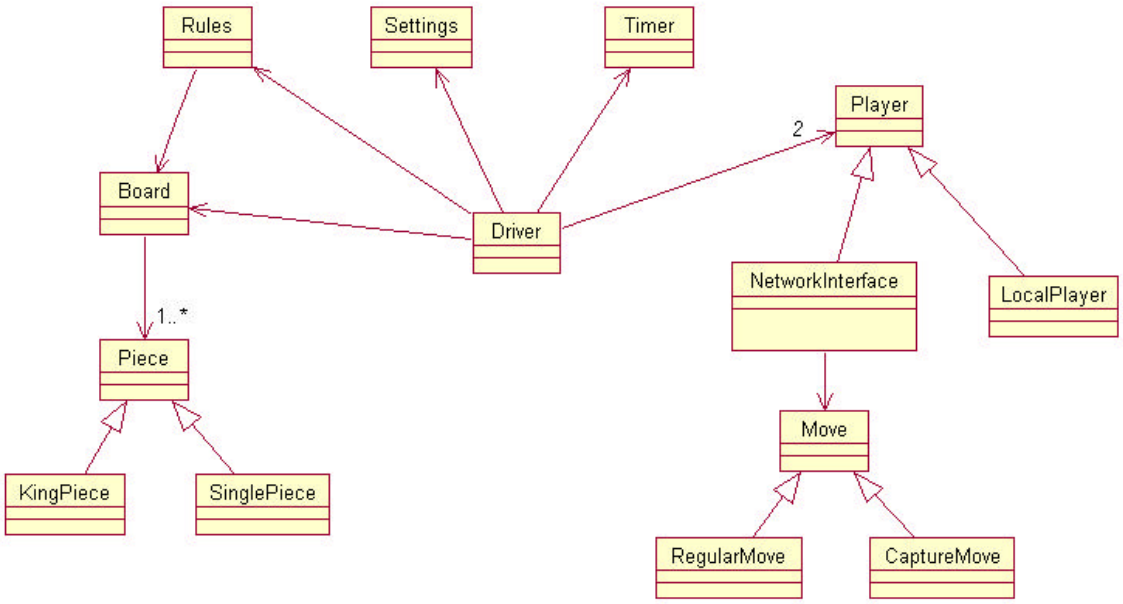
Team Name/Project Name	
Component	Requirements
Document	Analysis Class Diagram
Responsible	Support Leader
Version:1.0	**Date:**

Analysis Class diagram



Class Descriptions:

Board:

- Holds the pieces
- Keeps track of game piece locations
- Move piece from one square to the other
- Provides the information to the Driver class
- The Rules may need to access the board

Piece:

- Keeps track of game piece information
- It is an abstract class
- It is a representation of checkers piece on the board

KingPiece:

- Keeps track of kings piece information
- Extends the Piece class
- Has more capability then a single

SinglePiece:

- Keeps track of single piece information
- Extends the Piece class
- Has less capability than a king

Rules:

- Keeps track of the rules
- Verifies the moves
- Knows game ending conditions
- Provides the rules to the Driver class
- Sometimes provides the rules to the Board class

Player:

- Waits on user input
- Relays that input to Driver
- Takes input from GUI or network Interface
- It is an abstract class
- It is the middle man between the GUI and the Driver class

Driver:

- Talks to player
- Keeps track of turns
- Keeps track of the settings
- Keeps track of time if needed
- It communicates with the board
- It is the back bone of the program
- It provides for the flow of executions

Network Interface:

- Sends moves over network
- Receives moves over the network
- Disconnects if the game ends
- Detects if there is a connection to the other player or not
- Inherits from the Player class

Move:

- Holds the move information for transmission over network
- It is an abstract class
- It is sent by the network interface to the other computer

RegularMove:

- Sends a regular move over the network
- It will include what kind of piece it is and start and end locations
- It inherits from the Move class

CaptureMove:

- Sends a capture move over the network
- It will include the piece that is jumping and the start and end locations
- It inherits from the move class

LocalPlayer:

- It is same the network interface
- But it is a local interface for the player
- It inherits from the Player class

Settings:

- Holds the options information
- Provides this information to a Driver class

Timer:

- Holds the timer information
- The timer is used to tell when the player's turn is over
- Provides the information to the Driver class

Rationale

We tried to use abstraction whenever possible to make maintenance and future reference easier, this however will require more memory during execution. We made separate classes for Settings and Rules in order to allow for easier modification in the future, this however also generates a larger program footprint during execution since more objects will need to be instantiated. This program was written with maintenance, human readability and human understanding in mind more so than strict efficiency.

We chose to hide the fact that a game may be networked as much as possible from the core of the game (board, rules, settings, and the driver) in order to make the program less complicated. Fewer contingencies need to be taken care of in the kernel by abstracting where the users are by use of the Player class.

Our program generates most of the needed information on-demand instead of storing it in the various objects of our program. We do this in an effort to decrease the overall size of the program, needed objects/information will be passed to objects only when they need it for processing.

In splitting the rules from the board, we made a design decision to make the board responsible for maintaining only the locations of game pieces and available squares, while the rules class will hold the “rules of the game.” We also added the Player class in order to abstract the location of the users, and maintain some of the data that would have been stored in the driver in order to eliminate complexities in the driver class.