DIT184 Software Analysis and Design

B.Sc. Software Engineering

June 7th 2019

Time:	08:30 - 12:30
Place:	Lindholmen
Teacher:	D. Stikkolorum, Prof. Dr. M. R. V. Chaudron
Max. score:	100
Grading scale:	G=55, VG=75
Exam aids:	dictionary

The examination consists of 2 parts:

1. Theory questions (T1)
2. 5 modelling problems (P1 – P5) related to the case of a single system
    - Some problems build on each other; it's recommended to consider them in order.

Practicalities

- Write clearly and preferably with a pen; You may use pencil for diagrams.
- Explain your answers clearly
- Start each question/problem on a new sheet. (Sub-problems, may be put on the same sheet). Only write on the front of each sheet.
- Label each sheet with:
    - your *anonymous code (provided by the attendant).*
    - the *problem number, i.e., P1, P2, T1, …*
- The total points for the assignments add up to 90. The exam grade (on a 100-point scale) is obtained by multiplying by 10/9.

Before handing in:

- sort your sheets in problem order, and
- enumerate sheets as 1,2,3,....,
- check that your anonymous code is written on each sheet.
- explain your answers so that the graders can understand it.
- **write down/memorize your anonymous code** (to check anonymous exam results later)

Good Luck!

T1 (20 (6+4+4+6))

1. Explain 3 purposes of the Analysis-discipline in software development. (hint: ***do not*** (only/just) mention the result/output)

2. Explain the difference between a *model* and a *diagram.*

3. Explain 2 ways in which models help in designing software systems.

4. This question is about the concept of ‘layering’ in software design.
    a. What is *layering* of software design? I.e. How should layering be applied to software design? (How does it guide the design?)
    b. Which benefits are achieved by a proper layering of software design?

Online Tic Tac Toe

Tic Tac Toe (figure 1) is a very popular ‘pen and paper’ game. It requires two players, a piece of paper and a pen. First, a 3x3 grid is drawn. Then players choose if they want to play with an ‘x’ or an ‘o’. Subsequently, they take turns and draw an ‘x’ or an ‘o’ on the grid. A player wins when (s)he is the first to have 3 of his symbols in a row (vertical, horizontal or diagonal)

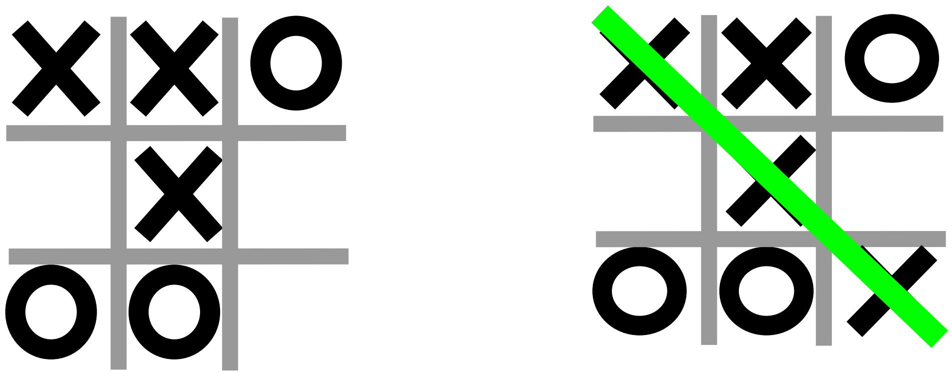


Figure 1: snapshots of a tic tac toe game

Mobile Tic Tac Toe app
You are asked to develop a mobile app version of the Tic Tac Toe game (**MTTT**).

With **MTTT** you are able to:

- Play a game against an AI player (computer player with artificial intelligence)
- Play against online players
- Play in an online tournament
- Replay games for analysis
- Register for the app (mandatory, needed for finding players online)

Based on the text below you are asked to first analyse (by modeling) the problem (domain) and then design (by modeling) the software.

Registration
Players should register a profile in order to use the app. An external online system holds the collection of player profiles. The profile consists of the following data:

- Nickname
- Age
- Country
- Email address
- Password
- XP (experience points, is 0 when a player is new)

When **MTTT** runs for the first time, a player is asked to fill in the registration.

Playing online
When registered, players can play against an online opponent. The opponent is chosen based on XP. Winning online games increases XP. XP decreases after 2 lost games.

Playing against the AI player
Playing against an AI player does not increase XP, but gives a player the opportunity to practice. The player can choose the difficulty of the AI player by selecting a certain value of XP of the AI player. During the game, the AI player can use certain strategies (algorithms) for deciding which move to play. One is called 'Monte Carlo'[1] the other one is called 'Book of Moves'[2]. The last one needs to access an online moves database.

Playing in a tournament, watching the ranking
A player can join an online tournament. Tournaments are organized at a certain date and time. They also have an end date. Again, players are matched to an opponent based on their XP levels. At the end of a tournament, a ranking (top 25 players) is published. The ranking of tournaments can be viewed from **MTTT**'s main menu.

Replay games
Players can replay their online games. The system remembers the last 10 games. The player is able to replay the game step by step. The replay data can be used for research and extending the book of moves.

NB: All account profiles, played games, tournament scores etc. are stored in an external system: the **MTTT** server that is connected to the internet.

[1] The Monte Carlo algorithm decides the next move by first simulating a high number of games based on the current state of the game.
[2] Using a book of moves gives the AI a collection of possible moves based on the current game state.

System Requirements

For the development of the **MTTT** system some requirements are written down:

<u>Application (system) requirements</u>

- In the future, the system:
  - should support the players in chosing other symbols instead of ‘x’ and ‘o’ and allow the player to choose colours for the symbols.
  - should support more types of algorithms for the AI player.
- The application uses an attractive graphical user interface.
- In general, the software should be maintainable, extensible and changeable.

<u>Hardware/infrastructure requirements</u>

- The system runs as an app on a mobile phone
- The system is connected to the internet

<u>Technical software requirements</u>

- The programming language for implementing the system is Java (an object-oriented language)
- The system uses a couple of predefined software classes:
  - **MTTTCommunicationController**, is responsible for handling the communication (sending method: *void sendMessage(messagedata)* and receiving method: *messageData receiveMessage()*) with the external **MTTT** server that stores profiles etc.
  - **Canvas**, the GUI module that is responsible for showing the graphical components of the application (such as text input, touch-buttons and text-messages).
  - The **Canvas** derives from **GraphicalComponent**, a reusable class that is responsible for handling events for graphical components.
  - The **Canvas** consists of several implementations of specialisations of **GraphicalComponent**:
    - **TouchButton**
    - **MessageArea**
    - **TextInput**
  - There should be derived classes of **GraphicalComponent** that are the implementation of the 3x3 grid and player symbols (‘x’ and ‘o’)
  - There should be a class that has the role of a central control unit.

    The remainder of the design is left to the developer and must be obtained by analysis (such as: games, registrations, players, profiles etc.). It could be that some of the classes above should be improved by extension/modification to fit the software design.

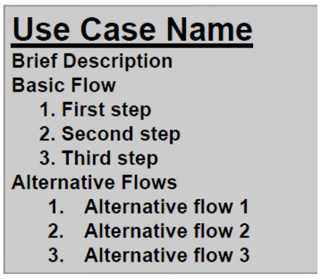
The following questions relate to the MTTT case.

P1. Use Case based requirements (15)

a) Make a use case diagram and include at least two sensible use cases in addition to the use case '*Register Player*' and *'Play Game'*. The use case '*Register Player'* can also be initiated from '*Play Game*'.

Next to the main use cases, the answer should at least use one include- or extend- use case.

b) Give a use case description, including 1 alternative flow, of '*Play Game*', following the standard notation (see figure).



Use Case Name
Brief Description
Basic Flow
1. **First step**
2. **Second step**
3. **Third step**

Alternative Flows
1. **Alternative flow 1**
2. **Alternative flow 2**
3. **Alternative flow 3**

P2. Problem Domain Modelling (15)

Build a domain model of the problem domain presented in the description on pages 2-3 through the use of a UML class diagram. You should consider using the richness of UML (such as association names, composition- , aggregation-, inheritance relationships and multiplicities) where appropriate.

P3. Design Modelling (20)

A developer of a software development team identified classes with the use of the following CRC (Class Responsibility Collaboration) cards:

PlayerProfile	
Responsibilities	*Collaborations*
Holds the player's personal information and XP credits	Game Tournament ....

AI Player	
Responsibilities	*Collaborations*
Control the opponent logics and chooses strategies	Game Strategy

Design the initial software structure for the **MTTT** system, with the use of **1) a UML class diagram and 2) a sequence diagram.** The design-level class diagram addresses the implementation aspects. The design should consider the requirements on page 3 and the classes presented by the CRC cards above.
For the full score, all classes, attributes, methods, and associations, association names, cardinalities, and inheritance relationships should be defined. Also, coupling and cohesion are considered for the score.
Procedure: Create 1 design class model by first constructing 2 sequence diagrams (or the other way around) for 1) *'Register Player*' for a date range and 2) 'Play Game Against AI*'*. **The answer should consist of your sequence- and class diagram.** The diagrams should be consistent.

P4. Design Principles and Patterns (10)

a) Organize your design (class diagram P3) in a layered architecture. Draw the classes without their details (attributes/operations). Use proper names for the layers. After introducing layering, it is possible that design principles are violated. Mention which principles are violated and how this should be improved in a next generation of the design. Explain your answer (don't alter the design).

b) Explain how a design pattern can support the application performance and/or usability in the case of handling the registration of a new player while communicating with the external MTTT (database) system.

P5. Behaviour Modelling (10)

The **Game** object holds the state of the game. It can be in different states: CREATED - PLAYING - PAUSED and FINISHED. When a player presses 'new game' a new game is created (CREATED). When a second player is chosen the game starts (PLAYING). While playing the time and the active player are displayed. After a players move the state will be PLAYING again and the status of a player's turn is updated (turn = true or turn = false). After every move, there is a check for the end of the game (a player wins or there is a draw). During play, a player can only pause (PAUSED) a game when it is his/her turn with the use of a 'pause' button. The game resumes when the button is pressed again or after 2 minutes. A game is in the FINISHED state when an end-of-game situation is reached.

To model the behaviour:
1) consider the game as an object.
2) describe the behaviour of the game by using a UML State Machine Diagram.
The state diagram should consist of proper internal (do/) activities, transition triggers and transition effects. It should involve method calls to the relevant objects of the class design of P3. It should be consistent with the answer to P3.