Software Engineering

Final Submission

Group 3

15th of May, 2015

UNIVERSITÄT MANNHEIM

Table of Contents

Preface

As a part of the lecture “Praktikum Software Engineering” in spring 2015 the students develop a new system, which interacts with a humanoid robot called “NAO” who plays Nine Men's Morris.

The team constellation of group 3 consists of 6 members:

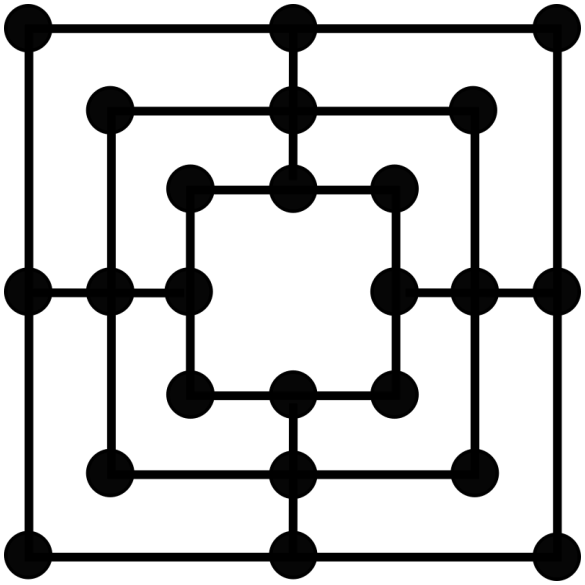
#	Last Name	First Name	Matriculation Number
1	Betz	Julian	14 159 36
2	Lu	Wei Hao	14 055 56
3	Narr	Aljoscha	14 120 06
4	Siefert	Mike	14 047 02
5	Thietke	Jonas	14 292 72
6	Weber	Nick	14 050 12

Introduction

The humanoid robot, called “NAO”, should be able to play Nine Men’s Morris against a human or another robot according to the following game rules. NAO is constrained by those rules and at the same time challenged by his opponent. Therefore the robot needs an artificial intelligence to control the gaming flow and additional motoric skills to move pieces, also called “men”.

“Nine Men’s Morris” - Gaming rules:

The board consists of a grid with twenty-four intersections or points. Each player has nine pieces, or "men", usually colored black and white. Players try to form 'mills' - three of their own men lined horizontally or vertically - allowing a player to remove an opponent's man from the game. A player wins by reducing the opponent to two pieces (where he could no longer form mills and thus be unable to win), or by leaving him without a legal move.



The game proceeds in three phases:

1. *Placing men on vacant points*
2. *Moving men to adjacent points*
3. *(Optional phase) Moving men to any vacant point when a player has been reduced to three men*[1]

[1] http://en.wikipedia.org/wiki/Nine_Men%27s_Morris, retrieved 2015-02-20

System Architecture

The system architecture will be structured in six main modules.

User Interface: ui.view

The first module is the User Interface (UI), which will be realized as a graphical Java Swing application using Java AWT elements as well. It will contain a button section, which enables the user to change the camera view, communicate with NAO, pause the game and terminate the actions. There also is a graphical representation of the current state of the board and information about the current number of men on the board for each color. Furthermore, there will be a clock counting from the beginning of the game and a time counter informing about the time already used to perform the current move. In situations that require human feedback, dialog windows will provide the option to directly interact with the system.

Central Control: application.control

The main class launches the system while the controller is in charge of providing the connection between NAO, AI and GUI.

Artificial Intelligence: application.ai

The Artificial Intelligence (AI) will calculate optimal moves within the constraints of the logical unit and contains the path-finding engine as well.

Game Logics: domain.represent

There will be a logical unit, which contains all of the rules of Nine Men's Morris and a virtual representation of the board. It will be operating in two states, which represent the two mandatory phases mentioned at the end of the introduction. The last – optional – phase will be left out.

Robotics: domain.nao

Enables the robot to execute a predetermined move in a physical or simulated environment. In order to do this the robot needs the ability to move within the environment correctly and as well recognize certain objects. The combination of both makes it possible for the robot to navigate to a specific point of the play board, grab a gaming piece and place it on a given field.

Data Structures: foundation.data

Provides data structures used in several other modules. The main focus is on efficient exchange of data between subsystems.

Frameworks

Framework	Description
Java	Java as an overall programming language contains several useful libraries in order to meet all the requirements. Version 1.8 was used.
Swing API	The Swing framework provides an interface to create a graphical user interface (GUI).
JUnit	JUnit is a framework helping to build test classes according to the requirements defined below helping to validate the systems behavior throughout the development stages.
NAOQI Java API	The framework provides several communication interfaces for the NAO robot, including motion, audio, video and sharing functionalities.
Apache Commons Validator	The Apache Commons Validator library contains useful classes and methods to validate different sets of data. Appache Commons Validator Source
Sigar API	The Sigar API provides a portable interface for gathering system information such as system memory, CPU, etc. Sigar API Source

System Requirements Specification

Functional Requirements

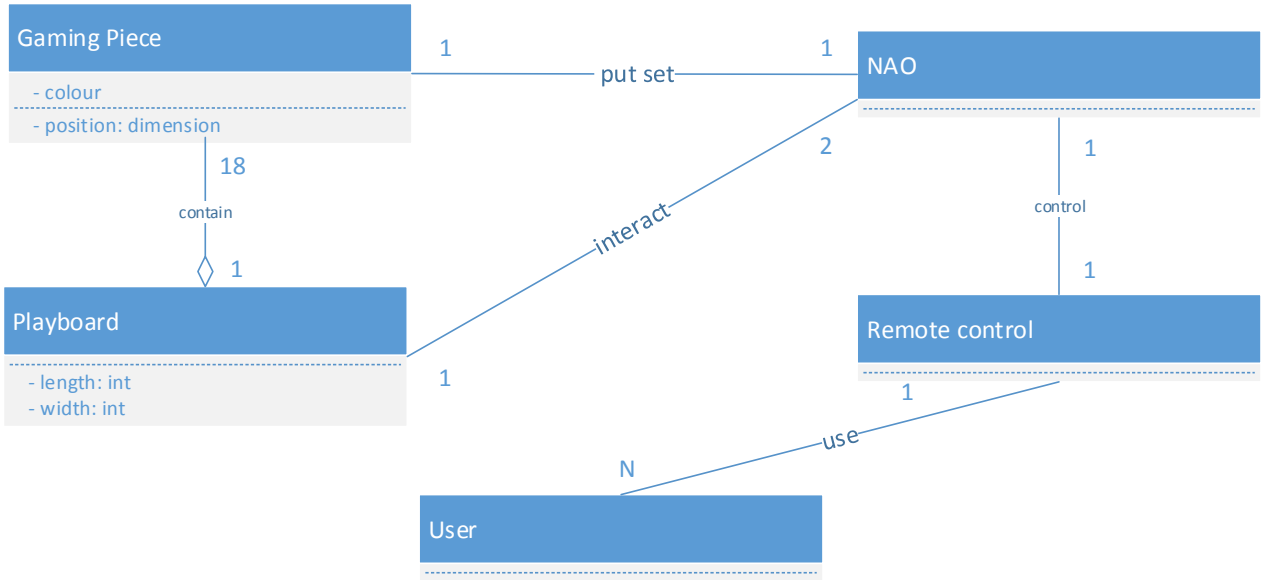
IDs	Description
FREQ1.	The robot shall be able to recognize positions on a Nine Men's Morris board
FREQ2.	The robot shall be able to reach every field from any colored circle on the play board
FREQ3.	A user shall be able to set the color of the robot's men
FREQ4.	The robot shall recognize the men of both colors
FREQ5.	The robot shall detect the men's position on the board
FREQ6.	The system shall generally distinguish between its men and opponents men by color
FREQ7.	The system shall be able to do a virtual representation of the actual state of the board
FREQ8.	The AI of the robot shall be able to plan a optimal next move on basis of the virtual representation of the board
FREQ9.	The robot shall be able to grab a man with the help of the user
FREQ10.	The robot shall be able to walk while holding a man
FREQ11.	The robot shall be able to set down a man with help of the user
FREQ12.	The robot shall be able to combine requirements FREQ9 to FREQ11 in order to relocate a man
FREQ13.	The robot shall leave the area in immediate range of the play board's fields as soon as its move is done
FREQ14.	The AI of the robot shall behave according to the rules of Nine Men's Morris
FREQ15.	There shall be a user interface (UI)
FREQ16.	The UI shall present the robot's camera views
FREQ17.	The UI shall present a visual representation of the board, containing all the men in play at their positions
FREQ18.	The UI shall present the current game status
FREQ19.	The user shall be able to start the next move of NAO via the UI
FREQ20.	The user shall be able to do a manual play board read in of the actual state of the board, if the automatically analyzed virtual representation is incorrect
FREQ21.	The user shall be able to terminable all processes of NAO through the UI
FREQ22.	Every action by the robot shall be interruptible and resumable by the UI

Non-Functional Requirements

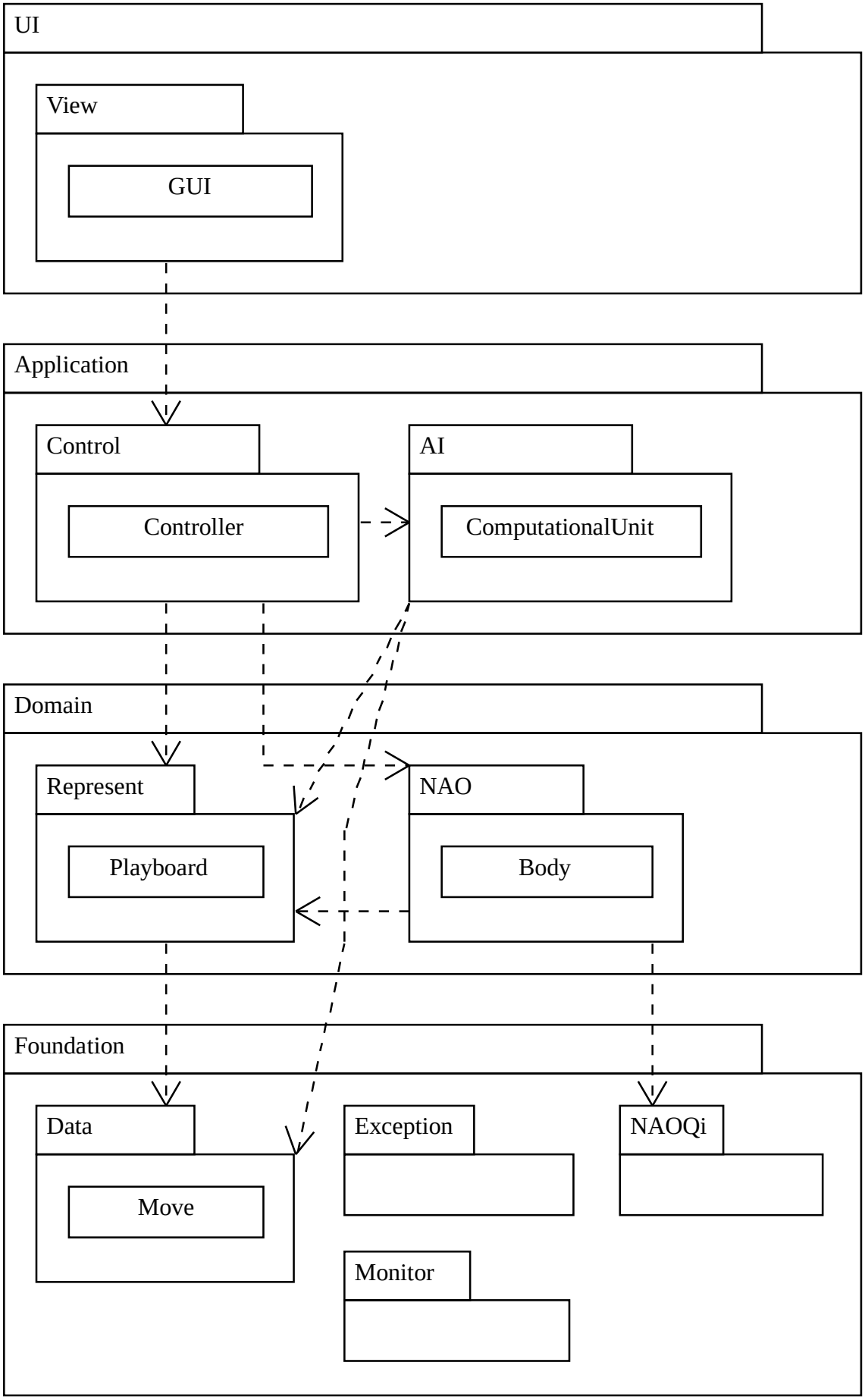
IDs	Description
NFREQ1.	The AI should try to win if possible
NFREQ2.	The AI should try to reach a draw if a win is impossible
NFREQ3.	In every turn the AI should choose an optimal action, taking a minimum of two moves into account
NFREQ4.	The AI shall dynamically adjust its accuracy to the current game situation
NFREQ5.	The AI shall calculate moves space-efficiently
NFREQ6.	The AI shall find shortest paths to men on the board
NFREQ7.	The robot shall place the men precisely
NFREQ8.	The UI shall be easy to use
NFREQ9.	All basic system activities shall be observable via a logger
NFREQ10.	The robot's software shall be useable with Linux, Windows and Mac OS
NFREQ11.	The system shall run stable without crashes
NFREQ12.	The system shall be resistant to exceptions
NFREQ13.	The robot shall respond within approximately 3 seconds
NFREQ14.	The system shall be testable
NFREQ15.	The system shall be well documented
NFREQ16.	The system shall be easy to maintain

System Model

Domain Model

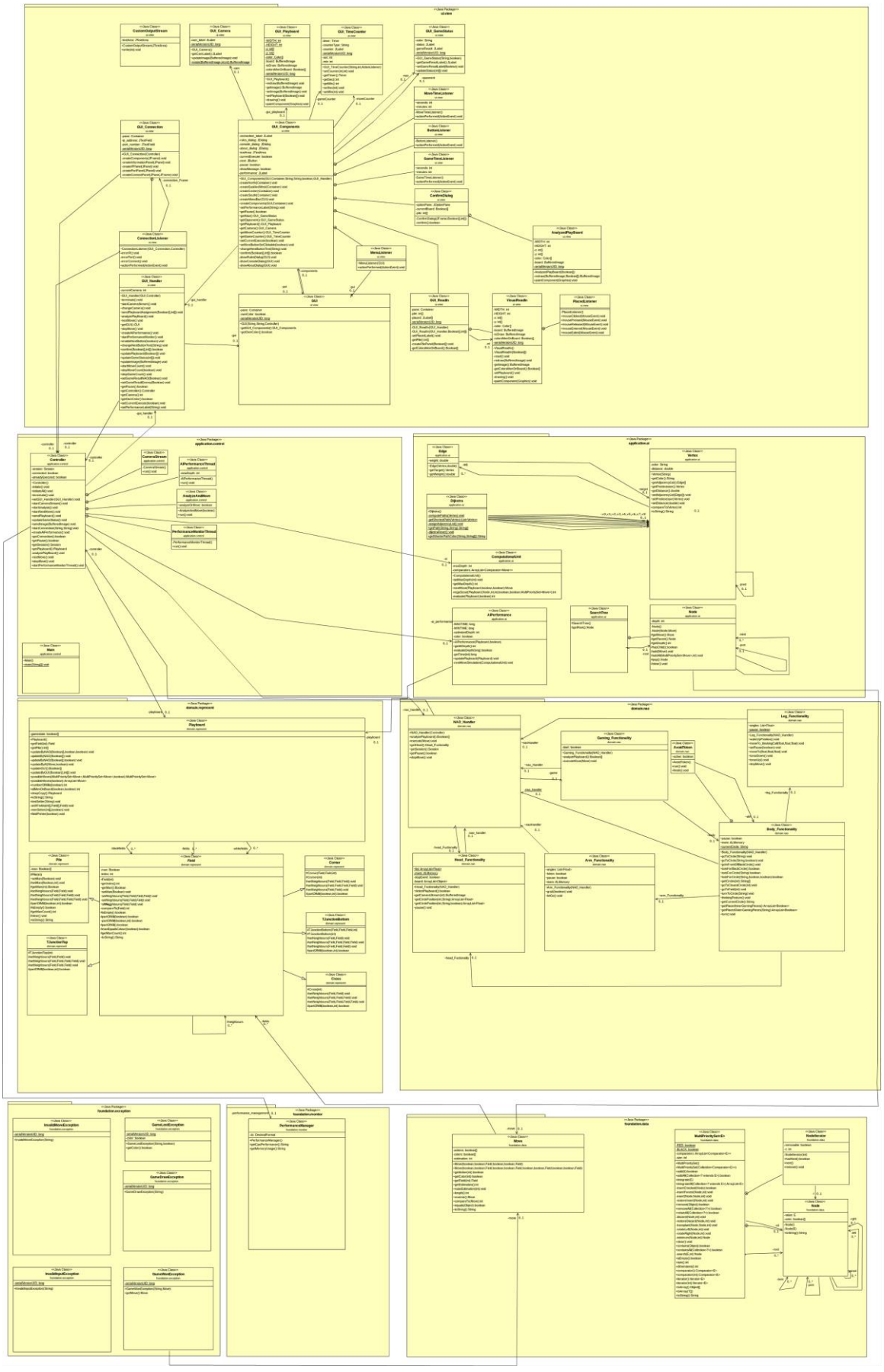


Architecture-Diagram



Class-Diagram

The class-diagram is too large for this document. It is submitted as an external PDF file.



Operation Contracts

OC1

Operation: startConnection(ip: String, port: String)

Preconditions: -

Postconditions:

- The connection to the robot was established

OC2

Operation: initiateAll()

Preconditions:

- The connection to the robot is established
- initiateAll() has not been invoked before
- A single instance of Controller was created

Postconditions:

- A Playboard instance playboard was created
- playboard was initialized in a valid state
- A NAO_Handler instance nao_handler was created
- A ComputationalUnit instance ai was created.
- The playboard was associated with the Controller
- The nao_handler was associated with the Controller
- The ai was associated with the Controller
- Attributes of Controller were initialized

OC3

Operation: analyzePlayboard(): Boolean[]

Preconditions:

- The connection to the robot is established
- Controller is created
- The nao_handler is created
- The playboard is created
- The playboard is in a valid state
- Controller is associated with the playboard
- Controller is associated with the nao_handler

Postconditions:

- All set men on the playboard are contained in a List

OC4

Operation: updateByNao(Boolean[] nao)

Preconditions:

- The Controller is created
- The playboard is created
- The playboard is initialized in a valid state.
- The playboard is analyzed

Postconditions:

- All virtual set men has been updated in the playboard according to their physical one

OC5

Operation: nextMove(board: playboard, color: boolean) : Move

Preconditions:

- The Controller is created
- The ai is created
- The playboard is created
- The playboard is initialized in a valid state.
- The playboard is analyzed

Postconditions:

- One of the best next Moves was calculated
- The Move was instantiated as a new instance move

OC6

Operation: getAIDepth(): Integer

Preconditions:

- The Controller is created
- The playboard is created
- The playboard is in a valid state

Postconditions:

- The depth for the next use of the ai was calculated

OC7

Operation: possibleMoves(input: MultiPrioritySet<Move>, output: MultiPrioritySet<Move>, colour: boolean): MultiPrioritySet<Move>

Preconditions:

- The Controller is created
- The playboard is created
- The ai is created
- ai is calculating the next move

Postconditions:

- All possible moves for the given color were returned as move objects in the set output
- All moves have been integrated into the set input

OC8

Operation: updateByAI(move: Move)

Preconditions:

- The connection to the robot is established
- The Controller is created
- The playboard is created
- The play is in a valid state
- The ai is created
- move is to be physically executed

Postconditions:

- The playboard was updated according to move

OC9

Operation: setAIDepth(depth: Integer)

Preconditions:

- The Controller is created
- The playboard is created
- The playboard is in a valid state
- The ai is created

Postconditions:

- The attribute depth of ai was set

OC10

Operation: execute(move: Move)

Preconditions:

- The connection to the robot is established
- The playboard is created
- The playboard is in a valid state
- The ai is created
- The nao_handler is created
- move is consistent with the actual state of the playboard

Postconditions:

- The men are physically relocated

OC11

Operation: pauseAllProcesses()

Preconditions:

- The connection to the robot is established
- The Controller is created

Postconditions:

- The robot's body parts were transferred into a stable posture
- The robot was halted

OC12

Operation: continueAllProcesses()

Preconditions:

- The connection to the robot is established
- The Controller is created
- The robot is paused

Postconditions:

- The robot has continued its action

OC13

Operation: terminate()

Preconditions:

- The connection to the robot is established
- The Controller is created

Postconditions:

- All processes were terminated, thus the program is closed

System Sequence Diagram