War of Robotcraft

Design Document

Team: A3

Team Members:

Name	**NSID**	**Student ID**
Fu, Chen	chf354	11183491
He, Jiahuan	jih889	11183346
Wang, Shisong	shw940	11157916
Xie, Ruida	rux793	11194258
Yang, Chen	chy202	11183550

Date: Oct 21, 2016

Document History Log:

Version Number	Description of Changes	Approved Date
0.1	First Draft	2016-10-21
0.2	Add architecture description.	2016-10-22
0.3	Added all class in model package.	2016-10-22
0.4	Added classes in view and controller package.	2016-10-22
0.5	Modified a few classes.	2016-10-23
0.6	Added interpreter class.	2016-10-23
0.7	Added UML model.	2016-10-23
1.0	Final edition.	2016-10-23

1. Introduction

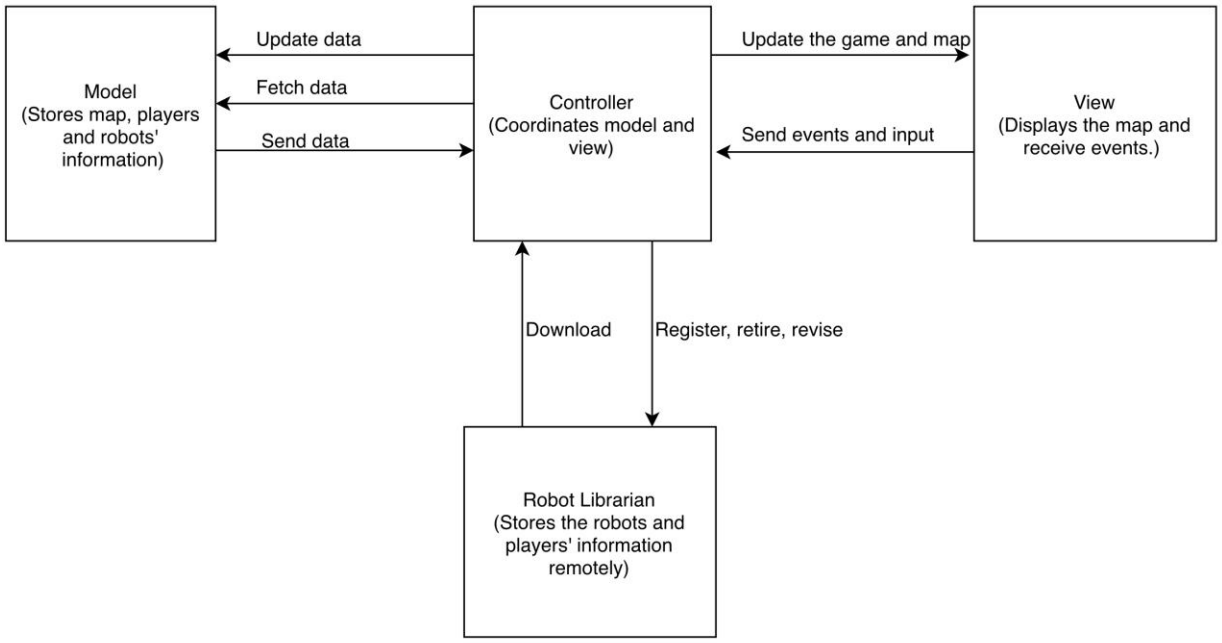
The Design Document is a document which provide a base level description for how the software should be built. It contains narrative and graphics of software design for each component. Topics covered within document including following: architecture, classes and UML diagram. The focus is placed on classes, class hierarches and interactions. In short, this document is meant to allow reader to procced with understanding of how this project works and how to implement in the next stage.

2. Architecture

2.1 Description

We choose Model-View-Controller architecture as our project architecture.
According to our requirement document, our system should update interfaces and robot data frequently, so that there would be lot of communications between these two components. Therefore, we have to connect different data model to each user interface, which will create complex dependencies. However, we would like to have fewer dependencies within our system. Since the MVC architecture can separate backend logic, user interfaces and data cleanly, it will reduce code complexity and solve this problem by decoupling data access and business logic from data presentation and user interaction by introducing an intermediate component: the controller. Thus, changes to the user interface will not affect data handling and that the data can be reorganized without changing the user interface.

2.2 Diagram of MVC architecture



3. Detailed Design

3.1 Packages and Classes

3.1.1 Model Package

3.1.1.1 Coordinate Class

This class is used to create a coordinate for an absolute position in the map. Since this game map is hexagonal, each coordinate consists of three dimensions, which are X, Y and Z. Every object in the map such as the hexagon inside the map and the robot on the map has an absolute coordinate.

Fields:
X: int
Y: int
Z: int

moveXPositive(): void
Summary: this method is to update the coordinate when move forward x-positive direction by 1 step.
Precondition: nothing
Input: nothing
Return: void
Postcondition: x is increased by 1; y is decreased by 1; z is the same.

moveXNegative(): void
Summary: this method is to update the coordinate when move forward x-negative direction by 1 step.
Precondition: nothing
Input: nothing
Return: void
Postcondition: x is decreased by 1; y is increased by 1; z is the same.

moveYPositive(): void
Summary: this method is to update the coordinate when move forward y-positive direction by 1 step.
Precondition: nothing
Input: nothing
Return: void
Postcondition: x is the same; y is increased by 1; z is decreased by 1.

moveYNegative(): void
Summary: this method is to update the coordinate when move forward y-negative direction by 1 step.
Precondition: nothing

Input: nothing
Return: void
Postcondition: x is the same; y is decreased by 1; z is increased by 1.

moveZPositive(): void
Summary: this method is to update the coordinate when move forward z-positive direction by 1 step.
Precondition: nothing
Input: nothing
Return: void
Postcondition: x is decreased by 1; y is the same; z is increased by 1.

moveZNegative(): void
Summary: this method is to update the coordinate when move forward z-negative direction by 1 step.
Precondition: nothing
Input: nothing
Return: void
Postcondition: x is increased by 1; y is the same; z is decreased by 1.

3.1.1.2 Game Class

This class is used to create a new game including a list of players, the number of all players, and the current player who takes control of a play. Whenever a player executes an action that is moving a robot or shooting at a specific direction, the game will be updated the status. Then controller will send these updates to corresponding view objects, and the view object displays them.

Fields:
playerList: LinkedList<Pair<int, Player>>
playerNum: int
currentPlayer: Player

setPlayerPosition(): void
Summary: this method is to set all player's position to corresponding side of the game board.
Precondition: the game is created.
Input: nothing
Return: void
Postcondition:
Set players to position Red and Green if the number of players is 2;
Set players to position Red, Yellow and Blue if the number of players is 3;
Set players to all positions if the number of players is 6.

goNextPlayer(): void
Summary: this method is to let the next player take control of the game

Precondition: nothing
Input: nothing
Return: void
Postcondition: the current player is set to the next player

runPlay(): void
Summary: this method is to run a play
Precondition: nothing
Input: nothing
Return: void
Postcondition: a new play starts

updateGame(): void
Summary: this method is to update the game after an action executed by the current player
Precondition: nothing
Input: nothing
Return: void
Postcondition: the game is updated

3.1.1.3 Map Class

This class is used to create a map model of the game. The map model is consisted of a list of coordinates that determine an absolute position in the map. Also, there is a map size to identify the size of a side of the hexagon map, which is 7 for the number of players is 6 otherwise 5. Any robot shows up on the map has an absolute coordinate. When a robot gets the permission to move or shoot, the relative position for the robot is present on the game board, and the relative position is determined by the abstract coordinate.

Fields:
coordinateList: LinedList<Coordinate>
mapSize: int

Map (int playerNum): void
Summary: this method is to initialize the map with the number of player
Precondition:

1. The playerNum is 6;
2. The playerNum is 2 or 3;
3. Otherwise.

Input: nothing
Return: void
Postcondition: The playerNum is 7, initialize each side as 7 hexagons; otherwise 5 hexagons.

updateMist (): void
Summary: this method is to update the mist on the map according to the robots' positions.
Precondition: nothing

Input: nothing
Return: void
Postcondition: the mist on the map is updated.

3.1.1.4 Player Class

The player class stores the data of player. The player's name, score, the robots the player has, as well as the death status are stored as attribute in this class. Moreover, the player class is also responsible for determining the status of the player and switch turns between players and give feedback to the gameController class. The gameController class gets messages from view classes then manipulate player data. When the game is on progress, the player class is to figure out which robot should move in this play, and move to next play.

Fields:
name: string
score: int
scoutRobot: Robot
sniperRobot: Robot
tankRobot: Robot

Methods:
isDead() : bool
Summary: this method is to return if the robot is dead.
Precondition: the game is not ended.
Input: None
Return: bool
Postcondition: The value is returned.

getCurrentRobot() : Robot
Summary: this method is to return the robot on current play.
Precondition: the game is not ended, the robot and the player are not dead.
Input: None
Return: robot object
Postcondition: The current robot is returned.

goNextRobot() : void
Summary: this method is to go to the next robot. (Run into next play)
Precondition: the game is not ended, the robot and the player are not dead.
Input: None
Return: None
Postcondition: The game runs into next play (It's next robot's turn to play the game).

3.1.1.5 HumanPlayer extends Player

This class extends player as a subclass. It contains all player's fields and attributes along with three new methods which are three actions of human's command. These actions will call the

robots inside the HumanPlayer class to execute specific actions. This class stores information of human player and the robots it possesses. It also determines which is the right robot to play. This class should respond to controller's command of queries.

Methods:
move() : void
Summary: this method moves a robot of the player one cell forward.
Precondition: the robot still has movement points and it's the current player's turn.
Input: None
Return: None
Postcondition: one of the robots is moved.

turn(int direction) : void
Summary: this method makes a robot of the player face a different direction.
Precondition: it's the current player's turn.
Input: int, the directions of the new direction.
Return: None
Postcondition: a robot of the player is turned into a new direction.

shoot(Coordinate coor) : none
Summary: a robot of the player fires to the direction it facing.
Precondition: The robot of this player hasn't shot and the coordinate is within the range.
Input: Coordinate, the coordinate of the target.
Return: none.
Postcondition: the robot of the player has shot, and the health points of all the robots at the target position are deducted.

3.1.1.6 AIPlayer extends Player

3.1.1.7 Robot

This class is the basic entry of the game. Each robot class stores its name, statistics and position. The robot is manipulated by players and perform specific actions. This class is associated with player class as an attribute of it. In this game, there are three types of robots with three sets of values of attributes.

Fields:
type: enum {scout, sniper, tank}
name: string
attackPoint: int
healthPoint: int
movementPoint: int
rangePoint: int
coor: coordinate
direction: int {1, 2, 3, 4, 5}

hasMoved: bool
hasShot: bool

Methods:
turn(int direction) : void
Summary: this method turns the direction of robot.
Precondition: the robot is alive.
Input: int direction.
Return: none.
Postcondition: the robots facing direction is turned.

move(Coordinate coor) : void
Summary: this method moves the robot once.
Precondition: the movement points are not used up. The robot is alive.
Input: Coordinate, the target position
Return: none.
Postcondition: the robot is moved one cell forward.

damaged(int attackPoint) : void
Summary: the robot get fired by another robot
Precondition: the robot is alive.
Input: the attachPoint
Return: none.
Postcondition: the robot's health point is deducted by certain points. If the robot's health point become zero, the robot dies.

isDead() : bool
Summary: to determine the if the robot is dead.
Precondition: none
Input: none
Return: none.
Postcondition: the robot's status is returned.

shoot(Coordinate coor) : none
Summary: to perform the shot action.
Precondition: The target is in the range. The robot has
Input: coordinate: the target position.
Return: none.
Postcondition: the robot of the player has shot, and the health points of all the robots at the target position are deducted. And the robot's play finishes.

3.1.2 View Package

3.1.2.1 GameStartView Class

The game start view welcome players and give the choice that start new game, manage the robots and exit the game. This class is used to create the game start frame including game name label, start game button, garage button, and exit button. When players click, or press these button, the controller will respond these events, and send the result to set game mode view or garage view.

Fields:

gameNameLabel: JLabel
startButton: JButton
garageButton: JButton
exitButton: JButton

Methods:

gameStartView():void

Summary: this method is to the game start frame, with game name, a button to start game, a button to go to garage and a button to quit the game.
Precondition: nothing
Input: nothing
Return: void
Postcondition:
The game start view constructed.

3.1.2.2 SetGameModeView Class

The set game mode view displays the various mode, give player choices to set game mode. This class is used to create the view frame including a list of radio button of game mode, a list of radio button of player's number, a list of combo box of player type, and a start button. After a player chooses the start game button, the controller will create the set game mode view. And any choice in this view will send to the controller.

Fields:

gameModeRadioButtonList: ArrayList<JRadioButton>
playerRadioButtonList: ArrayList<JRadioBUtton>
playerTypeComboBoxList: ArrayList<JComboBox>
startGameButton: JButton

Methods:

setGameModeView(): void

Summary: this method is to construct the set game mode frame, a list with three radio button to set game mode, a list of three radio button to choose player's number, a list of combo box to set player type, and a button to start game.
Precondition: nothing

Input: nothing
Return: void
Postcondition:
The set game mode view constructed.

3.1.2.3 GameBoardView Class

The game board view show the current game scene, display the game board, the timer and the information of player status, and give the choice to access the current game: end play, give up or go back home. This class is used to create the game board view. It has a method to access the information when the controller notifies to change.

Fields:
hexagonLabelList: ArrayList<JLabel>
timerLabel: JLabel
endPlayButton: JButton
surrenderButton: JButton
homeButton: JButton

Methods:
GameBoardView():
Summary: this method is to construct game board view frame, a list of hexagon label, a timer label, a button to end play, a button to give up, a button to go home view, and a player status panel.
Precondition: nothing
Input: nothing
Return: void
Postcondition:
The game board view constructed.

3.1.2.4 PlayerStatusView Class

The player status view display the player name, and the robot team of the player, and their health point. This is a class for the panel to be placed in the frame of game board view. It has a various of label to show the status.

Fields:
playerNameLabel: JLabel
soutLabel: JLabel
soutHealthPointLabel: JLabel
sniperLabel: JLabel
sniperHealthPointLabel: JLabel
tankLabel: JLabel
tankHealthPointLabel; JLabel

Methods:
PlayerStatusView(): void
Summary: this method is to construct the player status panel, a label with player name, a scout label, a label with its health point, a sniper label and label with its health point, and a tank label and a label with its health point.
Precondition: nothing
Input: nothing
Return: void
Postcondition:
The player status view constructed.

3.1.2.5 GarageView Class
The Garage view display the all robots, and player can manage the robot: register a new robot, revise and retire an exiting robot. After player choose garage button in the gameStartView, the controller will send a request to display to GarageView.

Fields:
robotListView: JList
registerButton: JButton
reviseButton: JButton
retireButton: JButton
nameTextField: JTextFied

Methods:
GarageView(): void
Summary: this method is to construct the garage view frame, a list of robot, with name, a button to register, a button to revise and a button to retire.
Precondition: nothing
Input: nothing
Return: void
Postcondition:
The garage view constructed.

3.1.3 Controller Package
3.1.3.1 GameController
The game controller is to keep all views and models in control. It stores a Game instance, a GameStartView, a SetGameModeView instance, a GameBoardView instance, a PlayerStatusView instance, and a GarageView instance. It will handle all button clicked events and all key typed events that take place in views. Also, it will update view depending on every change.
Fields:
game : Game
gameStartView : GameStartView

setGameModeView : SetGameModeView
gameBoardView : GameBoardView
playerStatusView : PlayerStatusView
garageView : GarageView

ActionPerformed(ActionEvent e) : void
Summary: handle all the button clicked event
Precondition: button is clicked in views
Input: e is an ActionEvent from clicked button
Return: none
Postcondition: a new SetGameModeView instance assigned to setGameModeView and displayed if startButton clicked; a new GarageView instance assigned to garageView and displayed if garageButton clicked; game quit if exitButton clicked; a new GameBoardView instance assigned to gameBoardView and displayed if start game button clicked; switch to next play if endPlayButton clicked; the specific player marked died if surrenderButton clicked; go back to gameStartView if homeButton clicked; a new robot registered if registerButton clicked; the specific robot revised if reviseButton clicked; the specific robot retired if retireButton clicked

keyTyped(KeyEvent e) : void
Summary: handle all the key typed event
Precondition: key is pressed in views
Input: e is a KeyEvent
Return: none
Postcondition: enter turning mode if pressed “T”; enter moving mode if pressed “M”; enter shooting mode if pressed “S”; the playing robot’s direction rotate n (n is the number of pressed number key) unit if pressed number key and in turning mode; the playing robot shoots n (n is the number of pressed number key) unit of range if pressed number key and in shooting mode

startTimer() : void
Summary: start a new timer to be displayed in GameBoardView
Precondition: none
Input: none
Return: none
Postcondition: a new timer is started

updateRobotPosition() : void
Summary: notify GameBoardView
Precondition: a robot moved
Input: none
Return: none
Postcondition: GameBoardView displays change of the specific player’s position

updateRobotHealth() : void

Summary: update a robot's health point that is displayed on map
Precondition: a robot is damaged
Input: none
Return: none
Postcondition: update the health point of the damaged robot, and remove the robot from map if it has been destroyed

updatePlayerDeath() : void
Summary: mark a player as died
Precondition: a player died
Input: none
Return: none
Postcondition: the died player is marked as died on the game board

updateGarage() : void
Summary: update garage view
Precondition: registered, revised, or retired a robot
Input: none
Return: none
Postcondition: a new-registered robot added to garage if registered a robot; the displayed information of the revised robot changed if revised a robot; a robot is marked retired if retired a robot

3.1.4 WarOfRobotcraft package

3.1.4.1 Initial Class

This class is the entry point for the whole game system. When the game gets started, it starts from the main function of this class. Then the a controller object is created, and the game starts to run.

Main(String[] args): void
Summary: this method is to start a new game.
Precondition: nothing
Input: nothing
Return: void
Postcondition: the game gets started.

3.1.5 Util package

3.1.5.1 Interpreter Class

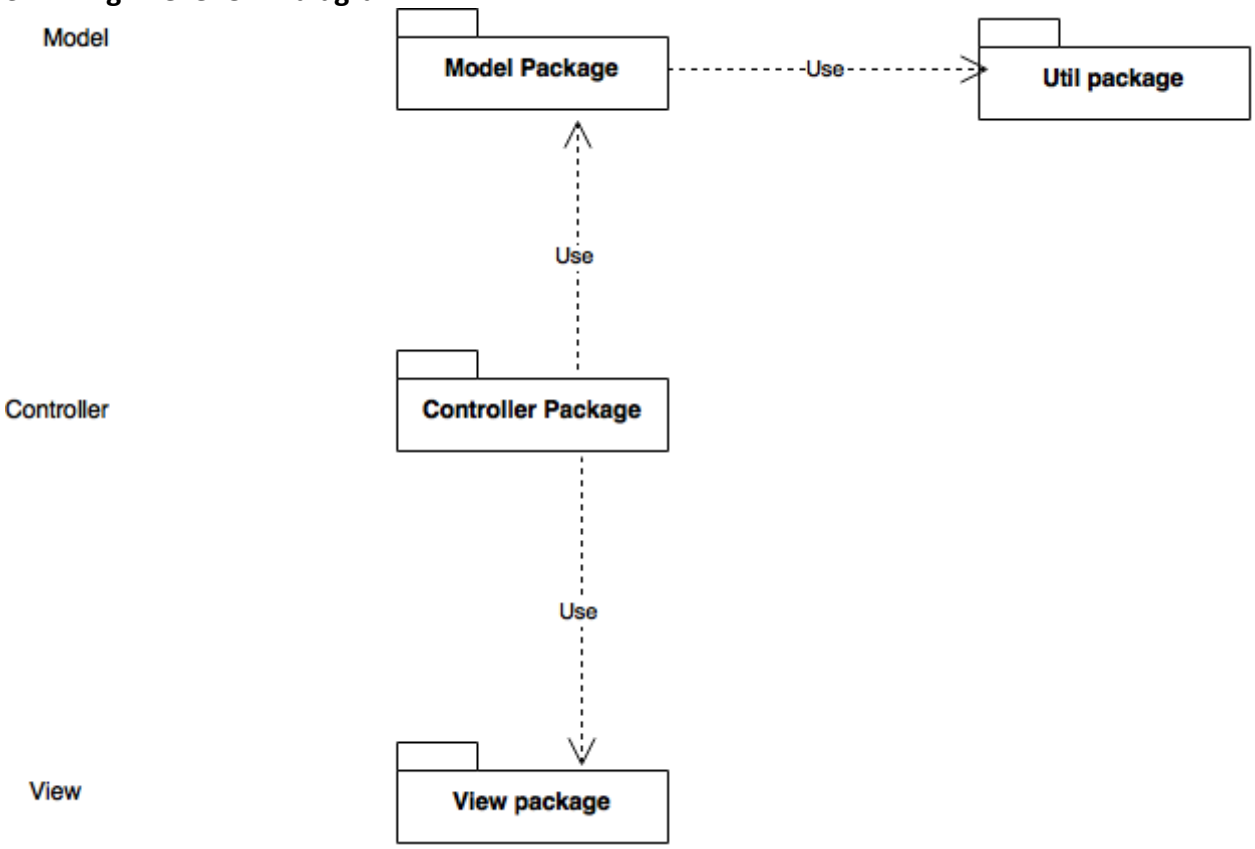
This class is used to interpreter the Forth script to Java code. The AI robots makes decisions by Forth script, which returns a string of actions and coordinate to the game. Then the game updates according to the return values. So the interpreter is used by the AI Player.

forthToJava(string forthScript): String
Summary: this method is to translate the Forth script to java source code.

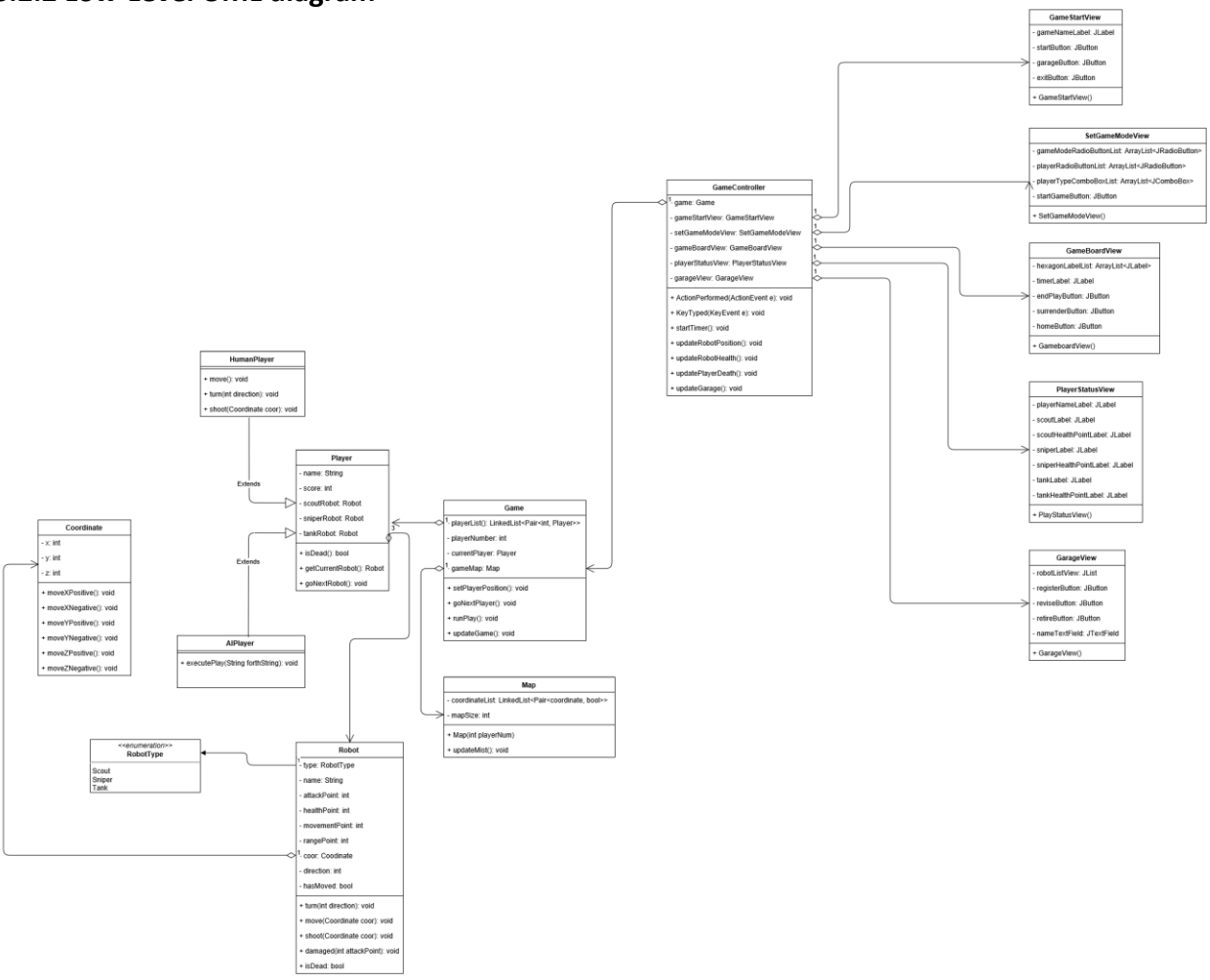
Precondition: nothing
Input: nothing
Return: void
Postcondition: the forth script is translated to the java source code.

3.2 UML

3.2.1 High-Level UML diagram



3.2.2 Low-Level UML diagram



4. Amendment

We make some changes from our requirement document:

- We separate Robot Librarian from our system. The Robot Librarian now is an independent system.
- We add an “Start Game” button in the SetGame view. This button will confirm the user select game model. We missed this button in our requirement document.