28th ACM International Collegiate Programming Contest

Asia Regional Contest 2003, Aizu, Japan

Java Challenge Contest

Snakes and Rabbits

Contents

November 1-3, 2003
Aizu-Wakamatsu

1. General Information

The big snake was placed on the magical island with rabbits playing on the wonderful grass. Rabbits are fat and move slowly. But they can jump on stones as well as hide outside the borders of the island's grass area. Under a magical force the snake can't stop its movement. It can only control its direction of movement. The snake must collect rabbits for leaving this island.

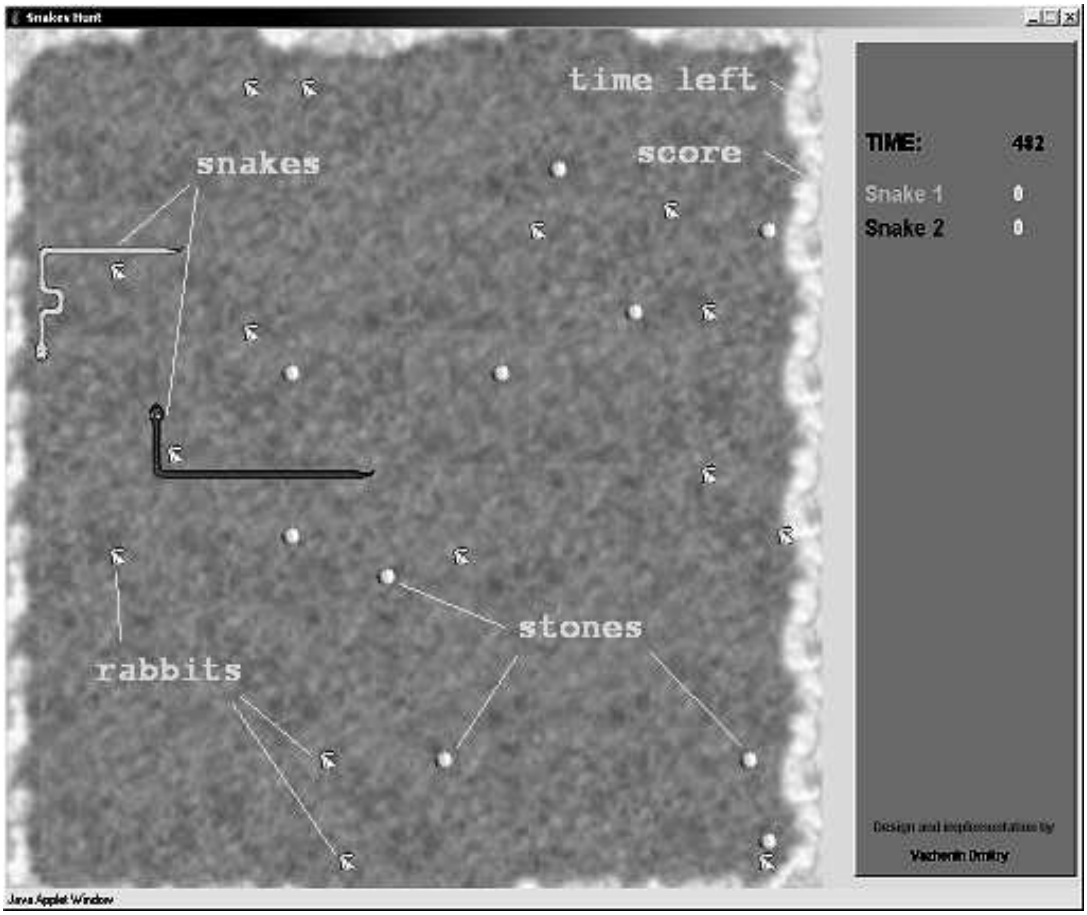
Your mission is to help the snake in collecting the rabbits. Importantly, there is an opponent snake, which implements the same job. The winner of this competition should collect more rabbits than the opponent. Design an effective algorithm and write a program simulating the snake behavior.

2. Control Operations

The island size is 40x40 cells. The cell coordinates can be changed from 0 to 39. The cell with zeroed coordinates is in the upper-left corner. Each cell can be empty, or occupied by a snake body, stone and/or a rabbit. Snakes have 15 cells in length. During one turn a snake can only move to neighboring cells. Each round lasts for fixed amount of timer steps (about 500 steps or 50 seconds).
A round is finished when:

- There are no rabbits left.
- Countdown timer reached zero value.

There are fixed amount of rabbits and stones in each round during competition.
As mentioned above, it is necessary to collect more rabbits than opponent to win the round. The result of match is calculated from two rounds results. The second round played with swapped positions of snakes. If it is a draw after two rounds, the third round must be played and if it is still a draw, the winner will be determined ***randomly***.



A snake can observe game environment and receive information about:

1. Amount and positions of stones,
2. Amount and positions of rabbits,
3. Self and counterpart snake body coordinates (position of each chain),
4. Timer value,
5. Snake own state parameters: the head direction, movement status, anticipated head position and the score.

All these functions to receive information are thoroughly described in the API section.

As was pointed above, a snake never stops its movement except several cases:

- When a snake met a stone,
- When a snake reached an island border,
- When a snake tries to cross its body,
- When the opponent snake blocks your snake.

The snake program should calculate the next turn command. The possible commands are **Turn Left** or **Turn Right**. If there are no commands specified for the turn then the snake will remain moving forward in the same direction. The calculation time for one turn is limited. It is about 100ms. The snake keeps current direction if time period of the current turn is expired. It's better to synchronize your snake with timer using **waitNextTurn()** method.

Rabbits can occasionally move to neighboring cells. They can also climb on stones, jump over the snake or island border. In these cases snakes can't collect them.

3. Template Program and API manual

This section contains an example of a Java-program simulating the snake behavior as well as description of the class **Snake.** You can edit this file to create your own class. Please create the java source-code file with the name **SnakeNN.java**, where NN is the number of your seat (table). Your code should be placed inside the **run** method. Examples of using Snake-methods are emphasized as ***BOLD-Italic***.

Don't forget to specify your own team name in the **setName()** method !!!!

```
/** Template program
 * Snake1.java
 * @see Snake.class
 *
 * The name of the class and file should be the same as the team name
 * (see setName method below)
 */

public class Snake1 extends Snake {

 /** Creates a new instance of Snake1. Constructor must be empty */
 public Snake1() {
 }

public void run() { // main control routine

       int hx = getNextHeadX(); // get the next head position,
       int hy = getNextHeadY(); // if it is free to move here

       int s[][] = getStones(); // stones handling is unimplemented here

      while(true) { // forever loop
```

```
            waitNextTurn(); // synchronization (pause till next tick of timer)

            int r[][] = getRabbits(); // rabbits handling is unimplemented here

            double v=Math.random(); // random decision about the next step
            if(v > 0.7) turnLeft(); // to turn head left
            else
            if(v < 0.3) turnRight(); // to turn head right

            // to move forward just do nothing

          } // end of while loop

        }

        public String setName() {
            return "Snake 1"; // set team name here
        }
    }
```

Snake API manual

Class Snake

`java.lang.Object`
  └ **Snake**

All Implemented Interfaces:
java.lang.Runnable

---

public abstract class **Snake**
extends java.lang.Object
implements java.lang.Runnable

Snake API.
This is the base class for a snake implementation. During the match, it is executed in a separate thread for each snake. The user must inherit the Snake class for designing his/her own control program.

Field Summary	
Boolean	**ismoving** Snake move indicatior.
Method Summary	
int[][]	**getBody**() Returns the 2D array with self body coordinates.
int	**getDirection**() Returns the self head direction.
int	**getNextHeadX**() Returns the X coordinate of the head in the next tick. (However, if snake can't move, it will stay at current position)
int	**getNextHeadY**() Returns the Y coordinate of the head in the next tick. (However, if snake can't move, it will stay at current position)
int[][]	**getOpponent**() Returns the 2D array with opponent body coordinates.
int[][]	**getRabbits**() Returns the 2D array with coordinates of rabbits.

int[][]	**getStones**() Returns the 2D array with coordinates of stones.
int	**getTime**() Returns current value of the time counter.
int	**getScore**() Returns the number of rabbits collected by snake.
abstract void	**run**() This method should be implemented by the competitor.
abstract java.lang.String	**setName**() This method is to specify a team name.
void	**turnLeft**() This method turns the snake head to the left. It is possible to have only one turn (left or right) per tick
void	**turnRight**() This method turns the snake head to the right. It is possible to have only one turn (left or right) per tick
void	**waitNextTurn**() Wait until the next time step starts. This method could be used for synchronization with the game timer.

Field Detail

ismoving

`public volatile boolean` **`ismoving`**

Snake move indication. True, if snake has moved last turn, false otherwise

Constructor Detail

Snake

`public` **`Snake`**`()`

Method Detail

waitNextTurn

`public void` **`waitNextTurn`**`()`

Wait until the next time step starts.
This method could be used for synchronization with the game timer.

---

getTime

`public int` **`getTime`**`()`

This method returns current value of the game timer in ticks. Zero means the last tick.

Returns:

integer timer value

---

getScore

`public int` **`getScore`**`()`

This method returns the number of rabbits collected by snake.

Returns:

integer score value

---

getOpponent

`public final int[][]` **`getOpponent`**`()`

This method returns the 2D array with coordinates of the opponent body. The first dimension is the index of coordinate (0 – horizontal coordinate X, 1 – vertical coordinate Y). The second dimension is a value of the corresponding coordinate.

Example:

int[][] x = getOpponent();

int hx = x[0][0]; int hy = x[1][0]; // coordinates of the snake head
int len = x[0].length; // length of the snake
int tx = x[0][len-1]; int ty = [1][len-1]; // coordinates of the snake tail

---

getStones

`public final int[][] getStones()`

This method returns the 2D array with coordinates of stones. It has the same structure as for the **getOpponent**.
Example:

int stones[][] = getStones();
int lenstones = stones[0].length; // number of stones
int x= stones[0][i]; // X coord of the i-th stone
int y= stones[1][i]; // Y coord of the i-th stone

---

getRabbits

`public final int[][] getRabbits()`

This method returns the 2D array with coordinates of rabbits. It has the same structure as for the **getOpponent**.
Example:

int rabbits[][] = getRabbits();
int lenrabbits = rabbits[0].length; // the total number of rabbits including collected ones
int x=rabbits[0][i]; // x coord of the i-th rabbit
int y=rabbits[1][i]; // y coord of the i-th rabbit

NOTES:

1. Rabbits may move occasionally, so be prepared.
2. Array returned by this method holds the **total number** of rabbits including collected ones.
3. **Negative coordinate** value means that the rabbit have been **collected already**.

---

run

`public abstract void run()`

This method should be implemented by the competitor. It holds the main snake control routine. It must contain "forever loop" which is to manipulate the snake in real-time.

Specified by:

`run` in interface `java.lang.Runnable`

---

setName

`public abstract java.lang.String setName()`

This method is to specify a team name.
Please implement it as:

public String setName() {
    return "Team name"; **// specify your team name here**
}

---

turnLeft

`public final void turnLeft()`

This method turns snake head to the left.
It is possible to have only one turn (left or right) per tick

---

turnRight

`public final void turnRight()`

This method turns snake head to the right.
It is possible to have only one turn (left or right) per tick

---

getNextHeadX

`public final int getNextHeadX()`

Returns the X coordinate of the head in the next turn.
(However, if snake can't move, it will stay at current position)

---

getNextHeadY

```
public final int getNextHeadY()
```

Returns the Y coordinate of the head in the next turn.
(However, if snake can't move, it will stay at current position)

getBody

```
public final int[][] getBody()
```

Returns the 2D array with self body coordinates. 1st dimension is the index of coordinate (0 - X, 1 - Y).
2nd dimension is a value of corresponding coordinate.
Example:

```
int[][] x = getBody();

int hx = x[0][0]; int hy = [1][0]; // head coordinates
int len = x[0].length; // length of the snake
int tx = x[0][len-1]; int ty = [1][len-1]; // tail coordinates
```

getDirection

```
public final int getDirection()
```

Returns self head direction of the snake.
// 0 - north, 1 - east, 2 - south, 3 - west

4. Design and Debugging

All files are collected in the **java-challenge** directory. There is the Snakes Hunt package provided for design and debugging. The **java-challenge** folder includes also the following files needed:

1. **SnakeHunt.jar** – The main application module for design and debugging.
2. **Snake.class** – The base java class for snake implementation (see example).
3. **Snake1.java, Snake2.java** – Snake templates (examples).
4. **Snake.html** – Manual for Snake.class in javadoc format.

Important Remarks!

- The name of your class designed should be as ***SnakeNN***, where ***NN*** – the number of your team (number of your table). The Java-file name should be as ***SnakeNN***.java.
- Type **java –jar SnakeHunt.jar SnakeNN Snake1** in the command prompt to start the execution of your SnakeNN program.
- It is only allowed to use the following Java packages**: java.Math** and **java.util**
- It is possible to use and design only ***inner classes.*** **This means that your code submitted should be a single Java-file.**
- It is restricted to use potentially harmful and unsafe java methods (like accessing to operating environment, file system, etc). Each round will run under a security manager.

- You can use a standard input-output to debug your program. **But these operations must be removed when you submit your program for the tournament.**

5. Submission Results & Tournament

Submission

You should provide to the referee team with the ***SnakeNN.java*** file with the source code of your snake controlling program. Please copy this file to the **java-challenge-result** directory. ***Your code submitted should be a single Java-file***. All input-output operations like **System.out** must be removed!!

Tournament information

All teams will be randomly distributed to their start positions in the tournament net. Each team has to play 2 or 3 rounds with its opponent on each level of tournament to proceed to the next level of the tournament. The 4-teams tournament example is shown in the picture.

Team will be qualified for next round when:

- Cumulatively collected more rabbits than an opponent team, or
- Win 3rd round if there is a draw result after 2 rounds, or
- Randomly, after three rounds with the draw result.

Credits

- Idea by Prof. Alexander Vazhenin, University of Aizu (vazhenin@u-aizu.ac.jp),
- Gameplay design and development by Vazhenin Dmitry, University of Aizu, Post-graduate school (dmvazh@sparth.u-aizu.ac.jp).

Special thanks to Rentaro Yoshioka and Pierre-Alain Fayolle for testing.