Home Exam – Software Engineering – D7032E – 2025

Teacher: Josef Hallberg, josef.hallberg@ltu.se, A3305

Instructions

- The home exam is an individual examination.
- The home exam is to be handed in by **Wednesday October 22, at 09.00**, please upload your answers in Canvas (in the Assignment 6 hand-in area) as a compressed file (preferably one of following: rar, tar, zip), containing a pdf file with answers to the written questions and any diagrams/pictures you may wish to include, and your code. Place all files in a folder named as your username before compressing it (it's easier for me to keep the hand-ins apart when I decompress them). If for some reason you can not use Canvas (should only be one or two people) you can email the Home Exam to me. Note that you can NOT email a zip file since the mail filters remove these, so use another compression format. Use subject "**D7032E: Home Exam**" so your hand-in won't get lost (I will send a reply by email within a few days if I've received it).
- Every page should have your full **name** (such that a singular page can be matched to you) and each page should be numbered.
- It should contain ***original*** work. You are not allowed to cheat, in any interpretation of the word. You are allowed to discuss questions with classmates, but you must provide your own answers. Additionally, you are allowed to use AI tools and use the material produced by these tools, however, ***what you hand in should represent your knowledge and understanding of the subject*.** That means, if AI tools are generating answers or code for you, it is your responsibility to study the material provided by an AI and **make sure you understand the generated material, understand design decisions that were made by the AI model, and are able to justify and motivate design decisions.**
- Your external references for your work should be referenced in the hand in text. All external references should be complete and included in a separate section at the end of the hand in.
- The language can be Swedish or English.
- The examiner reserves the right to refuse to grade a hand-in that does not use a correct/readable language. Remember to spellcheck your document before you submit it.
- Write in running text (i.e. not just bullets) – but be short, concrete and to the point!
- Use a 12 point text size in a readable font.
- It's fine to draw pictures by hand and scanning them, or take a photo of a drawing and include the picture; however make sure that the quality is good enough for the picture to be clear.
- Judgment will be based on the following questions:
  - Is the answer complete? (Does it actually answer the question?)
  - Is the answer technically correct? (Is the answer feasible?)
  - Are selections and decisions motivated? (Is the answer based on facts?)
  - Are references correctly included where needed and correctly? (Not just loose facts?)
- **To have part B assessed and graded you need to motivate your design in a one-to-one oral exam session. The purpose of the oral exam session is to assess whether the material you have handed in represent your understanding of the subject. This is to make sure you do not hand in AI generated material, or material produced by other people, that you do not yourself understand. The examiner reserves the right refuse assessing part B in cases where the material you have handed in does not represent your understanding of the subject.**

Total points: 26	
Grade	Required points
5	22p
4	18p
3	13p
U (Fail)	0-12p

Good luck! /Josef

Scenario: Rivals for Catan

Rivals for catan is a 2 player card-game. Players take turn building their principality and scoring victory points through settlements, cities, and heroes. The full original game consists of 180 cards, but the introductory game (which is currently implemented) only utilizes 94 cards: 49 center cards, 36 basic cards, 27 settlement/city expansions, and 9 event cards. In addition to introductory game there are a number of era that adds new cards and rules, and there are also at least two expansions that add further new cards and rules. Complete original rules along with other useful files as well as rules for expansions are available as PDFs in Canvas.

- Video of game instructions: https://www.youtube.com/watch?v=etiR7CwatF8&ab_channel=thesettlersofcatan

The designer, Mr. VajbCruncher, is very proud of himself because he managed to turn Rivals for Catan into a computer game on his own using AI tools (ChatGPT and Git CoPilot), and now he considers himself a professional developer with mad skillz. Now that he thinks he has done the major part of the development he just wants you to fix support for the different eras and the available expansions, but he think it will be an easy task now that he's done most of the work (☺ - Evil teacher laugh). Your role is to refactor the game according to best practices and structure the code in such a way that cards, rules, and game mechanics that come with different era and expansions can be added. That is, follow design principles and best practices to achieve modifiability, extensibility, and testability. You do **NOT** need to actually implement the additional era or expansions, just refactor the code for the introductory game.

Figure 1 – the principalities for the two players

Rules (note: the turn overview PDF describes the process well – available in canvas):

1. There are two players, either one local and one remote, or two remote players both connecting to a server.
2. There are 94 cards: 49 center cards, 36 basic cards, and 9 event cards
    a. Center cards consist of 24 region cards, 9 settlements, 7 cities, 9 roads
    b. The two principalities are organized as depicted in Figure 1.
    c. The remaining regions are assigned dice roll as follows and are then shuffled:
    Field: 3 and 1, Mountain: 4 and 2, Hill: 5 and 1, Forest: 6 and 2, Pasture: 6 and 5, Gold Field: 3 and 2
    d. Regions, settlements, cities, and roads become 4 separate stacks for use during the game.
    e. The basic cards consist of 9 action cards, and 27 settlement/city expansions. These are shuffled and divided into 4 equal draw stacks for use during the game
    f. The event cards stack is shuffled, and then the Yule card is moved to the 4th from the bottom
3. Each player draws 3 cards from a draw stack of their choice
4. The players take turns (random player starts the game):
    a. Roll the event die and the production die (If a brigand attacks, resolve the event before the production, otherwise start with production)
        i. Production: give both player the resource corresponding to the production die (0..3)
        ii. Event: Roll 1 = Brigand, 2=Trade, 3=Celebration, 4=Plentiful Harvest, 5&6=Event card
            1. Brigand: if you have more than 7 resources you lose all your wool and gold
            2. Trade: If you have the trade advantage, you receive 1 resource of your choice from your opponent
            3. Celebration: If you have the most skill points, you alone receive 1 resource of your choice. Otherwise, each player receives 1 resource of his/her choice
            4. Plentiful Harvest: Each player receives 1 resource of his/her choice
            5. Event card: draw a card from the event card stack and resolve the event
    b. Action phase: Play basic cards (either from hand or from the center stack), Trade
        i. Refer to card descriptions for basic cards, page 18-20, in the rulebook for behavior and cost
    c. Replenish your hand (choose which draw stack to draw from)
    d. Exchange a card from your hand (you may return one card to the bottom of a draw stack, and take one card from a draw stack of your choice, or pay 2 resources to choose 1 card from a stack)
5. The winner is announced when a player has 7 or more victory points at the end of his/her turn

Future modifications: (you do not need to implement these but create your design for modifiability and extensibility)

6. Support for different era (available in the core rulebook), and expansions (separate core rulebooks available as pdf in Canvas). Changes include (and may not be limited to) new cards, new game mechanics, and new rules, as well as a different victory condition.

Quality attributes:

7. Design with priority on Modifiability, Extensibility (requirement 6), and Testability.

Important note: As far as requirements go these are a bit abbreviated to avoid complexity. These requirements should not be used as an example of how to write requirements (unless it is to illustrate how to write poor requirements). When designing unit-tests in your application you are expected to include tests for the behavior and mechanics of the 94 cards available in the core rulebooks (several cards are duplicates or have similar functionality while being named differently).

Questions

Part A

1. Unit testing (3p, max 1 page)

Which requirement(s) (rules and requirements 1 – 5 on previous page) is/are currently not being fulfilled by the code (refer to the requirement number in your answer)? For each of the requirements that are not fulfilled answer: *(note that requirement 4-a-ii-5 include 9 different cards/requirements, and it is a similar case with requirement 4-b-i which include several basic cards)*

- If it is possible to test the requirement using JUnit without modifying the existing code, write what the JUnit assert (or appropriate code for testing it) for it would be.
- If it is not possible to test the requirement using JUnit without modifying the existing code, motivate why it is not.

2. Software Architecture design and refactoring (9p, max 2 pages excluding diagrams)

a. Identify shortcomings in the original design. Use SOLID principles and Booch's metrics in your reasoning (1p)

Consider the requirements (rules and requirements 1 – 5), and future changes (requirements 6). Refactor the code according to specified quality attributes (requirement 7). Create an architecture design and reason about design choices. The documentation should be sufficient for a developer to understand how to develop the code, know where functionalities are to be located, understand interfaces, understand relations between classes and modules, and understand communication flow. Use good software architecture design and coding best practices according to SOLID principles and Booch's metrics.

b. Reason about design choices in your **new** (refactored) design using SOLID and Booch (3p)
c. Reason about how quality attributes have been satisfied in the **new** design. What design choices did you make specifically to meet these quality attribute requirements. Divide your reasoning into separate sections per quality attribute (helps me assess your exam) (3p)
d. Motivate design patterns or choice not to use in your **new** design. If any, what purpose do they serve and how do they improve the design (2p)

Note: The oral exam-session will cover primarily your software architecture design. For this session you will be expected to answer questions about your design and motivate design choices. You are allowed to refer to diagrams and code provided in your hand-in. It may therefore be more important to produce supporting diagrams than lengthy texts, since motivations for design choices can be given verbally. You should be able to answer questions about your design and about design choices without having to look at the written text (but looking at diagrams is recommended).

Part B (page 4/5)

3. Quality Attributes, Design Patterns, Documentation, Re-engineering, Testing (14p)

Refactor the code so that it matches the design in question 2 (you may want to iterate question 2 once you have completed your refactoring to make sure the design documentation is up to date). The refactored code should adhere to the requirement (rules and requirements 1 – 5 on previous page). Things that are likely to change in the future, divided into quality attributes, are:

- Extensibility: Additional era and expansions, such as those described in the "Future modifications to the game" may be introduced in the future.
- Modifiability: The way network functionality is currently handled may be changed in the future. Network features in the future may be designed to make the server-client solution more flexible and robust, as well as easier to understand and work with. In addition, the potential game extension (requirement 6) adds new types cards, game mechanics, and rules, which will change how some operations are handled.
- Testability: In the future when changes are made to both implementation, game rules, and game modes of the game, it is important to have established a test suite and perhaps even coding guidelines to make sure that future changes can be properly tested. You only need to make tests for requirements 1-5, but note that some of these requirements refer to card behaviour mechanisms available in the core rulebook.

a. To what degree can the game be extended (extensibility) in the future (requirement 6)? (2p)
b. To what degree can it be modified (modifiability) in the future (requirement 6)? (3p)
c. To what degree is the code designed for testability? (1p)
d. To what degree is the code unit-tested (requirement 1-5) (2p)
e. To what degree does the code follow best practices (structure, standards, naming, etc.) (1p)
f. To what degree does the game correctly implement the functionalities of the game (2p)
g. To what degree are errors/exceptions handled and reported appropriately? (1p)
h. Is the code appropriately documented? (1p)
i. Is the code true to the design in question 2 (1p)

Please help Mr. VajbCruncher by re-engineering the code and create better code, which is easier to understand. There is no documentation other than the comments made inside the code and the requirements specified in this Home Exam on page 2. The code and official game rules for the core game as well as additional game extensions are available in Canvas

The source code is provided in the zip file in Canvas which contains Server.java, Card.java, Player.java, and OnlinePlayer.java for game functionality and logic. There is also a jar-file for json processing (gson.jar) and a manifest of all of the cards in the original game (cards.json). If all files are in the same folder:

- You can compile the game with "`javac -cp gson.jar:. Server.java`"
- You can run the server with "`java -cp gson.jar:. Server`"
  (will then for a remote player to connect)
- You can connect an "online" client player: "`java -cp gson.jar:. Server online`"

The server must be started before the client is started. The server waits for the online clients to connect before starting the game.

In the re-engineering of the code the server does not need to host a player and does not need to launch functionality for this. It is ok to distribute such functionalities to other classes or even to the online Clients. The essential part is that the general functionality remains the same.

Add unit-tests, which verifies that the game runs correctly (it should result in a pass or fail output), where appropriate. It is enough to create unit-tests for requirements for the introduction game (requirements 1-5 with support of the rulebook for behaviour of basic cards). The syntax for running the unit-test should also be clearly documented. Note that the implementation of the unit-tests should not interfere with the rest of the design.

If you are unfamiliar with Java you may re-engineer the code in another structured programming language. However, instructions need to be provided on how to compile and run the code on either a Windows or MacOS machine (including where to find the appropriate compiler if not provided by the OS by default).

It is not essential that the visual output when you run the program looks exactly the same. It is therefore ok to change how things are being printed etc.

Additional notes about evaluation

Note about evaluation for question 2: During the 15-minute oral exam it will be difficult to assess to what extent the student has been able to satisfy the quality attribute and therefore the assessment of this will be done in Question 3. Instead, the assessment of Question 2 will focus on the student's ability to correctly reason about and justify design choices that have been made to satisfy the quality attributes with arguments supported by best practices, SOLID principles, and Booch's metrics.

Note about question 3: Assessment of Extensibility overlaps slightly with assessment of Modifiability and assessment of Testability due to the nature of the Extensibility quality attribute. Therefore, the evaluation of extensibility will focus specifically on to which extent expansions can be integrated with the game without affecting existing (compiled) implementation. This includes being able to unit-test and integration test the extension prior to launching it (unit-tests may update when adding an extension).