Checkers Game Engine

Design Documentation

Prepared by Brendan Grau

Last updated: 2020-10-15

Summary

The Checkers Game Engine(CGE) is a system for playing checkers against an intelligent AI opponent implemented using the algorithm described by K. Chellapilla and D.B. Fogel's paper titled *Evolving Neural Networks to Play Checkers Without Relying on Expert Knowledge*. The system will be a playable checkers game consisting of both command line and graphical interfaces and will store games for later review.

The system is implemented as a C++ application using the [INSERT GRAPHICS LIBRARY] as the graphics library and defaults to this graphical interface. The system is structured using the model view controller design pattern.

Requirements

V1 Functional Requirements:

1. The CGE shall run checkers games according to the American Checkers Federation rules
    - A. Players shall be notified and prevented from violating a game rule
2. The CGE shall implement player vs player, player vs computer, and computer vs computer modes
    - A. Mode shall be selectable at the start of a game
3. The CGE shall have a command line and graphical interface
    - A. Players shall be notified when it is their turn
    - B. The CLI shall receive moves as a space separated pair of numbers representing the start and end position of a piece respectively.
        - i. Input shall be accepted one move at a time. Additional moves shall be ignored
        - ii. Input will be checked for errors and the users shall be asked to reenter input if check fails
            - I. Errors can be from invalid format or invalid moves
            - II. The CLI shall be redrawn before asking for a corrected move
    - C. The CLI shall display the game board when asking for move
4. The CGE shall store previous completed games in standard Draughts notation
    - A. The game shall be notated to include tha player name, date, and result
5. The CGE shall implement an ai to play against
    - A. The ai shall be a neural network
    - B. The ai shall be trainable

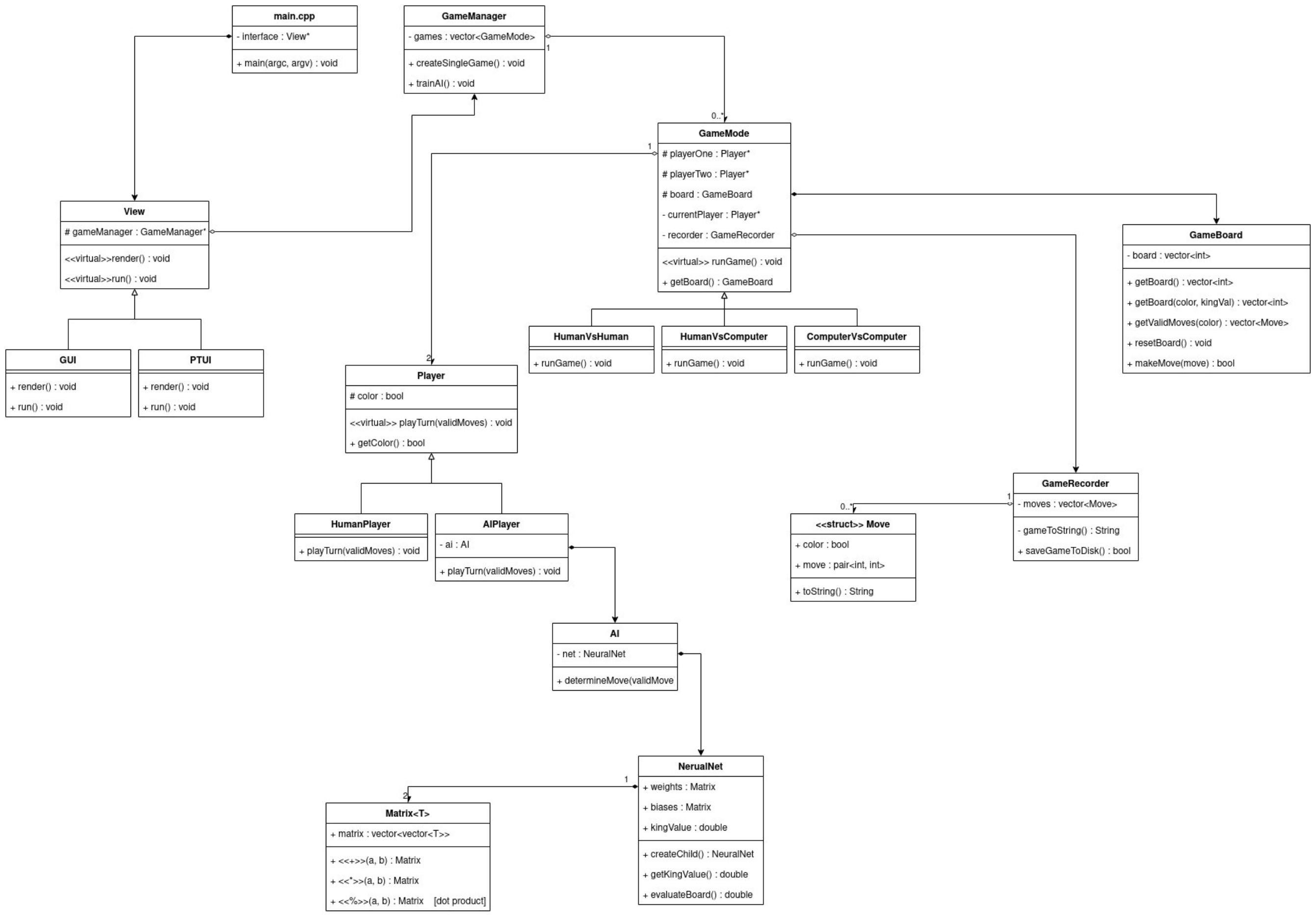
V1 Non Functional Requirements:

1. The system shall store the board as a size 32 array
2. The system shall be implemented in C++ as a desktop application
3. The system shall store ai weights and biases in a plain text file
4. The system shall store games as portable draughts notation(.PDN) files using the 3.0 standard
5. The system shall train the ai using the fitness function genetic algorithm
    - A. To speed up training, threading shall be used to train the system

Potential Enhancements

1. AI analysis of previous games
2. Live AI analysis of current game
3. AI difficulty levels
4. View saved games

Class Diagram:



Subsystems

This section provides detailed design for various subsystems

AI

The ai subsystem is the brains of computer player and determines the best move for the computer to make. Moves are determined by using a min-max tree with alpha-beta pruning and other optimizations. The neural network is responsible for determining the value of each node.

Name: AI	
Participants	
Class	**Participant's contribution in the context of the application**
AI	This is the main controller of the system and determines the move to be made using a min-max tree. It also handles creating its own children AIs as part of training
GameManager	Responsible for training AIs.
NeuralNetwork	Determines the value of a node in the min-max tree
Matrix	Matrix datatype used for representing the neural network
Requirements being covered: 5	

View

The view subsystem contains the visual components as well as controllers for the system. Graphics are selectable as either GUI or PTUI.

Name: View	
Participants	
Class	**Participant's contribution in the context of the application**
View	Abstract class for GUI and PTUI classes
PTUI	Handles command line input and output
GUI	Handles graphical input and output
Requirements being covered: 3	

Game Recorder

This subsystem is responsible for recording games and saving them to disk.

Name: Game Recorder	
Participants	
Class	**Participant's contribution in the context of the application**
GameMode	Interface for classes used to pass moves to the game recorder
GameRecorder	Records moves and allows game to be saved to disk
Move	Datatype representing a single move
Requirements being covered: 4	

Game

This subsystem is responsible for running a game of checkers

Name: Game Recorder	
Participants	
Class	**Participant's contribution in the context of the application**
GameMode	Interface for classes used play various types of checkers games
HumanVsHuman	Derived class for playing a human vs human game
HumanVsComputer	Derived class for playing a human vs computer game
ComputerVsComputer	Derived class for playing a computer vs computer game
GameBoard	Responsible for managing the board and validating moves
Player	Abstract class used to define a player
HumanPlayer	Derived class which implements human behavior for playing a turn
AIPlayer	Derived class which implements AI behavior for playing a turn
Requirements being covered: 1, 2	

Status of the Implementation

Provide a complete description of the status of your implementation. This should specify all known defects in the system, and indicate requirements that your implementation does not cover.

Appendix

This section provides fine-grained design details for all of the classes in your design. You will capture this information using the CRC (Class-Responsibilities-Collaborators) card format below.

Class: `MyClass1`	
Responsibilities: ...	
Collaborators: ...	
Users: ...	**Used by:** ...
Author: ...	