Manual of Asterobots

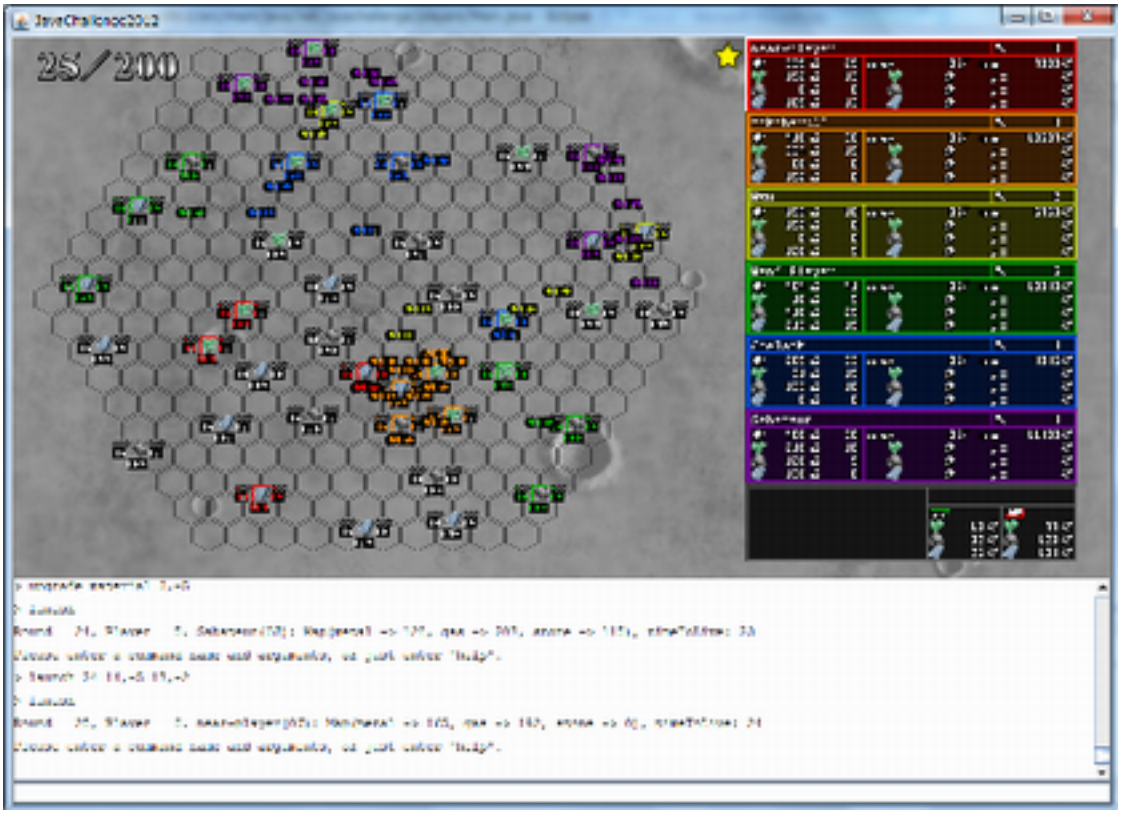
- No pain, No vein -

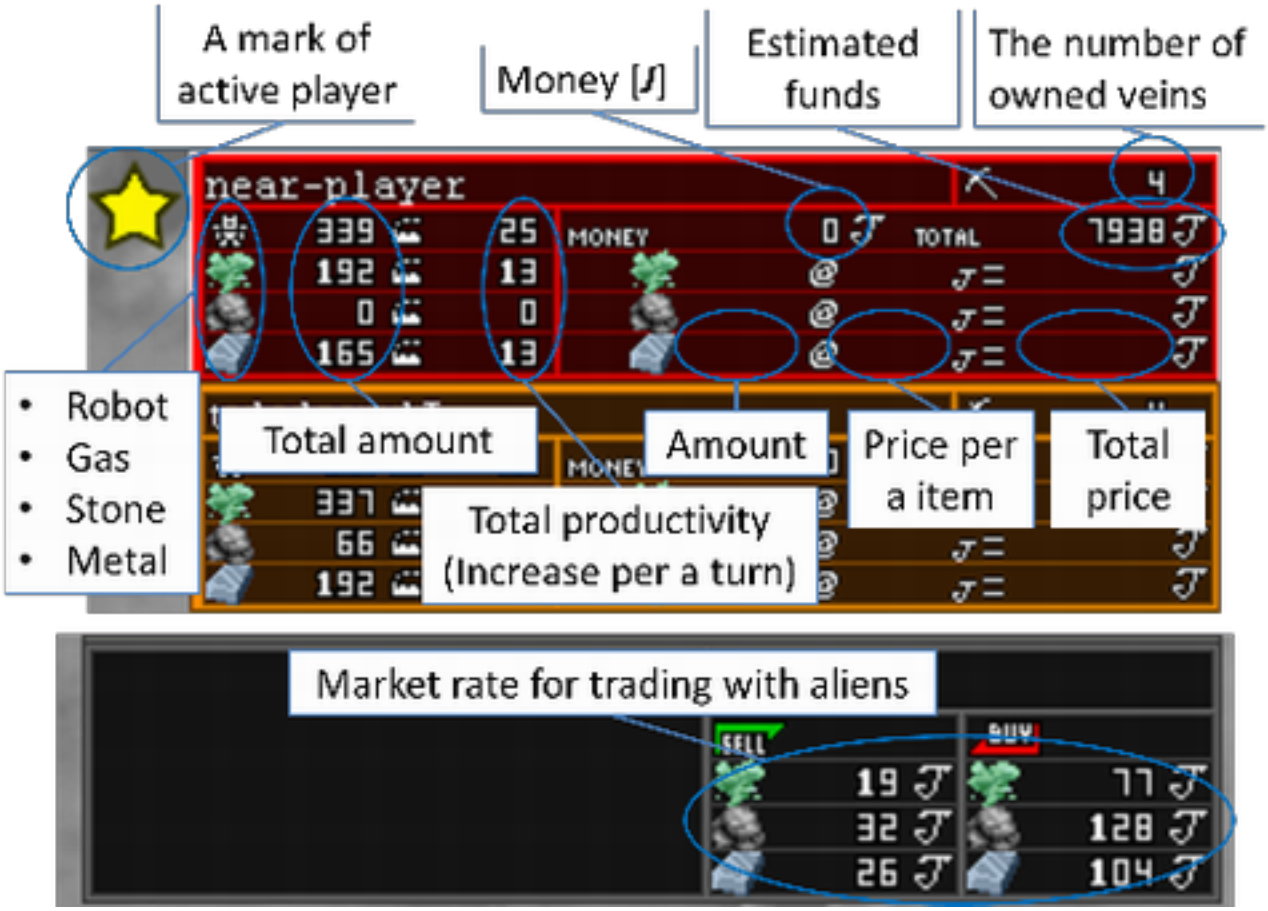
Final version (16th November 2012, 18:00 GMT: +9:00)

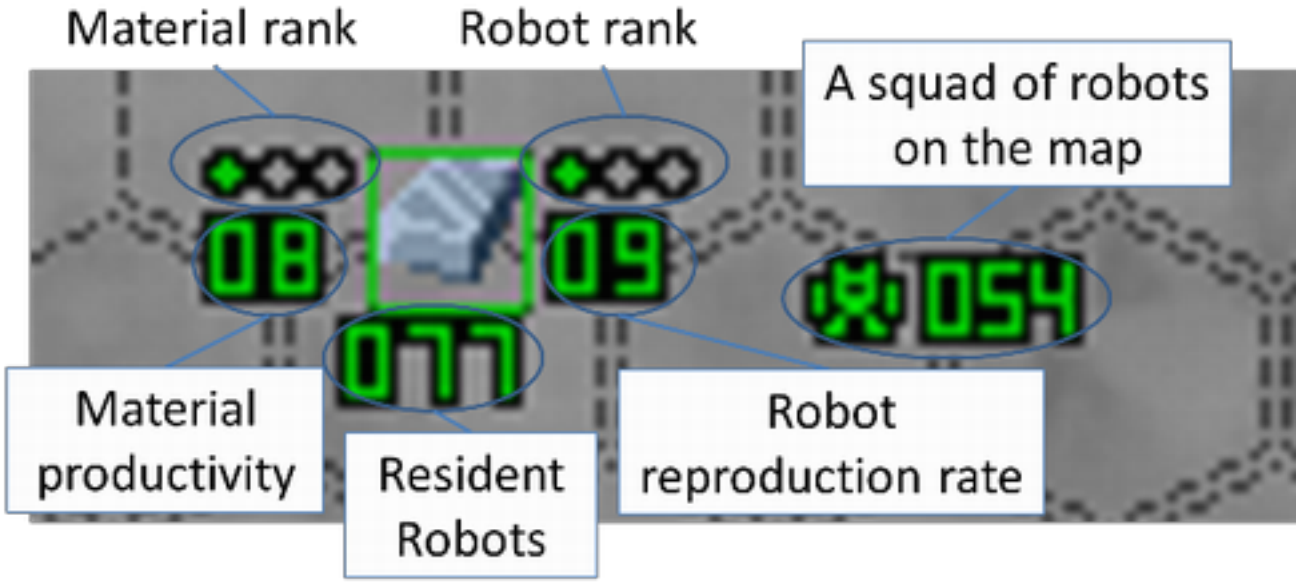
Notandum

If you have any question or notice any bugs, please contact JavaChallege2012 Committee (javachallenge2012@googlegroups.com).

Screen




A mark of active player
Money [ℐ]
Estimated funds
The number of owned veins
near-player
MONEY
TOTAL
Robot
Gas
Stone
Metal
Total amount
Amount
Price per a item
Total price
Total productivity (Increase per a turn)
Market rate for trading with aliens
SELL
BUY


Material rank
Robot rank
A squad of robots on the map
Material productivity
Resident Robots
Robot reproduction rate

Story

The resources of the earth will be exhausted soon! Fortunately, we have a unique unmanned rocket which has landed on an asteroid plenty of natural resources. However, hostile aliens are there as well. The rocket contains robots who can battle aliens to conquer veins, develop mines on veins to get resources and produce robots to extend our territory automatically. To get resources efficiently, the robots require a good AI (artificial intelligence). Now let us develop AI programs for the robots and battle each other to elect the best AI program!

Overview

Number of players

6 people

Programming language for developing AI program

Java only

Goal

Players gather materials by extending their territory and trading materials and money with other players and aliens. Players produce robots in their veins by consuming materials and launch robots to other veins to conquer them. The winner will be the player who has conquer the greatest number of veins.

Player status and initial values (modified at 14th, 11:00)

- Money (which unit is ***J***, called “Java”)
  - Initial money is **0** [***J***]
- The amount of each material
  - Initial amount of each material is 0.
- Veins
  - You can select 2 veins at the beginning of the game. See Vein Section.
- Robots on the field
  - There are no robots on the field (except in veins) when the game starts.

Concepts in game

Field

The field is a large hexagon formed by small hexagon tiles. One side of the large hexagon contains 10 adjacent small hexagon tiles. 40 vertices of all 600 vertices contain a vein and other 560 vertices have nothing. 14 veins produce Gas, 13 veins produce Stone and 13 veins

produce Metal. The locations of the veins are decided randomly when the game starts. Note that two veins are separated by at least three small hexagon's sides.

Coordinate system

The left part of Figure 1 shows the coordinate system of a 2 size map. The right part of Figure 1 shows that the coordinate system is based on triangle tiles.

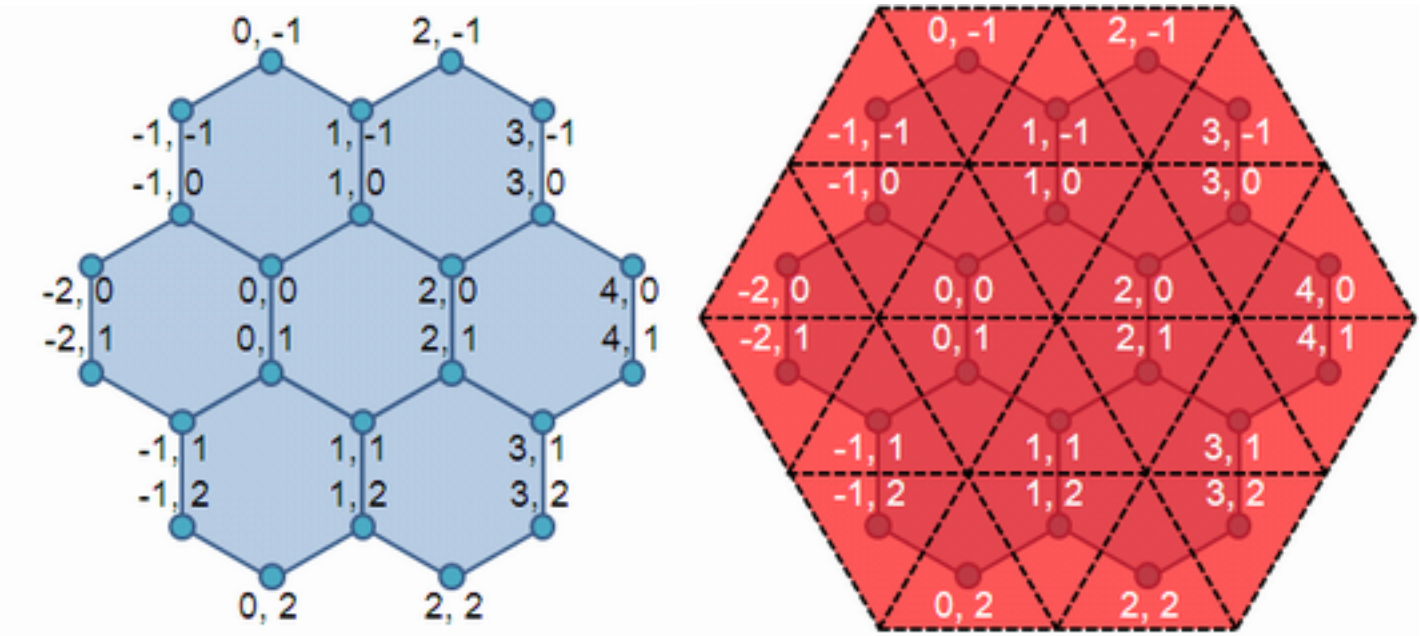


Figure 1. A field of 2 size and coordinates

Vein (modified at 14th, 13:35)

The vein has the following parameters.

- The number of resident robots
- The material produced by the vein
- Material productivity: the amount of material produced per turn
- Robot reproduction rate: the number of new robots created per turn
    - Initial values of both productivity and reproduction rate are more than or equal to 5 and less than or equal to 9 (randomly decided).
- Owner (player id)
- Material rank (1-3)
- Robot rank (1-3)

The productivities and reproduction rates are calculated by the formula:

initial productivity * (2 ^ (*rank* - 1)).

Both material and robot ranks are decreased by 1 (the minimum rank is 1) when the owner of the vein changes. Players can upgrade material and robot ranks by consuming the following materials up to 3.

Material rank

The material productivity is doubled when increasing 1 rank.

- Upgrading the rank from 1 to 2: 200 [Gas] and 100 [Stone] are required
- Upgrading the rank from 2 to 3: 100 [Gas] and 300 [Stone] are required

Robot rank

The robot reproduction rate is doubled when increasing 1 rank.

- Upgrading the rank from 1 to 2: 200 [Gas] and 200 [Metal] are required
- Upgrading the rank from 2 to 3: 300 [Stone] and 500 [Metal] are required

Material

There are 3 kinds of material: Gas, Stone and Metal. You can get materials from your veins and can also trade materials and money with other players or even with the aliens (buy or sell). The market rate is flexible. Please see Trading with aliens.

Robot

The launched robots on the field can only stay on vertices. Robots move 1 small hexagon side at the end of the turn of the player owning them.

Game flow

The game has two phases: a phase to select veins and the main game phase. During the phase to select veins, each player selects two veins. The game is turn-based. The player order is decided when game starts. Note that “Player 1” is the player chosen to start the game in the order decided at the game start.

Vein selection phase

Players select two veins at the start of the game. All veins have initial resident robots. When selecting a vein, the player owns the vein as well as the resident robots. Unselected veins are owned by neutral players (aliens). Aliens do not do anything, but you should fight them to conquer their veins. Note that the game engine select a vein with a rule (likely randomly) instead of the player’s selection when a player select an invalid location (not vein or other player’s vein). Each AI program should end this phase until 10 seconds. The API provides a method for saving a temporal selection. When a program exceeds the time limit, the game engine consider that the program selects saved temporal location or invalid location (if no temporal selection).
N players select veins in the following order:

Player1 -> Player2 -> ... -> PlayerN -> PlayerN -> PlayerN-1 -> ... -> Player2 -> Player1.

Main phase

1. The active player executes any number of commands in no particular order. Players can execute the same commands. Note that the maximum executable commands in a turn are 100 commands. The 101th or later commands and invalid are ignored. Invalid commands are also ignored.
    - Upgrading material / robot rank
    - Trading materials and money with other players or aliens
    - Launching robots from your veins to other veins
2. At the end of each turn, the following actions are executed.
    - Increasing resident robots and materials of the active player veins
    - Moving launched robots of the active player to the next connected vertices. Conquering other veins when reaching these.
    - Increasing the round number when the last player finishes his/her turn
3. Change the active player to the next player which has more than or equal to one vein
    - The players which have no vein are skipped cosnidered as dead and cannot execute any commands
    - **When squads of players who have no vein conquer some veins, the players come back as living players**

Active players change in the following order:

Player1 -> Player2 -> ... -> PlayerN -> (Increaing round number) -> Player1 -> Player2 -> ... -> PlayerN -> (Increaing round number) -> Player1 -> ....

Only an active player can execute commands. Note that each AI program should finish this phase until **1** seconds. The API provides a method for saving a temporal command list. When a program exceeds the time limit, the game engine consider that the program decide the temporal commands.

We will explain about the commands which players can execute in their turn in the following sections.

Upgrading material / robot rank

Players upgrade material / robot rank by consuming materials. See Vein Section.

Trading materials and money

Trading with other players

Players can make offer and demand to sell or buy materials. The offers and the demands consist of the one kind of material with the wanted amount of it and the price [***J*** / piece]. The offers and demands are published for a player until his/her next turn ends. The offers and demands are automatically canceled if other players ignore them. Players cannot make more than one demand or offer per material and cannot make both a demand and an offer for the

same material at the same time. Note that materials or money of the player decrease when proposing offers and demands, the materials or money of the player return back when canceling them.
Players also accept offers / demands to buy or sell materials from other players by executing **buy** / **sell** commands. The trade is immediately established when other players accept in their turn. The trade can be established by partially buying / selling materials. Here is a sample trading flow.

1. Player 3 offers 50 [Gas] then finishes his/her turn.
2. Player 4 ignores the offer then finishes his/her turn.
3. Player 5 buys 10 [Gas] from Player 3 then finishes his/her turn.
4. Player 6 buys 40 [Gas] from Player 3 then finishes his/her turn.
5. Player 1 does not know the above trade because Player 3's offer is completed.

Trading with aliens

Players can always sell or buy materials with aliens in his/her turn independently on other players. The market rate is calculated by the following formulas:
total = *the total material amount of all material kinds which all players have*
amount = *material amount of each material kind which all players have*
buying price = 100 * (*total* + 300) * (*total* + 300) / (*amount* + 100) / (*amount* + 100) / 9
selling price = *buying price* / 4.
Note that 100 * (*total* + 300) * (*total* + 300) should be calculated as a long value on JVM.

Launching a squad of robots

Players can launch a squad of robots from his/her vein to other player's vein. Squad move 1 small hexagon side at each of its owner's active turn (it means they move from a vertex to the connected vertex). Players can select a path from start vein to goal vein or can select the shortest path by abbreviating route indication. When robots conflict other robots on a vertex or reach a vein, the following processes are executed.

When a squad conflicts other squads on a vertex

Both squads lose the same number of robots, which is equal to the number of robots of the smallest squad of the two. e.g. When Player1's squad of 10 robots conflicts Player2's squad of 5 robots, only Player1's squad of 5 robots survive. If both squad contain the same number of robots, neither of them survive.

When a squad reaches his/her owned vein

The number of the resident robots on the vein increase by the number of robots in the squad.

When a squad reaches another player's or aliens owned vein

Both the squad and the resident robots lose the same number of robots. When the number of robots of the squad is greater than the number of the resident robots, the squad owner conquer the vein.

Game end

End conditions

When one of the following condition is met, the game ends.

- A player conquers all other players' veins except aliens' veins. Even if squads of the other players exist on the map, the player wins.
- 200 rounds end

Player ranking

The player ranking is decided by the following numbers. The time to live is most important to win.

1. The time to live, that is, the number of the last round where the player were active (considered as living)
2. The number of owned veins
3. The total number of robots
4. The current amount of money added to the price of all the owned materials if it were sold to the aliens
5. The turn order (a lesser id is stronger than a grater id)

Input/output for AI program

You can develop your AI program to manipulate your player by using our API. Please see API Document (it will be change until 16th November 2012, 21:00 GMT: +9:00). You can download the game engine version 0.9.1 for user play mode only here.

Input

- Field including veins and squads
- Status of all veins
- Status of all players
- Existing offers and demands
- Market rate for trading with aliens

The programs can not get command logs via API. But your program can store your commands. Note that all object given by API are immutable. Thus the objects always have same state independently on game progression.

Output

Vein selection phase

The location of the vein to select

Main phase

The program can send the list of the following command in no particular order to execute. When multiple commands are given, they are executed in the order given by the list below. And ignored invalid command.

- **BuyFromAlienTradeCommand** with the material and its amount
- **SellToAlienTradeCommand** with the material and its amount
- **BuyFromOfferCommand** with the player trade and the material amount
- **SellToDemandCommand** with the player trade and the material amount
- **OfferCommand** with the kind of the material, the material amount and the price per piece
- **DemandCommand** with the kind of the material, the material amount and the price per piece
- **LaunchCommand** with the number of robots, the departure point, the destination
- **LaunchCommandWithPath** with the number of robots, the departure point, the destination and the path
- **UpgradeMaterialCommand** with the location of the vein to upgrade
- **UpgradeRobotCommand** with the location of the vein to upgrade

External library

You can use the following library which are contained by the provided package.

- Scala standard library 2.9.1
- Guava 13.0.1
- Apache Commons 2.6

Contest regulation

When participants violates the following prohibited acts, they are eliminated.

- The submitted program CANNOT communicate other programs including programs developed by other participants except for the game engine and the above libraries.
- The submitted program CANNOT use any reflection API, that is, it CANNOT manipulate the information of the game without our API.

Sample code as the API tutorial

```
public class YourTeamPlayer extends ComputerPlayer {
  @Override
```

```
public String getName() {
  return "YourTeamName";
}

@Override
public TrianglePoint selectVein(Game game) {
  // Store your temporal selection. When your program exceeds
  // the time limit (10 second), this selection are accepted.
  this.saveTemporalVeinLocation(Make.point(0, 0));

  // Should wrap any collections acquired by the API
  // around ArrayList or some Java collections.
  List<Vein> veins =
    new ArrayList<Vein>(game.getField().getVeins());
  Collections.shuffle(veins);
  for (Vein vein : veins) {
    // Find neutral veins (not selected yet)
    if (vein.getOwnerId() == game.getNeutralPlayerId()) {
      return vein.getLocation();
    }
  }

  // Return the invalid locations, thus,
  // the game engine selects a vein (likely randomly).
  return Make.point(Integer.MAX_VALUE, Integer.MAX_VALUE);
}

@Override
public List<Command> selectActions(Game game) {
  // Note that invalid commands are ignored and
  // skipped to the next command in the list.
  List<Command> commands = new ArrayList<Command>();

  // Note that commands are executed in ascending order of index.
  commands.add(Commands.sellToAlienTrade(Material.Gas, 1));
  commands.add(Commands.sellToAlienTrade(Material.Metal, 1));

  // Display all your veins using a list.
  int myId = game.getMyPlayerId();
  List<Vein> veins = game.getField().getVeins(myId);
  for (Vein vein : veins) {
    System.out.println(vein.getLocation() + ": " + vein);
  }
```

```
    // Display all your veins using a map.
    Map<TrianglePoint, Vein> veinMap =
      game.getField().getVeinMap(myId);
    for (Entry<TrianglePoint, Vein> pointAndVein :
        veinMap.entrySet()) {
      TrianglePoint p = pointAndVein.getKey();
      Vein vein = pointAndVein.getValue();
      System.out.println(p + ": " + vein);
    }

    // Store your temporal command list. When your program exceeds
    // the time limit (1 second), this command list are executed.
    this.saveTemporalCommands(commands);

    // Do too hard works to exceed the time limitation
    try {
      Thread.sleep(2000);
    } catch (InterruptedException e) {}

    commands.add(Commands.buyFromAlienTrade(Material.Stone, 1));

    // Return your command list. When your program DOES NOT exceed
    // the time limit (1 second), this command list are executed.
    return commands;
  }
}
```

Sample code 2 as the API tutorial

```
public class SamplePlayer extends ComputerPlayer {
  /**
   * The neutral player id is -1.
   */
  static final int NEUTRAL_OWNER_ID = -1;

  @Override
  public String getName() {
    return "Sample Player";
  }

  @Override
  public TrianglePoint selectVein(Game game) {
    // Store your temporal selection. When your program exceeds the time
```

```
    // limit (10 second), this selection are accepted.
    this.saveTemporalVeinLocation(Make.point(0, 0));

    Field field = game.getField();
    List<Vein> allVeins = field.getVeins();
    Vein selectVein = allVeins.get(0);

    Map<Vein, Integer> veinScores = new HashMap<Vein, Integer>();

    for (Vein vein : allVeins) {
      if (vein.getOwnerId() != NEUTRAL_OWNER_ID) {
        veinScores.put(vein, Integer.MIN_VALUE);
        continue;
      }
      int score = vein.getNumberOfRobots();
      veinScores.put(vein, score);
    }

    int maxScore = Integer.MIN_VALUE;
    for (Vein vein : allVeins) {
      int veinScore = veinScores.get(vein);
      if (veinScore > maxScore) {
        selectVein = vein;
        maxScore = veinScore;
      }
    }

    return selectVein.getLocation();
  }

  @Override
  public List<Command> selectActions(Game game) {
    List<Command> commands = new ArrayList<Command>();

    Field field = game.getField();
    Player myPlayer = game.getMyPlayer();

    List<Vein> myVeinList = field.getVeins(game.getMyPlayerId());
    List<Vein> veinList = field.getVeins();
    // To sort veins, make a mutable list of veins, which are still
immutable.
    List<Vein> mutableVeinList = new ArrayList<Vein>(veinList);
```

```
// Launch robots
for (Vein myVein : myVeinList) {
  Collections.sort(mutableVeinList, new VeinDistanceComparator(myVein));
  Vein to = mutableVeinList.get(0);

  if (myVein.getNumberOfRobots() > 100) {
    commands.add(Commands.launch(myVein.getNumberOfRobots() - 10,
        myVein.getLocation(), to.getLocation()));
  }
}

// Upgrade veins
for (Vein myVein : myVeinList) {
  if (myVein.getRobotRank() >= 3) {
    continue;
  }
  boolean upgrading = true;
  for (Material material : Material.values()) {
    int materialAmount = myPlayer.getMaterial(material);
    if ((myVein.getRobotRank() == 1 && game.getSetting()
        .getMaterialsForUpgradingRobotRankFrom1To2(material) >
            materialAmount)
        || (myVein.getRobotRank() == 2 && game.getSetting()
            .getMaterialsForUpgradingRobotRankFrom2To3(material) >
                materialAmount)) {
      upgrading = false;
    }
  }
  if (upgrading) {
    commands.add(Commands.upgradeRobot(myVein));
    break;
  }
}

// Trade with aliens
AlienTrade alienTrade = game.getAlienTrade();
for (Material material : Material.values()) {
  int amount = game.getMyPlayer().getMaterial(material);
  if (amount > 1000) {
    commands.add(Commands.sellToAlienTrade(material, amount - 250));
    commands.add(Commands.offer(material, amount - 250,
        alienTrade.getSellPriceOf(material) * 2));
  }
```

```
      if (amount < 100) {
        List<PlayerTrade> playerTrade = game.getOffers(material);
        if (!playerTrade.isEmpty()) {
          Math.min(playerTrade.get(0).getAmount(),
              Math.min(myPlayer.getMoney() /
                  playerTrade.get(0).getPricePerOneMaterial(), 100));
        } else {
          commands.add(Commands.buyFromAlienTrade(material,
              Math.min(100, myPlayer.getMoney() /
                  alienTrade.getBuyPriceOf(material))));
        }
      }
    }
    return commands;
  }

  /**
   * A class for comparing distance to sort veins.
   */
  class VeinDistanceComparator implements Comparator<Vein> {
    private Vein vein;

    public VeinDistanceComparator(Vein vein) {
      this.vein = vein;
    }

    @Override
    public int compare(Vein o1, Vein o2) {
      if (o1.equals(vein)) return 1;
      if (o2.equals(vein)) return -1;

      if (o1.getOwnerId() == vein.getOwnerId()) return 1;
      if (o2.getOwnerId() == vein.getOwnerId()) return -1;
      return vein.getDistance(o1) - vein.getDistance(o2);
    }
  }
}
```

Acknowledgement

This game is inspired by “Die Siedler von Catan (The Settlers of Catan)” and “Galcon”. We are deeply grateful to the authors.

Committee

Kazunori Sakamoto	(Waseda University)
Hiroaki Hosono	(Tokyo Institute of Technology)
Seiji Sato	(Waseda University)
Daniel Perez	(Waseda University)
Dai Hamada	(Waseda University)
Shogo Kishimoto	(Tokyo Institute of Technology)
Masayuki Asakura	(Waseda University)
Ryohei Takasawa	(Waseda University)
Fumiya Kato	(Waseda University)
Masahiko Wada	(Waseda University)
Naohiko Tuda	(Waseda University)
Daichi Ota	(ACCESS,INC.)