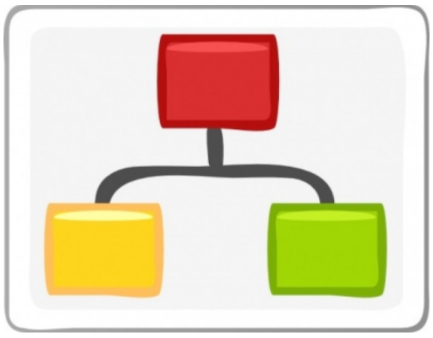

Basics of Computer Programming 2020/2021

# Project instruction - part 1

## Problem 1 – General Friday the 13th game

Author: Tomasz Goluch

Translation: Tytus Pikies and Marcin Jurkiewicz

Version: 1.4

## I. Game description: Friday the 13th

*Goal: familiarize yourself with rules of the game, which is a basis of the first problem of the project.*

You can download the rules of the orginal game (by Reinera Knizia) here.

In the original game we have 50 numbered cards. Each card has a color and a value. In the game 8 cards are green (poison cards) with value 4 and three colors (blue, red and violet) each consisting of 14 cards with values: 1, 1, 1, 2, 2, 2, 4, 4, 5, 5, 5, 7, 7, 7. Originally, the players alternately place any chosen card from their hand into 3 cauldrons (piles of cards), starting with the first player. We will use the terms cauldron and card pile interchangeably.

If in a cauldron there are some cards with a given color, then you can put a card of the same color into this cauldron. The exception is the green color (poison), it can be insert on an arbitrary pile. If there are only poison cards in a cauldron, the a player can insert a card with any color. Of course, at the beginning of the game, you can also put any card on an empty pile.

A player who exceeds the sum 13 (card values) in one of the cauldrons causes an explosion and takes all cards from this cauldron (he/she keeps them in a face-down stack).

A round ends when all players have got rid of cards from their hands. Each player count our cards. Each poison card is worth 2 negative points, the others are worth 1 negative point. There is an exception, i.e., a player do not count cards if he/she has the most cards with a given color. Then, the next player shuffles and deals the cards.

The entire game is played over 3 rounds. Finally, all points are added up with negative scores, whoever has the least wins.

## II. General version of Friday the 13th

*Goal: familiarize yourself with the first problem of the project.*

In the general version of the Friday the 13th game, we assume that a number of players **n** is at least 2 and at most **s**, where **s** is the sum of all cards. Each player receives at least one card. We also assume that the special rules (for three players) described in the original game do not exist. Such a gameplay for any number of players should be generated using a universal algorithm, i.e., we do not consider special situations for 3 and 2 players. In the general game we have:

- **k** – a number of cauldrons, which corresponds to the number of card colors - 1. This is due to the fact that we can add exactly one color of cards to each pile, except green, which can go to each of them,
- **g** – a number of green cards in the deck, we assume that green cards are always in and act as the poison, but their number and value may vary,
- **gv** – a value of green cards is the same for all poison cards, other colors can have different values,
- **o** – a number of cards of a color other than green,

- **ov** – the set of values of non-poison cards for a given color. As in the original game, every color has the same set **ov**, i.e., the cardinality of **ov** is equal to **o**,
- **e** – a value beyond which cauldrons explode,
- **r** – a number of rounds.

The number of cards in a deck is equal to: s = **g + k * o**. In the original game we have: $3 \leq$ **n** $\leq 6$ , **k** = 3 , **g** = 8, **gv** = 4, **o** = 14, **ov** = {1, 1, 1, 2, 2, 2, 4, 4, 5, 5, 5, 7, 7, 7}, **e** = 13, **r** = 3 with the exception for three players.

## III. Programming environment

*Goal: familiarize yourself with software used in the project.*

Use the language C. Any compilers and editors of this language can be used. The tools and IDE that are used to write a code, can work with various operating systems: Windows or Linux. To obtain greater compatibility with the STOS system, which will be a platform that will automatically verify initial functionalities of the project (details in the next section), it is recommended to use Visual Studio version 2017 or higher (the compiler version with Visual Studio 2017 version 15.9.5 is used in the STOS). For reading/writing files the functions from the stdio.h library can be used. The following can also be used:

- the standard C libraries: stdlib.h (e.g. functions: atoi, srand, qsort etc.), stdio.h (e.g. functions: fscanf, fopen, gentline etc.), string.h (e.g. functions: strcmp, strcpy etc.),

- objects cin, cout, cerr, clog and their counterparts from the standard library <iostream>.

All remaining libraries should not be used. In particular, it is **not allowed to use**:

- the standard template library (STL), and libraries: <ifstream> , <string>, <sstream> etc.

  The project can be written with the use of object oriented programming paradigm, and C++ can be used for compilation.

## IV. Details of grading

*Goal: To familiarize with the requirements, and with the additional and bonus criteria of grading the project*

The code of the project has to be submitted via STOS platform, using an account with login consisting of exactly 6 digits (no letters!). Moreover, the account has to be added to the following course: https://stos.eti.pg.gda.pl/index.php?p=viewprzedmiot&cid=356. Parts of the project can be submitted separately, as small, automatically-tested, solutions to subproblems. The final version of the code has to also be submitted via STOS, but it will be not graded automatically. The results of automatic grading are for the instructors to perform more robust grading of the project. Failing tests on STOS doesn't disqualify a given part of the implementation. The grading by an instructor is to be performed on the code submitted to the platform. The submission to STOS certifies that the deadline was met and that the code passed anti-plagiarism tests. A student is still allowed to demonstrate that the constructed program meets the functional requirements. Passing the tests on STOS gives a submitter a sense of psychic comfort and increases the likelihood that the tested functionalities will be graded at nearly 100%. However, the grade can be lowered in the case of: proved plagiarism (in this case a negative number of points can be given for this part of the project), weak comprehension of submitted code, logical errors in the code, bad style of coding, etc. Also, the listed properties may affect the final grading of the parts graded manually.

**Necessary functionalities (7 pts)**

All the listed elements have to be implemented. Lack of any is equivalent to obtaining 0 pts from this project.

**Building of a deck (1pt)** – Generating a file containing unshuffled deck of cards for the generalized version of the game. The parameters describing the game are to be provided as an input. The functionality may be graded automatically by STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=423&cid=356.
**Dealing the cards (1pt)** – Generating a file containing the state of the game after dealing the unshuffled deck of cards. The parameters describing the game are to be provided as an input. The functionality may be automatically graded by STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=194&cid=356.
**Loading a state of the game – part I (0.5 pt) –** Loading a file containing a state of the game and displaying chosen information. The information has to be consistent with the file. The functionality may be graded automatically by STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=833&cid=356.
**Loading the state of the game – part II (0.5 pt): -** the second part of the loading. Also may be graded automatically by STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=910&cid=356
**Checking the number and values of green cards (0.5 pt) –** after loading the state of the game the number of loaded green cards and their values shall be checked.
If no green card was loaded (no player has such a card and no cauldron has any) or there are cards with different values, then an appropriate message shall be printed.
This requirement might be graded automatically by STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=912&cid=356
**Checking the numbers of the cards (0.5 pt) -** after loading the state of game the program shall check if the numbers of cards of color different than green are equal.
An appropriate message shall be printed.
This functionality may be also graded automatically by STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=913&cid=356
**Checking the values of the cards (1 pt) –** after loading the state of the game it should be checked if the values of all colors of cards other than green are identical.
An appropriate message shall be printed.
This functionality may be also graded automatically by STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=918&cid=356
**Checking if the state of the game is proper (1 pt)–** after loading the game the program shall check: if the number of cards assigned to a player are proper, if at any

stack the number of cards is no greater than maximum possible, and if a stack does not contain a forbidden combination of colors.
An appropriate message shall be printed.
This functionality may be also graded automatically by STOS:
https://stos.eti.pg.gda.pl/index.php?p=show&pid=914&cid=356
**A simple move (0.5 pt)** – after loading the state of the game, the current active player shall be able to make a simple move, the new state of the game shall be written to a file.
This functionality may be also graded automatically by STOS:
https://stos.eti.pg.gda.pl/index.php?p=show&pid=915&cid=356
**Explosion of a cauldron (0.5 pt)** – after a simple move the program shall check if a cauldron exploded. If yes, then before saving the new state of the game all cards from the cauldron shall be moved to the set of the cards of the active player
This functionality may be also graded automatically by STOS:
https://stos.eti.pg.gda.pl/index.php?p=show&pid=916&cid=356

**Additional functionalities (8 pts)**

Additional points can be obtained for implementing the functionalities listed below (Not all have to be implemented. Also, they do not have to be implemented in order):

**End of round check (1 pt)** – the program shall check if the round ended. If yes, then an information about points received by the players shall be printed.
The requirement may be automatically graded by STOS:
https://stos.eti.pg.gda.pl/index.php?p=show&pid=917&cid=356
**A simple match (1 pt)** – 2 copies of the programs can be started and a match between them can performed. The programs do not have to play reasonably: playing a random card is sufficient. However, each of the copies have to follow rules. In particular, it has to: verify that the numbers and types of cards, verify if the values of the cards in each cauldron is at most **e** – maximum allowed, verify if the number of cards players is proper, and if the round or game has ended.

Each copy has to be given the number of player represented by the copy, for example via command line parameter or via some configuration file.
The programs can synchronize by any mean, for example by:

The file representing the state of the game. There can be an external arbiter (another program), which stores an information about the state of the game and is responsible to supply it to the program, during the execution of the program. Without an arbiter program the information can be stored in the file containing the description of the state of the game, and the programs may periodically read the file to check if it is their turn.
An implementation of the program which is able to start two instances of „players" and sharing the data structures between them. This is the easiest version, due to the lack of any communication between processes.
Using sockets, message ques, external services or any other means of communication.

Moreover, before the start of the game the cards have to be shuffled.
Due to the fact that the cards assigned to the players can be seen by anyone, it can be verified if their number is proper.

**Choosing the smallest card (1 pt) –** an implementation of a player who choose the card with lowest value and put it at the stack of the cards with the lowest sum of the cards.
**Dropping the biggest cards (1 pt) –** in the case that any move of type described previously results in an explosion of the cauldron, the card with highest value shall be chosen and put in the least filled cauldron. It allows to drop a dangerous card and increase the likelihood of receiving a small number of cards.
**Choosing an optimal card (1 pt)** – an implantation of the player adding always the biggest card that does not explode the cauldron. If this is not the case adding the biggest card as in **dropping the biggest cards.**
**Maximization of a color (1 pt) –** if the player has to explode a cauldron by some card, then she chooses the card that maximizes the number of cards of a chosen color. The player considers what will be the state of her cards after each explosion type she can cause. The player compare these states and chooses the one that maximizes the cards of any (chosen by the program) color except the green (i.e. in the new state the number of cards of a chosen color except the green will be maximum).
**Complex game (1 pt) –** a game for greater number of players (6-20) can be started.
**Testing the effectiveness of the players (1 pt)** – performing automatic tests of at least two types of players – i.e., players of highly different strategies. For example: comparing the number of games won by a program playing randomly and a program avoiding exploding a cauldron, by always adding biggest card that does cross the maximum of a stack/ cauldron. A few hundreds of matches is sufficient.

**Bonus requirements (3 pts)**

**Taking part in the tournament (1 pt) –** the program is able to take a part in a tournament. It means an implementation that plays all required matches with all registered opponents in accordance with the rules. The program does not have to win any of the matches but is should not crash or hang.

**Winning the tournament (0.5pt -2pts) –** i.e. obtaining high enough position. Precise information about the thresholds to obtain points will be available only after the tournament. They will be determined by the number of program registered for the tournament, the number of programs disqualified, and the general quality of the programs.

The first 11 taksks that correspond to the subsequent functionalities can be written in a form of short separate programs or in a form of a larger program that can be then `adjusted' to the STOS requirements; this does not influence the marking. It is important that the „simple match" all these functionalities are put together into a single program that plays the game. Such a program should:

1. know its identity as a player (which player am I?),
2. be able to load the current state of the game,
3. in case when the player controlled by the program has its turn, it should make a move according to the rules,

4. when an error (incosistency) in the state of the game is detected, then the program should display an error,
5. inform about the outcome if the loaded state or the state after making a move is a final one.

Moreover, for the functionality „simple match” two copies of the program should be executed and in this way a full game should be played, i.e.:

1. Depending on the parameters **n, k, g, gv, o, ov, e**, an initial state of the game should be generated, the cards should be shuffled and dealt.
2. A sequence of moves should occur, leading to a final state of the game.
3. The current players' scores should be shown.
4. The above steps should be repeated **r** times.

## V. Tournament

*Goal: familiarize yourself with the tournament.*

Submitting solutions on the STOS has one more task, it allows to standardize the file format that allows you to save the game state between moves of subsequent players. This will allow you to play games between different versions of your project, but also between programs of different authors. In the final, we will conduct a competition for the best player. The tournament will consist of a series of games between two or more programs playing simultaneously. Its exact form is not yet defined and it is quite difficult to describe completely in this manual. All details will be described on the STOS in the content of the course: https://stos.eti.pg.gda.pl/index.php?p=viewprzedmiot&cid=356 in the TOURNAMENT tab. It is almost sure that communication between the programs will take place using a file with a strictly defined format in the Windows 10 environment. This means that programs will be compiled by the Microsoft compiler cl.exe with msbuild.exe from Visual Studio.