Software Requirements Specification
for
e.DO Robot

Adrian Ionascu
Eric Laberge
Trevor Kretschmann
Arhum Ahmed

February 2019

Contents

1 Introduction

1.1 Purpose

The purpose of this Software Requirements Document (SRS) is to outline the features and capabilities the e.Do Robot Checkers AI is expected to have. This SRS was designed for all stakeholders involved in this project, including the software engineering team as well as our client Comau. This document contains a description of our project, the various interfaces, the core requirements, and additional constraints and features. The intent of the SRS is to create a contract of expectations and parameters, so there is no ambiguity between the client and the software engineering team about the capabilities of the desired, final product.

1.2 Scope

The e.Do Checkers AI is constructed from three parts. The first part is the image detection API which uses OpenCV to process frames from the camera pointing down at the checkerboard. It extracts squares and circles from the image and relays that information to the AI and the manipulation code to give the exact board state and piece locations. The second part is the actual AI which analyzes the board state and generates the best possible move based on the Minimax algorithm. The heuristic that determines the favorability of each state is supplemented with a machine learning algorithm that modifies and improves the heuristic based on historical data from previous plays. The third part is the manipulation code that makes calls to ROS (Robot Operating System) on the Raspberry Pi powering the e.Do robot to direct the motors to physically move the checkers pieces around the board. The robot will be able to play checkers against an opponent reasonably well and demonstrate its ability to improve with additional trials. The purpose of the AI is for the client to demo the robot playing checkers to children to interest them in robotics and illustrate the possibilities of thoughtful, creative software.

1.3 Definitions, Acronyms, and Abbreviations

AI: Artificial Intelligence. Algorithms that make strategic decisions based on current situation.
API: Application Programming Interface (API) is a set of functions and code that allows for different pieces of software to communicate with each other.
Cartesian: A coordinate system based on 3 planes of space that give signed distances from each other.
Heuristic: A strategy to solve an abstract solution (such as a playing effectively at checkers).
Machine Learning: The capacity for the AI to improve based on analysis of data collected from additional trials.
Minimax: The Minimax algorithm is commonly used in game theory and AI. The algorithm computes the best move to make by minimizing the maximum loss allowed from all possible paths at any given state. **OpenCV**: Open Source Computer Vision Library (OpenCV) is a

library containing several functions that assist with object detection.
ROS: Robot Operating System (ROS) is a collection of libraries and tools that provide abstraction for many low level functions relating to robotic movement and processing.

1.4 Overview

The rest of the SRS contains a description section, a requirements section, and an analysis models section. The description section contains general information about the project such as the structure and basic functionality. In addition, design constraints and dependencies are explained here. The requirements section contains more technical details, such as the interfaces and specific functional and non-functional requirements. Finally, the analysis models section gives diagrams to elaborate on the software's intercommunication and scope.

2 General Description

2.1 Product Perspective

This system will consist of a single script that will call multiple APIs to function. The system will be completely self contained and be able to fully function even with no internet connection. As this will be self contained, there will be no database. The script will call APIs that will connect to the e.DO robot for arm manipulation and image detection. The script will also call an AI API.

2.2 Product Functions

The sole function of this software is to play checkers at a reasonable level. To be able to properly do this, the software must be able to accurately depict the board with image detection software, accurately move pieces from one spot on the board to another, and decide which move will most likely lead it to a victory. At behest of the client, further implementation will include allowing the AI to learn from previous games to further optimize its checkers playing via machine learning.

2.3 User Characteristics

The main users for this software will be high school students that are showed the e.DO robot for educational purposes. Other users will include possible customers looking to purchase the e.DO robot. Since all the users will not be well versed in checkers to begin with, the AI does not need to be optimal to defeat its opponent. The client have specified the desire for machine learning, however the main focus will be on image detection and robot manipulation. As the purpose of the program is to amaze students and prospective customers as well as demonstrate the e.DO robot capabilities, the image detection and robot manipulation running smoothly is crucial.

2.4 Design Constraints

The primary design constraint for this project is the physical hardware limitations of the e.DO robot and the accompanying camera. Both pieces are stationary and already in place which limits how we can do image detection and the reach of the e.DO robot for moving pieces.

2.5 Dependencies and Assumptions

The primary assumption for this software is the e.DO robot and the accompanying camera are functioning properly. If there is any issues with either hardware, the software will cease to function. The software must also assume proper lighting for the camera and placement of the checkerboard/checker pieces. If the lighting is inadequate then the camera will be unable to detect the squares on the board or the pieces properly. Similarly, if the pieces or board are not completely within range of the camera it will fail to make a proper representation of the board for the AI.

3 System Features

3.1 External Interface Requirements

This section provides a detailed description of all inputs into and outputs from the system. It also gives a description of the hardware, software and communication interfaces and provides basic prototypes of the user interface.

3.1.1 User Interface

The only user interface will be a terminal output that constantly displays information about the status of the game. For every move, the program will print out information such as the computed move the AI wants to perform, whose turn it is, and any error it encounters with the AI or object detection portions of the program.

3.1.2 Hardware Interfaces

The hardware being used is a robot made by Comau named "e.DO". The e.DO robot is a 6-axis arm equipped with a gripper and powered by a raspberry pi. The robot works with a camera a fixed distance away from a board for problems involving image recognition and object detection.

3.1.3 Software Interfaces

The robot is controlled through a robotics middleware called the "Robot Operating System" (ROS). ROS provides many of the functionalities of an operating system, allowing for abstraction of low-level control of the robot. The robot comes equipped with many "topics", which are channels for instructions/data, and packages that allow one to write programs to "publish" and "subscribe" to those topics. The topics that move the physical robot are '/bridge_move' and '/bridge_jog'. The topic '/cartesian_pose' gives the robot's current position in 3D space.

The project will also be using OpenCV for object detection purposes. Using OpenCV, camera video feed will be scanned and used to determine information such as the board state

and checker piece location. This information will then be sent to the AI being implemented to determine moves to be made.

The Checkers AI being implemented is an open source project found on Github. It was designed as a standalone program that included a GUI for a user to drag and drop their moves on the virtual checkers board. For our project, we removed the GUI and rewrote certain elements so that the board states being processed would rely on an data coming from OpenCV rather than relying on internal tracking. The AI accepts an array containing all the checker piece locations and attributes as an input. Using the Minimax algorithm, the AI computes the best possible move given the current board state. This is fed back to ROS to execute the move.

3.1.4 Communications Interfaces

The robot can be sent messages over Wi-Fi, ethernet, or through a serial port located on the base of the robot. The only way to control the robot over these methods is to send ROS messages to the robot, direct hardware-level access is not permitted. The camera data is not sent to the robot directly, but rather the programs sending instructions to the robot.

3.2 Functional Requirements

3.2.1 Object Detection

ID: OFR1
Title: Checker Detection
Requirement: Program will detect all checker pieces.
Description: The program will look at the camera video feed and detect checker pieces.
Forward Dependencies: OFR3, OFR4, MFR1
Backward Dependencies:

ID: OFR2
Title: Checker Squares Detection
Requirement: Program will detect all playable checker squares on the board.
Description: The program will look at the camera video feed and detect all playable checker squares and their location in the image.
Forward Dependencies: OFR3, MFR2
Backward Dependencies:

ID: OFR3
Title: Checker Placement Detection
Requirement: Program must detect all checker pieces and which square they are in.
Description: The program must be able to look at camera video feed and detect all checker pieces and determine their location on the board.
Forward Dependencies:
Backward Dependencies: OFR1, OFR2

ID: OFR4
Title: Detect opponent moves
Requirement: Program must detect when the opponent has made their move.
Description: The program must be able to look at camera video feed and determine if the opponent has made there move yet or not.
Forward Dependencies:
Backward Dependencies: OFR1, OFR2, OFR3

ID: OFR5
Title: Detect current move
Requirement: Program must detect when the opponent is in the middle of their move.
Description: The program must be able to look at camera video feed and determine if the opponent is in the middle of a move. It will then delay its next move until the opponent is finished making their current move.
Forward Dependencies:
Backward Dependencies: OFR2

ID: OFR6
Title: Convert board state to a matrix
Requirement: Program must be able to convert current board state to matrix.
Description: The program must look at the board state through the camera video feed, and output the board state as a matrix or logical array or AI use or output.
Forward Dependencies:
Backward Dependencies: OFR1, OFR2, OFR3

ID: OFR7
Title: Detecting Piece Color
Requirement: Program can differentiate between different color pieces.
Description: The program will look at the checkers board and be able to differeniate the different color pieces that represent pawns and kings for both teams.
Forward Dependencies:
Backward Dependencies: OFR1

3.2.2 Artificial Intelligence

ID: AFR1
Title: Possible Moves
Requirement: The checkers AI must determine what possible moves it can make.
Description: The checkers AI must be able to determine all possible moves based on the current board state.
Forward Dependencies: AFR2
Backward Dependencies:

ID: AFR2
Title: Future Moves
Requirement: The checkers AI must determine all possible board states n moves ahead.
Description: The checkers AI must be able to determine all possible board states, given the current board state, that the game could result in n moves ahead.
Forward Dependencies:
Backward Dependencies: AFR1

ID: AFR3
Title: Invalid Movement
Requirement: The checkers AI must determine if a player move is invalid.
Description: The checkers AI must determine if the board state following a player move was the result of a legal checkers move.
Forward Dependencies:
Backward Dependencies: OFR4

ID: AFR4
Title: Checkers Heuristic
Requirement: The checkers AI must determine the quality of each board state.
Description: The checkers AI must be able to use a heuristic to determine how favorable any given board state is for the AI .
Forward Dependencies:
Backward Dependencies:

ID: AFR5
Title: AI Movement
Requirement: The checkers AI must determine the moves the robot makes.
Description: The checkers AI must be able to use the heuristic with all possible board states and use a Min Max Tree to determine the best move.
Forward Dependencies:
Backward Dependencies: AFR2, AFR4

3.2.3 Manipulation

ID: MFR1
Title: Pick Up Pieces
Requirement: The Robot arm must be able to pick up checkers pieces.
Description: When the e.DO Robot is told to pick up a piece on a given spot, it must properly pick up the piece on that spot.
Forward Dependencies: MFR2
Backward Dependencies: OFR1

ID: MFR2
Title: Place Pieces
Requirement: The Robot arm must be able to accurately place checkers pieces.
Description: When the e.DO Robot is told to place a piece on a given spot, it must properly place the piece on that spot.
Forward Dependencies:
Backward Dependencies: OFR2, MFR1

3.2.4 User Interface Requirements

ID: UR1
Title: Printing Board
Requirement: The program will print a visual image representing the state of the checkers board.
Description: The program will print an image representing the state of the checker board as it has interpreted it.
Forward Dependencies:
Backward Dependencies: OFR6

ID: UR2
Title: Print Turn
Requirement: The program will print the current turn.
Description: When the program detects a change in turn, it will print the current turn that is being taken.
Forward Dependencies:
Backward Dependencies: OFR4, OFR5, OFR8

ID: UR3
Title: Print Invalid Move
Requirement: The program will print a message when it detects an invalid move.
Description: If the player makes an illegal checkers move, the program will output a message stating the move was not valid and will wait until the user corrects the move.
Forward Dependencies:
Backward Dependencies: OFR4, OFR5, OFR8

3.3 Non-Functional Requirements

3.3.1 Performance

ID: NFR1
Title: Turn Detection Performance
Requirement: The program will respond to a human move within 20 seconds.
Description: The program will respond accordingly to the changed state of the board following a human move within 20 seconds of its completion.

ID: NFR2
Title: Pick up Performance
Requirement: The program will correctly pick up pieces 95 percent of the time.
Description: The program will correctly pick up a checkers piece after being told to but the AI at least 95 percent of the time.

ID: NFR3
Title: Placement Performance
Requirement: The program will correctly place pieces 95 percent of the time.
Description: The program will correctly place a checkers piece after being told to but the AI at least 95 percent of the time.

3.3.2 Reliability

ID: NFR4
Title: Reliability
Requirement: The program will not crash during more than 10% of games.
Description: The program will be rigid, crashing during, at most, 10% of games.

3.3.3 Data Integrity

ID: NFR5
Title: Data Integrity
Requirement: The program will assure it has the correct status of the game board.
Description: The program, using image recognition, will assure it reads the state of the board correctly.

3.4 Design Constraints

3.4.1 Hardware Constraints

ID: HC1
Title: Hardware Limitation
Constraint: The program will only be developed and run on the Comau e.DO Robot Arm and uses the Logitech c920 webcam.

3.4.2 Software Constraints

ID: SC1
Title: Control Limitation
Constraint: The program will control the robot through the Robot Operating System (ROS).

ID: SC2
Title: Language Limitation
Constraint: The program will control the robot through the rospy packages for Python.

4 Analysis Models

4.1 Data Flow Diagrams