合肥學院
HEFEI UNIVERSITY

Distributed Computing
Homework 1: Battleship Game

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://www.it-weise.de

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Overview

- Use sockets in `Java`
- Implement communication for a distributed game

Battleship

- Battleship [1, 2] is a simple game for two players

Battleship

- Battleship [1, 2] is a simple game for two players
- Usually played with pen and paper
  - Each player has a play field divided into a grid of cells drawn on her paper, hidden to the other player

Battleship

- Battleship [1, 2] is a simple game for two players
- Usually played with pen and paper
  - Each player has a play field divided into a grid of cells drawn on her paper, hidden to the other player
  - Hidden to the other player, she places ships of different sizes on her grid

Battleship

- Battleship [1, 2] is a simple game for two players
- Usually played with pen and paper
  - Each player has a play field divided into a grid of cells drawn on her paper, hidden to the other player
  - Hidden to the other player, she places ships of different sizes on her grid
  - Each player's grid has the same size, the same number of ships is placed

Battleship

- Battleship [1, 2] is a simple game for two players
- Usually played with pen and paper
  - Each player has a play field divided into a grid of cells drawn on her paper, hidden to the other player
  - Hidden to the other player, she places ships of different sizes on her grid
  - Each player's grid has the same size, the same number of ships is placed
  - Players take turns in "firing" on the other player's grid ("I shoot at cell (3,2)!")

Battleship

- Battleship [1, 2] is a simple game for two players
- Usually played with pen and paper
  - Each player has a play field divided into a grid of cells drawn on her paper, hidden to the other player
  - Hidden to the other player, she places ships of different sizes on her grid
  - Each player's grid has the same size, the same number of ships is placed
  - Players take turns in "firing" on the other player's grid ("I shoot at cell (3,2)!")
  - After each shot, the other player announces success or failure: "You hit one of my ships!" or "You missed."

Battleship

- Battleship [1, 2] is a simple game for two players
- Usually played with pen and paper
  - Each player has a play field divided into a grid of cells drawn on her paper, hidden to the other player
  - Hidden to the other player, she places ships of different sizes on her grid
  - Each player's grid has the same size, the same number of ships is placed
  - Players take turns in "firing" on the other player's grid ("I shoot at cell (3,2)!")
  - After each shot, the other player announces success or failure: "You hit one of my ships!" or "You missed."
  - Each cell can be fired at at most 1 time

Battleship

- Battleship [1, 2] is a simple game for two players
- Usually played with pen and paper
  - Each player has a play field divided into a grid of cells drawn on her paper, hidden to the other player
  - Hidden to the other player, she places ships of different sizes on her grid
  - Each player's grid has the same size, the same number of ships is placed
  - Players take turns in "firing" on the other player's grid ("I shoot at cell (3,2)!")
  - After each shot, the other player announces success or failure: "You hit one of my ships!" or "You missed."
  - Each cell can be fired at at most 1 time
  - If all cells of a ship have been hit, the ship sinks

Battleship

- Battleship [1, 2] is a simple game for two players
- Usually played with pen and paper
  - Each player has a play field divided into a grid of cells drawn on her paper, hidden to the other player
  - Hidden to the other player, she places ships of different sizes on her grid
  - Each player's grid has the same size, the same number of ships is placed
  - Players take turns in "firing" on the other player's grid ("I shoot at cell (3,2)!")
  - After each shot, the other player announces success or failure: "You hit one of my ships!" or "You missed."
  - Each cell can be fired at at most 1 time
  - If all cells of a ship have been hit, the ship sinks
  - If all ships of a player have been sunken, she loses

Distributed Battleship

- Instead of pad & pen, let's use computers

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in `Java` of all the necessary stuff except communication

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in `Java` of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`)

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in `Java` of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`)

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in Java of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`) by creating
  - events (`BattleshipModelEvent`)

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in Java of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`) by creating
  - events (`BattleshipModelEvent`) to which
  - listeners (`IBattleshipModelListener`) can register.

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in Java of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`) by creating
  - events (`BattleshipModelEvent`) to which
  - listeners (`IBattleshipModelListener`) can register.
  - A main (`Main`) program for starting the game

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in Java of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`) by creating
  - events (`BattleshipModelEvent`) to which
  - listeners (`IBattleshipModelListener`) can register.
  - A main (`Main`) program for starting the game
  - A skeleton class (`Communicator`) where you can introduce the communication.

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in Java of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`) by creating
  - events (`BattleshipModelEvent`) to which
  - listeners (`IBattleshipModelListener`) can register.
  - A main (`Main`) program for starting the game
  - A skeleton class (`Communicator`) where you can introduce the communication.
- You know how to do communication!

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in `Java` of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`) by creating
  - events (`BattleshipModelEvent`) to which
  - listeners (`IBattleshipModelListener`) can register.
  - A main (`Main`) program for starting the game
  - A skeleton class (`Communicator`) where you can introduce the communication.
- You know how to do communication: Use sockets [3–6]!

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in `Java` of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`) by creating
  - events (`BattleshipModelEvent`) to which
  - listeners (`IBattleshipModelListener`) can register.
  - A main (`Main`) program for starting the game
  - A skeleton class (`Communicator`) where you can introduce the communication.
- You know how to do communication: Use sockets [3–6]! (either TCP [7, 8] or UDP [7])

Distributed Battleship

- Instead of pad & pen, let's use computers
- 2 players in the network can play against each other
- I provide a *simple* implementation in Java of all the necessary stuff except communication:
  - A simple game engine (`BattleshipModel`) which updates
  - a GUI (`BattleshipView`) by creating
  - events (`BattleshipModelEvent`) to which
  - listeners (`IBattleshipModelListener`) can register.
  - A main (`Main`) program for starting the game
  - A skeleton class (`Communicator`) where you can introduce the communication.
- You know how to do communication: Use sockets [3–6]! (either TCP [7, 8] or UDP [7])
- You do not need to do (touch, read) anything else, only class `Communicator` !

Task

- Fill the class `Communicator` with code that allows us to play *Battleship* over a network connection!

Task

- Fill the class `Communicator` with code that allows us to play *Battleship* over a network connection!
- You can choose the protocol (e.g., TCP or UDP) you want to use and how the connection should work

Task

- Fill the class `Communicator` with code that allows us to play *Battleship* over a network connection!
- You can choose the protocol (e.g., TCP or UDP) you want to use and how the connection should work
- In the class, some parts are marked with `TODO`

Task

- Fill the class `Communicator` with code that allows us to play *Battleship* over a network connection!
- You can choose the protocol (e.g., TCP or UDP) you want to use and how the connection should work
- In the class, some parts are marked with `TODO`
- Read the comments on these parts and fill in code accordingly

Task

- Fill the class `Communicator` with code that allows us to play *Battleship* over a network connection!
- You can choose the protocol (e.g., TCP or UDP) you want to use and how the connection should work
- In the class, some parts are marked with `TODO`
- Read the comments on these parts and fill in code accordingly
- The connection must be established by code being added to the method `void start(int ownPort, String enemyHost, int enemyPort)` which receives the connection parameters from the GUI

Task

- Fill the class `Communicator` with code that allows us to play *Battleship* over a network connection!
- You can choose the protocol (e.g., TCP or UDP) you want to use and how the connection should work
- In the class, some parts are marked with `TODO`
- Read the comments on these parts and fill in code accordingly
- The connection must be established by code being added to the method `void start(int ownPort, String enemyHost, int enemyPort)` which receives the connection parameters from the GUI
- The message-receiving code should be put into the method `void run()`

Task

- Fill the class `Communicator` with code that allows us to play *Battleship* over a network connection!
- You can choose the protocol (e.g., TCP or UDP) you want to use and how the connection should work
- In the class, some parts are marked with `TODO`
- Read the comments on these parts and fill in code accordingly
- The connection must be established by code being added to the method `void start(int ownPort, String enemyHost, int enemyPort)` which receives the connection parameters from the GUI
- The message-receiving code should be put into the method `void run()`
- The other methods should be used/implemented according to the comments therein

Result

- Send me a zip archive named `hw01_[your_student_id].zip` (where `[your_student_id]` is replaced with your student id) containing the complete Eclipse project, including the `src` and `bin` folder with, with all the code of this project and your modifications of it.

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://www.it-weise.de

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. Jeffrey P. Hinebaugh. *A Board Game Education*. G – Reference, Information and Interdisciplinary Subjects Series. Lanham, MD, USA: Rowman & Littlefield Publishing Group, R&L Eduction, 2009. ISBN 1607092603 and 9781607092605. URL http://books.google.de/books?id=nvqw9vK0R8sC.
2. Brian C. Ladd and Christopher James Jenkins. *Introductory Programming with Simple Games – Using Java and the Freely Available Networked Game Engine*. New York, NY, USA: John Wiley & Sons Ltd., 2009. ISBN 0470212845 and 9780470212844. URL http://books.google.de/books?id=EMuxvH65pW8C.
3. *Standard for Information Technology – Portable Operating System Interface (POSIX)*, volume 1003.1,2004. Piscataway, NJ, USA: IEEE (Institute of Electrical and Electronics Engineers), 2004.
4. Lesson: All about sockets, 2009. URL http://docs.oracle.com/javase/tutorial/networking/sockets/.
5. Kenneth L. Calvert and Michael J. Donahoo. *TCP/IP Sockets in Java: Practical Guide for Programmers*. Morgan Kaufmann Practical Guides. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN 0123742552 and 9780123742551. URL http://books.google.de/books?id=lfHo7uMk7r4C.
6. Merlin Hughes, Michael Shoffner, and Derek Hamner. *Java Network Programming: A Complete Guide to Networking, Streams, and Distributed Computing*. Manning Pubs Co. Greenwich, CT, USA: Manning Publications Co., 1999. ISBN 188477749X and 9781884777493. URL http://books.google.de/books?id=xapQAAAAMAAJ.
7. Charles M. Kozierok. *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. San Francisco, CA, USA: No Starch Press, 2005. ISBN 159327047X and 9781593270476. URL http://books.google.de/books?id=Pm4RgYV2w4YC.
8. Douglas Comer. *Internetworking with TCP/IP: Principles, Protocols, and Architecture*. Upper Saddle River, NJ, USA: Prentice Hall International Inc., 2006. ISBN 0131876716 and 9780131876712. URL http://books.google.de/books?id=jonyuTASbWAC.