Verklegt námskeið 1 – T-113-VLN1 – Haustönn 2025

Software for e-Sport event - RU's e-Sport Extravaganza

Introduction and goals

We here at Reykjavik University have taken inspiration from the good old days of Roman times and decided that the populous (i.e., students) need a distraction after all their hard work. Due to the fact that times have indeed changed since the Roman empire it would not be suitable to put on gladiatorial games of old. We can however do the second best thing, organize an e-Sport tournament called "RU's Glorious e-Sport Extravaganza". There are grand plans for turning the Sun into the Shangri-La of all LAN-parties. The problem, and this is where you can help, is that we do not have the software necessary to organize and managed this glorious event.

The goals of this course are many. First and foremost the aim is to give students an opportunity to hone the skills they have picked up during the first term's 12 week courses, especially Programming 1. Other important goals are to learn and practices working as a group towards bigger, more complex, task; and doing so in a structured and organized way. Students will learn to follow "best practices" such as using Git (have to make a handbook for your own use), following a design pattern (3-layer design), using encapsulation and wrappers; working with multiple classes, in multiple source files, reading and writing data to and from text-based files (also how to map in-memory data relations to and from files). Last but not least, the whole process must be documented as a large part of the grading is the design report that must be handed in with the code-base and other material. **It should be stated from the get-go that it is expected that the program should be user friendly, not crash on incorrect inputs from users, etc. We strongly encourage actual user testing as early and as often as possible.**

Project Description

The software that is to be created is intended to manage an e-sport event and therefore the program needs to be able to record and display information about players, teams, matches and tournaments. Minimum requirements are all A requirements, listed in more detail below, and in order to pass the course you must meet all minimum requirements. There are various additions to the minimum requirements and they are listed in a separate sections called B and C requirements. Before starting on any C requirements, all B requirements should be completed first (unless you have specifically argued and gotten permission from the main teacher to do otherwise). It is advisable to such agreements writing and state in your design report why such changes were justified.

Finally there is the section regarding C requirements. As we have written it, those are more suggestions and/or ideas on cool things to add to the software but you are free to propose your own. I

also want to remind you that it is perfectly OK to have requirements in the design report for type B and especially type C that do not finally get implemented in the code base. Think of the A's as **"must have"**, B's as "Should have if we can" and the C's as "Nice to have but not needed for the software to be functional".

However, **what ever requirements you claim to have implemented should then be fully implemented, tested and it should work.** We will NOT give any credit, and in fact it may harm your grade, if you have half completed and/or broken functionality in the code that is handed-in for grading. In such cases it is better to remove that functionality from the code. We especially get upset if the broken functionality is of a higher priority than some functioning requirements. Bottom line is **always do the A's first!**

Your system should be designed and implemented using a 3-tier design, where the top layer deals with user interaction (UI-layer), the middle layer enforces the rules (Logic-layer) and the bottom layer deals with mapping, storing and retrieving data from storage (Data-layer).[1] Each layer has it own role and they should be "well separated" as we give very harsh penalties for all violations of the 3-layer principles.[2] Between each layer you should define interfaces and we recommend the use of wrappers to hide how each layer is implemented (more on this in lectures).

The program shall have a Text-based User Interface (TUI) and the programming language used is Python. Note however that **we require that you always use type hinting for declaration of variables and functions**. Also keep in mind that you will be using object oriented programming but we also define a difference between functional classes and model classes (classes specifically designed to hold data). This part of the 3-tier design principles and will be explained in lectures.

The data layer shall be implemented in text-based files (*.txt*, *.csv*, *.json*. You are NOT free to import any library you want into your Python code be we have allowed, if asked nicely, to use libraries that help students pars the more advanced text-based file formats. Just make sure you get approval first before you import any libaries into your code base.

Students will be expected to creat and use the following resources:

- Prioritized (A, B, C) **list of requirements** both for system and UI, with functional and non-functional requirements (isl. Kröfulista til kerfis og viðmóts, með virknikröfum, nytsemis og notendaupplifunarkröfum).
- **Use cases** for the requirements.
- Tests or method of testing requirements (when applicable).
- **Class diagram(s)**. Can be good to split them: One for Model classes, another for 3-tier design.
- **State diagrams**, as need.
- **User group analysis** (isl. Notendahópagreiningu).
- **UI-design**, including a Wire-frame where you show "happy paths".[3]

We will primarily look at the design documents and see if students have actually implemented what the set out to do. Thus, **it is very important to update your design documents** as we will be checking them during grading.

[1] See lecture 1 on Canvas for more details on 3-tier design.

[2] Example of layer violations: We do not want to find any print statements in code that is not in the UI layer and UI and logic should not even know the data is stored in text-based files in the data-layer.

[3] We highly recommend testing the early prototype UI before you start implementing. It is much easier to fix any issues early in the design phase then re-writing all the code later.

Deliverables, Evaluation and Grading

Git and the Diary: Each group must use a version control utility (i.e., Git) and the group MUST keep a diary that is committed to Git every day. Every day the group should have a meeting to update the diary and after that meeting a new diary entry should be committed to Git with the following information:

- Date and time and who was in attendance (on-site and/or remote).
- A summary of "State of the project" since last meeting. This can include stuff like listing completed tasks, list any issues discovered and/or any other issues that are worth discussing with the whole group.
- "ToDo-lists": Depending on how the group is organizing (one person or working in pairs), there should be one ToDo-list of task to be worked on today for each person/subgroup. ToDo-lists should be prioritized and the goal is to keep everyone working on what is needed next and so everyone knows what task to do. Make the list to long rather than to short as it is fine to not finish the list every time. Next meeting you just update them by adding task and changing priority.
- "State of the group": Use this section to list comments, disagreements, summarize you meeting discussion and write down questions for the teacher and/or TA's. This is your chance to document how the group work is going overall (remember you hand in the Diary and it is used in grading).

Grading: Developing the software is a group task, should be done as a group, the grade given for the deliverables is assigned to the group as a whole. This applise to all of the following deliverables unless stated otherwise.

- **Functionality and quality of the software and code 40%**
  We will not hesitate to penalizes strange, un-intuitive or buggy functionality. We expect user-friendly, crash resistant UI as the intended users may not always be computer experts. We would also like to remind you that we will not give credit, and in fact we may punish, functioning "fancy stuff" (C's) if the mandatory functionality is missing (A's). It is better to remove from the code unfinished functionality than to leave in half finished code. Included here is also code quality with regard to function (i.e., not cosmetic).
- **Design report and Dirary 30%**
  We can not emphasize too strongly that the design should align with the implementation. Thus, as often happens, the plans may change when they are implemented but then you MUST update the design report.
- **Nice structure and code cosmetics accounts for 10%**
  This includes: consistency, file structure, indentation, commenting and overall style quality. YES, we will deduct for lack of and/or poor comments. Focus on explaining *WHY* not *WHAT*.
- **Instructional video accounts for 10%**
  One of the last deliverables created is a video that students create to help new users learn how to use the system they have created. This video must not exceed 10 min. in length. The goal is not to show every functionality, so focus on the most common tasks that the user is most likely to need the most.
- **Self and Peer evaluation is 10%**
  We will both use diary entries as well as have a self and peer evaluation process.
- **Special deductions 3-tier violations**
  We just want to give a clear and early warning that ALL violations of the 3-tier design principles will be severely punished (up to 10% for each infraction). Please take extra care to not brake the rules and you must ask for Teacher/TA's guidance if you have any suspicion that you may be in violation of the rules.

Project Organization

Students will be assigned into 4-6 person groups and each group will be assigned a table in the Sun (or at HA). We will also use a Discord server for interactions both within each group and as a queue for assistance. Thus, it is important that all students join Discord, be assigned to their group (role on Discord) and please use it for communication (like if you are sick, join the queue for help and to leave messages / status reports between teammates). **Students are expected to focus fully on this course. We do not have many lectures. Instead you work on the project as a team and much of the teaching will be in the form of assistance as you work on the project. It is therefore expected that each student puts in a full days work, every working day, for the full 3 weeks.**

Note for 1st Day! The first tasks of the team should be to 1) Hold the first team meeting; 2) Document the team meeting in the first entry to your Diary (part of grading); 3) In the team meeting prepare a schedule (when each person will be working), discuss your individual expectation from the course, what effort you are willing to put in and make a team-contract where you come to a common agreement on how to organize the team effort in the coming 3 weeks. 4) Nominate a "Chairman/Leader" who is responsible for making the Git repository and the first Diary entry Git commit (all Diary entries at the end of every team meeting must be committed to Git – Instructors may ask to see the Diary at any time).

The few lectures we will have will be held in a class room but will also be live-streamed on Discord. Please make sure at least one member of the team shows up to class. We will also record the lectures and put recordings of them on Echo 360 and links in the Canvas->Modules.

As stated earlier, communication will be done via Discord. Each groups has their own space with both video and text channels that only they and instructors can see. There is also a special queue channel to seek assistance. For messages that do not need urgent attention (like letting know that you are sick today or that you have a non-urgent question) please use @-messages in the teams text-channel (so we know who you are and what group you are in). *For more sensitive messaging you can use PM's directly to the teacher but remember to state your full name and group as in the PM's we can't see who you are – PM's will not get high priority. If it is really urgent, you can call (number is on first lecture's slides).*

You will need a Git for version control, not only for your code but also the Diary and design documents etc. Thus one of your first tasks (day 1) is to setup a Git repository. One of the team members should be nominated the Git expert and he should make and handbook for the rest of the team. Please find a description of that task on Canvas.

Important dates:

- Monday Nov. 24th. Day 1 – Lecture + first team meeting.
- Friday Nov. 28th. First hand-in: The design report (pdf) and first team/self evaluation (quiz) is due before midnight. Both are handed in on Canvas.
- Monday Dec. 1st. Lecture + first day of coding.
- Tuesday Dec. 2nd. Feedback on the design report should be back.
- Friday Dec. 5th. Second hand-in: Second team/self evaluation is due before midnight.
- Friday Dec. 12th. Last team/self evaluation + **All finale deliverables are due:**
  - Updated *Design Report* + Copy of the *Diary* + *Git-Manual*.
  - Code, Readme and list a requirements.txt (for pip) – *all in one .zip file.*
  - Link to the *Instructional video for new users* – No longer then 10 min explaining the main functionality to new users.

Lectures

This course is designed to do most of the teaching in an *on-demand basis*, as the students solve task at hand. There will therefore be only a few lectures scheduled. Other material will be made available and can, and should, be used when the group finds that they need it. **Please take some time to look at what material we have made available on Canvas and visit it back when you need it**.

Scheduled lectures:

- Monday Nov. 24th. Lecture on Day 1. Here we will outline the course, the project and 3-tier design etc.
- Tuesday Nov. 25th. Lecture on Day 2. We will go over UML: Do's and Don'ts.
- Monday Dec. 1st. Lecture on Day 6. Start of coding and good advice + general feedback on design report.
- Tuesday Dec. 2nd. Lecture on Day 7. Good advice on Python Coding and some coding examples.
- We may add more lectures if needed and/or online material. This will be advertised on Discord.

A Requirements – The minimal requirements to pass the course

The software that you have been tasked to create is a bookkeeping system for hosting and managing an e-sport tournament event. The software does NOT need to be real-time, as competition games play out, but rather we need to be able to revive result sheets and users need to register the results and our software will then update the standings and statistics accordingly.

This will require team captains to have access to the system such that they can register their team. Each team may have 3 to 5 players and for each player we must register their name, date of birth, home address, phone number, handle (unique nickname used during play). Each team must have a unique name, a captain (one of the players in the team) but can also have optional fields like a URL to the teams website.

The teams will then compete in tournaments and as a minimum the software must schedule a knock-out tournament of at least 16 teams. Tournaments are hosted at a specific venue (location), at a specific time (can be multiple days) and competition can take a few days (say one week-end).

Once the players, teams, and tournament data has been registered the software should organize the competition. Each match should have a date, time and serverID (where the match is played) and of course the same team can not play two different matches at the same time. For A requirements you can assume that just a single game is played in each tournament. Each tournament may have a limited number of servers and thus the schedule has a limit on how many matches can be played at the same time. One of the necessary requirements is to print out the match schedule such that it can be made available and/or posted on-site.

Once the matches have been played the results need to be registered into the system by the tournament organizer(s). Anyone should be able access the system to see the current standings and statistics on any tournament (past, future or present), as well as the schedule a given on-going or upcoming tournament that has bee fully registered.

The following as a more formal list of requirements from the customer.
Note however that this may not be the full list. You must add any requirement that you identify as also needed from the description and/or your own discovery in the design phase. If you are unsure you should ask us (the instructors) to clarify for you).

Players – Each player should have: name, date-of-birth, address, phone, e-mail, link (url or social media), handle (unique name used during play) and what team he is in. *For A-requirements you can safely assume players are only in one team.*

Teams – Each team should have a unique name and a captain (that must be one of the players in that team). Other data can be registered, like ASCII-art logo, link to web-page etc. Captains should be able to add, change or modify players in his team. *For A-requirements you can safely assume players will not change and the team is just created once. It is however possible to change phone numbers, addresses etc. and this is then done by the captain of the team.*

Organizer For A requirements you can assume that there is just one admin account that can register new teams, organize tournaments. Further more, you can assume there is just one type of game played in each tournament.

Tournament – Organizers need to be able to create a new tournament. Each tournament should have a start date, an end date, unique name, a venue (location) and a registered contact person with e-mail and phone number. All games must be scheduled after the start-date and before the end-date.

The tournament is a knock-out style competition and the organizer should be able to command the system to create the game schedule once all the data has be registered. *For A-requirements you can safely assume that once the schedule has been generated it will not be altered and no further changes to teams nor players will be needed.*

Games and Results – As the games are played the results will come in. The organizer needs to be able to enter the results into the system and the system. Once results have been added it should be possible to create updated game schedule (showing results in completed matches and updated info for incomplete matches).

Information retrieval – Anyone should be able to use the system to get the game schedule. Completed games should have the results, while incomplete matches should show who is playing (or will) and when they are scheduled for. Further more, anyone should be able to call up information on each team, what players it has and what tournaments teams/players have participated in. *Note! that everyone should only see team names and the player handles, not the personal information of each player. Only the organizers and team captains should see the personal stuff.*

Your task is now to make the full requirement list for A-requirements and find any issues and or unknowns and then ask the instructors for further guidance.

B Requirements

The following are additional requirements. As before, this list is not exhaustive nor final. Your task is to further refine this text into a detailed list of requirements. You and your group should always ask the instructors for further guidance if, and as, needed.

Allowing a more complex tournament style. For example, it is common to have tournaments with with *Double-Elimination* a.k.a. a *Looser's bracket*. More information can be found on (Wikipedia: Double-elimination tournament).

One of the main features missing so far is that multiple computer games, let us call them "events", are played during the same tournament. There are a few ways handle multiple events but the **added requirement is that now the same team can sign up to compete in multiple events**. The scheduling must now take this into account when it is organizing the matches in both events.

Clubs – Another feature to be added is that each team should belong to a club. The club is like an umbrella term, that has a name, club colors, home town, country.

Tournament Rewards – At top tournaments there are prices for the best performers. While there is no money offered at RU's tournaments, teams and players can earn points for doing well. Added requirements is to keep track of such rewards at both the player, team and club level.

Statistics and history One may think that sports is all about the games, skills and completion.. One would then be very wrong. Sports are the grand buffet for people that love statistics. **New requirements for showing statistics for players, teams and clubs are as follows:**

- Player – When viewing player's information it should be possible to see (or ask to see) player statistics like (wins, earnings, clubs played for, teams played on, tournaments participated in etc.).
- Team – When viewing a team's information it should be possible to see (or ask to see), won tournaments, sum of all wins, sum of all earnings, ask to list all players, show player list sorted by wins/points/earning etc.
- Club – When viewing a Club's information it should be possible to see (or ask to see) all teams and it should be possible to sort the team list by earnings, wins and number of tournaments participated in etc.
- *Please note that statistics for teams and clubs is quite tricky if we allow players to change teams. Take this into account when/if you decide to support this feature.*

C Requirements - Following are ideas for you to consider

You are however welcome to come up with your own ideas for C-requirements. Remember that you do not need to actually implement all/any of them. It is often good to write the down as even if they are not implemented now, one can design the rest such that it is easier to add them later. Here are some ideas for you to work with:

- While teams may show up for multiple tournaments, they do not always have the same players every time. In fact players can change teams. An added requirement could thus be that a new team of players should be registered for each tournament. *NOTE! this will severely complicate that statistic calculations.*
- Be able to process result files instead of having the organizers type in the results from games by hand.

Remember to first completer all the A and most, if not all, the B requirements first. Also, it is better to remove all code for an unfinished/crashing requirement from the final deliverable (but keep them in the design report), as we will punish poor functionality.