POLISH-JAPANESE ACADEMY
OF INFORMATION TECHNOLOGY

Turn-based game

1. Project goal

The goal of this project is to demonstrate student's knowledge of basic elements of the C++ programming language as well as the general knowledge of creating smaller programming projects. Including, among other things, the proper selection of programming tools and creation of a documentation of constructed auxiliary modules.

2. Project description

Project is based upon a concept of a console application which will simulate a turn-based combat game with different interactions between enemies (example: Pokemon battle system).

The player must have a set of 6 creatures which they can control. The goal of the game is to successfully beat all of the opponents with least expense of their own team.

Application should be run from the command line and, based on the commands read from the keyboard, use defined actions for the character.

Detailed specification of the project is presented in the rest of this document.

3. Gameplay description

Gameplay should be separated into at least 4 rounds, where each round should have at least 4 enemies. Every enemy is chosen randomly from 15 creatures from which the player can choose their team.

At the beginning of the gameplay, player can choose a team of six creatures from pre-defined set.

Gameplay is partitioned into player-enemy turns, where in one turn character can take one of the following actions:

- Use special ability/attack, or
- Change active creature, or

- Evolve the creature

After falling below zero Health Points, the creature faints and cannot be used again till the end on the battle. After defeating an enemy, an announcement should appear with a possibility to save the state and close the game or to continue the gameplay.

After defeating all enemies, the game ends with the player winning.

4. Non-functional requirements

Project should be implemented in accordance with clean code production rules and good programming practices (both universal and specific to C++).

Bearing in mind that some of these criterium might be a moot point, student should make sure that their understanding of these concepts is compliant with elements presented in class – for example via consultation with the tutor.

The implementation should be thought trough. This means that student should choose the possibly best programming tools and use them according to the previously stated good practices. A solution that "just works" might not be sufficient to earn a passing grade.

Programs that do not compile will be granted 0 points. Student should make good use of splitting the project into multiple files.

The project should be documented. It is required that every non-trivial function, method and class (but not their attributes) has a comment concerning them compliant to the Doxygen standard. Main aspects that should be brought up are the parameters (`@param`), return values (`@return`) and a short description of functionality (purpose) of the method/function/class.

This description should not include details concerning the way it was implemented. Please put emphasis on **what** is done and not on **how** it is done.

A good example of this is the Java String class documentation that can be found here.

In tools like CLion or Visual Studio, after implementing the function, class, or method, one is able to generate a template of such documentation by inputting the following sequence of characters right above the element one wants to document: /**, followed by the `Enter` button.

5. Functional requirements

Every creature should have certain attributes defining its characteristics and impact on the gameplay. The attributes are as following:

- Strength – defines how much damage can creature deal with singular attack;
- Dexterity – defines the chance of dodging an incoming attack;
- Health Points – defines, how much damage creature can take before fainting;

- Special power – defines the characteristics of special power for a creature and the maximum number of uses during one battle;
- Experience Points (EXP) – how many Experience points a creature can gain by defeating the enemy.

Special Power must be of adequate type for the creature to use (for example: Water type creature cannot use Tornado or Fire Breath as a special power). Special power must be split into offensive and defensive powers, which may temporarily (for certain number of turns) alter the attributes of the player or the enemy.

Every creature also has a type (Water, Earth, Air, Fire, Ice, Steel) which impacts the effectiveness of their attacks against other creatures. The table defining those interactions may be found below (Table 1).

Attacking creature	Water	Earth	Air	Fire	Ice	Steel
Water						
Earth						
Air						
Fire						
Ice						
Steel						

Table 1. Interactions between types..

The color **green** defines increased effectiveness, while the color **yellow** defines lessened effectiveness.

The program should have a set of at least 15 creatures, from which the player can choose their team and from which the enemy teams are drawn.

The game should have at least 4 enemies, where every enemy should have at least 4 creatures. It is required to implement difficulty levels that will change those values.

Creature and attack selection should be done via keyboard. User interface should be readable and must have instruction manual obtainable by inputting in the console `-h` or `-help` command.

Every creature after gaining a certain amount of EXP may evolve, increasing its attributes. The player should be able to choose which two attributes they want to increase for the creature.

6. Grading criteria

Table 2 defines the number of points student may earn by implementing certain functionalities.

Requirement	Points	Additional comment

Appropriate selection and use of programming tools.	5	For example, using standard algorithms as intended, using templates rather than macros, using `std::function` instead of pointers to functions, properly selecting containers according to their purpose. Selecting and using tools so that they clearly indicate programmatic intentions - this also includes clean code, and therefore the selection of descriptive, good variable names.
Adherence to *const-correctness*.	1	This means using `const` not only where it can be used, but also wherever an element should not conceptually be changed.
Avoidance of unnecessary copies of data in the program.	1	According to the knowledge presented in the exercises. This applies to types larger than primitives - it is advisable to copy variables of type int or double instead of passing them with `const &` (of course, for unmodified copies, marking them with `const` is still required).
Creation of the documentation complaint with Doxygen standard.	4	
Clarity of error messages during incorrect use of the program	3	
Correct implementation of polymorphic hierarchy of classes/ enemy generation.	5	When generating opponents, the student should think about the generation algorithm, where each creature should have balanced features and skills. The algorithm shouldn't be trivial. If in doubt, please consult the teacher.
Correct implementation of saving to and reading from `.txt` file.	4	
Correct implementation of the interaction table.	3	The solution of the interaction tables should be as close to the optimal one as possible. The closed range of interactions and their potentially frequent search should be taken into account.

Correct implementation of the attributes and their impact on gameplay	5	
Correct implementation of the gameplay.	10	Gameplay includes: user interaction, creature exchange, turn-based actions, enemy exchange, scalability of difficulties, interface design.
Correct implementation of creature evolution.	5	Evolution implementation includes calculation of the EXP points needed to get the creature to evolve and the selection of the statistics to increase.
Correct implementation of attack action.	3	
Correct implementation of `-help` function.	1	

Table 1. Grading criteria.

7. Defense

Student will be questioned about the finer details of their implementation. Failure to understand and identify any piece of code may be the basis for **failing the entire project**. It is suggested that students who finished the project long before the due date made sure to remember the finer details of their work.

8. Additional remarks

It is forbidden to use solutions that are not well understood. Every tool must be mastered by the student. Using the tools not shown during the class and lectures is allowed but bear in mind that during the defense student might be questioned about those elements too.

In case of any ambiguities please consult the content and requirements of the project with the tutor.