1. General description

This program is question answer game. Game can have 2 to 4 players. Main idea is answer right at questions and collect points. Players get points every time they answer right. Wrong answer does not take points away. Winner is the player who has most points collected when questions end. There is one question file already in program that players can use or then players can add their own question file if they want to use modified questions. Format of question file must be the specific format that program can parse it into questions. This format is described later this document. Question file needs to be the same folder where the code is so that it can be used on game. Game has only command line user interface at this point. Program is not based any previous work and it has been implemented during period 4.

How to run the program (NOTE! Make sure you have python 3.x version):

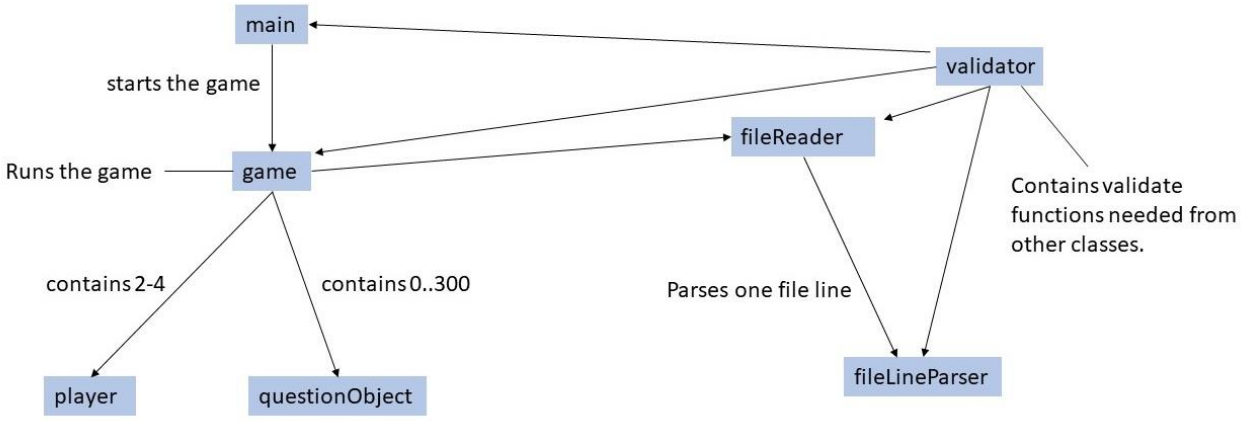
- You can start program running the main.py file on pyChram editor. Program is tested to work on school environment with pyChram editor.
- You can run the program on command line by navigating a dictionary where you have the codes and writing: python main.py On school environment some computers may have Python 2.7.5 version, like for example linux desktops. Command line cannot read then user name inputs and NameError will arise. This is because input() function has been renamed on python 3.x version and it had different name lower python versions. Pychram editor can still for some reason run the program.

2. Structure of program

In Table 1 there is presented all classes that are used when program is running to make sure that game can be played. Table 1 contains class names and short description of each class. After that there is Picture 1 which describes how these classes are working together. Table 2 contains name and description of implemented unit test classes and question file description.

Table 1: Classes that are needed to run a program.

Class/file name	Description
fileLineParser	Parses one file line and creates questionObject from it.
fileReader	Reads the question file and calls fileLineParser to parse file lines one by one.
questionObject	Contains all information for one question.
game	Game contains players, questionObjects and state of the game. This class runs the game.
main	Starts the program and collects player information and question file information from user and starts game after that.
player	Contains all information for one player.
validator	Contains all validate functions which are needed to validate user inputs on this program.



Picture 1: Structure of the program.

Table 2: Unit test classes and question file.

File name	Description
readyQuestions	File containing ready questions for the game. User can input own question file also.
unitTestsForFileLineParser	File containing all necessary unit tests for fileLineParser.
unitTestsForQuestionObject	File containing all necessary unit tests for question object.
unitTestsForPlayerObject	File containing all necessary unit tests for player object.
unitTestsForValidator	File containing all necessary unit tests for validator.

Following question file line is correct format for question file that program can read: 1. question: Who discovered penicillin? answer_options: [a. Alexander Fleming][b. test answer 1][c. test answer 2] correct_answer: [a]

First there is a question number and then “question:” tag with question and then “answer_options:” tag with three different answer options and in the end there is “correct_answer:” tag with correct answer inside brackets. Question file is also accepted if all or some answer options are empty, for example: 1. question: Who discovered penicillin? answer_options: [a. Alexander Fleming][b.][c. test answer 2] correct_answer: [a] would be totally fine question file line. There still needs to be brackets and letter a, b or c and dot inside the brackets that line is accepted. If the question file has some problems program will inform user where the problem was when it tries to read a file.

3. Secure programming solutions

During the code input validation has been made as secure as possible. That was also the main point of this programming task that input validation is as secure as possible. Input validation is important so that program can be sure that only proper data can enter to the system and other parts of the program. If inproper data can have access to program it will cause security vulnerabilities. In program input validation is made right after user tries to input data in program so that wrong kind of data will be caught as early as

possible. Input validation has been implemented using different validation functions from implemented validator.py file. Program uses regexs, allowed character lists, correct data type checks and maximum character amounts to make sure that only proper data will be inputted to program.

Proper error handling is implemented to inform user when he tries to input data that is not allowed or the file user tries to input to the program is not valid. Program does not crash if input contains some data which is not allowed and it tells user what went wrong and asks user to try again. Program uses regex and valid character list to make sure that only specific characters are allowed and that it is not possible to try to input some commands that may cause unexcepted behavior inside program.

Because we cannot never trust that some one would not modified the original question file, that file is parsed and validated every time when game starts. Users' own file is parsed and validated before questions are entered to the program. The limit of length of question file is now 300 lines. Line amount is limited so that there will not arise memory leaks or buffer overflow vulnerabilities.

Program cannot rise race condition because program is not running in multiple threads and any part of the program is not depending on the timing some other parts in program. Program goes smoothly to one state to another. Program is implemented clear parts so that it is more easily to maintain and that is how it also raises programs' security. Security solutions are also commented in code with tag: SECURE_PROGRAMMING_SOLUTION.

4. Security testing

There is included some unit tests on code. Classes that have unit tests: validator, questionObject, player and fileLineParser functions. All these tests are passed, and they are comprehensive. Total number of unit test implemented is 43. Unit tests can be run on pyChram editor by clicking the green arrow (inside test file) next to test name that is wanted to run. Other parts of program and its' functionality were tested manually.

Manual tests included testing main functionality for the program and that correct error messages raised when needed. Main functionality tests included playing game as a user would have play the game and inputting all kind of wrong data during the game was running. There were also tests that tested correctness of question file and error messages which raises from there. Regexs are only manually tested using valid and invalid inputs during the time they were created. There were some problems that came when testing regexs but they are all corrected and now regexs are working as they should. Regexs are used inside some validator functions which have unit tests so partly of them are tested via those tests.

Parts of error handling was fixed during manual testing, because it seemed that error messages weren't very clear or the was no proper error message raised when something unwanted happened. There were also some problems on validator functions which were founded when writing unit tests on validator functions and those problems were fixed.

5. Suggestions for improvement

Suggestions for improvement, what could be implemented:

- Pause, restart and reset game features when game has already started.

- Graphical user interface. Now user interface is only command line user interface.
- Unit test for regexs. Now they have only tested manually because there were some problems when I tried to write unit tests for them. But they are quite critical part of program functionality so it would be good to have unit test them also.
- For some reason unit test reports some chracter erros when trying to run them on command line but they work correctly when running on pyChram editor. Error was: “SyntaxError: Non-ASCII character '\xc2'” trying encoding for some reason did not help. This should be fixed later.