Jump**: As a King I want to Jump over opponent discs so that I can eliminate the opponent disc
  + The king coin should be distinct
  + The king should be able to move in any direction
  + The king can conquer coins
* **Player Resignation**: As a PLAYER I want to leave the game so that I can stop my game
  + The player should be able to resign from a game
  + The player should be able to play with another player who is signed in the system
  + The player should be able to leave his game in the middle of a game
  + Notification will be in the form of a message box where is will also display the opponent has won the game
  + The player should 'submit turn' and resign the game
  + The player will be notified when his opponent resigns when he makes his move.
* **Player Game Win:** As a Player I want to conclude the game so that I win the game
  + The 11 opponent pieces are captured
  + The system will direct the player to 'win game' page
* **Player sign out:** As a PLAYER I want to sign out so that I can exit Web Checkers game
  + The player has to resign the game and then he/she will be able to sign out
  + After sign out, the system shows the login page

1. **Domain Model**

A close up of a logo

Description generated with very high confidence

1. Class Diagram  
     
   A close up of a piece of paper

   Description generated with high confidence
2. Application Architecture

The web checkers application contains three tires namely: Model, UI and Application. Below mentioned are the classes contained in each layer:

Model:

* Board
* Message
* Move
* Piece
* Player
* Position
* Row
* Space
* WebCheckersGame

UI:

* GameController
* HomeController
* JsonUtils
* PostCheckTurnController
* PostSubmitTurnController
* PostValidateMoveController
* WebServer

Application:

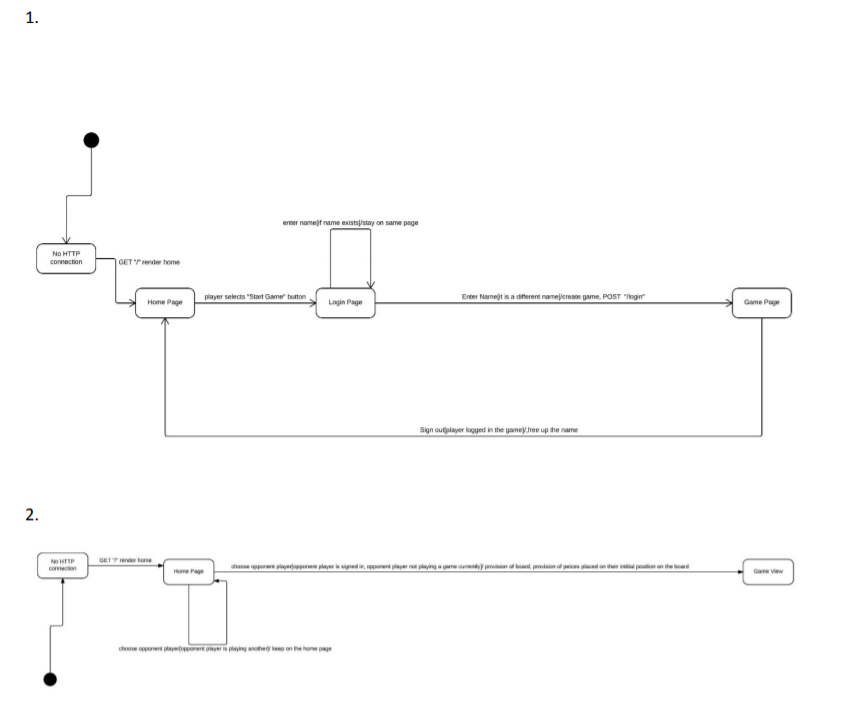
* GameCenter

Adherence to design principles:

The design principles of SOLID are being implemented in this code. We have separated the concerns of each object. Each component has a different class so that they are independent, and we can anytime add more functionalities to them as we will extend the project. Low coupling is performed with no dependency of any module or any method on some other methods. However, the elements are tightly cohesive to all its related elements. A high degree of cohesion and low degree of coupling is achieved in the code.

The principle of “DRY” (Don’t Repeat Yourself) is used in the code. Code re-usability is ensured in the project.

1. State-based behavior



1. Sequence Diagram

Start a game

A close up of text on a white surface

Description generated with high confidence

1. Code Coverage Analysis

This is our analysis at the top level of the project for Sprint 2.

## 

The model, UI anad application tier’s code has been well tested and the webCheckers package has a lot of code that has not been tested since there it needs to mocked up by creating mock classes.

# UI Tier

This is our analysis at the UI Tier code for the project.

## 

The UI tier classes has been tested and the above screenshot shows the result of the Junit test coverage

# Application Tier

This is our analysis at the UI Tier code for the project.

## 

The classes in this tier that have been tested are GameCenter and PairWithPlayers game and their Junit test code coverage is mentioned in the above screenshot.

# Model Tier

This is our analysis at the UI Tier code for the project.

## 

This tier consists of a lot of Junit test cases for the classes present and the above screenshot shows the Junit code coverage.

1. Complexity Metrics  
     
   WMC Metrics: WMC stands for Weighted Methods Per Class and it is simply the method count for a class.

WMC = number of methods defined in class

It is a good practice to keep WMC down. A high WMC has been found to lead to more faults. Classes with many methods are likely to be more more application specific, limiting the possibility of reuse. WMC is a predictor of how much time and effort is required to develop and maintain the class. A large number of methods also means a greater potential impact on derived classes, since the derived classes inherit (some of) the methods of the base class. Search for high WMC values to spot classes that could be restructured into several smaller classes.

What is good WMC? Different limits have been defined. One way is to limit the number of methods in a class to, say, 20 or 50. Another way is to specify that a maximum of 10% of classes can have more than 24 methods. This allows large classes but most classes should be small.

