Change Log

Major design changes will be recorded here.
Format: “Name: date, time, class name: Changes. Reason”

Alex: 17/05/2023 4:44pm, Type class: created as means to easily do type effectiveness calculation for the Pokemon class.

Alex: 18/05/2023 3:49pm, JSON class: created a JSON class that parses the PokeAPI to receive relevant Pokemon data such as moves and statistics.

Alex: 18/05/2023 8:10pm, Button class: made to display usable buttons using sfml on the game ui that will perform commands such as attacking moves or switching out Pokemon.

Alex: 20/05/2023 9:00pm, Stats class: due to statistical modification issues, a class was created to handle the complexity of generation 1 statistic modifications and to hold 12 base/modified statistics to be used by Pokemon. The Stats class shares an associative relationship with Pokemon in a one-to-one relationship.

Alex: 20/05/2023 10:00pm, Type class: created to hold information on the type matchups and typings instead of Pokemon class.

Style Guide

Headers

- All header files should have define guards and should be in the format FILES_H
- A header must be self contained; meaning it includes all the dependencies it requires. E.g. “#include <string>” or “#include <SFML/graphics.hpp>”
- The extern keyword is forbidden. Use header files instead.
- Do not indent preprocessor statements. They should always be at the beginning of a line for easy identification
- Each file will include the following standard header which includes a brief description of the class

```
/*****************************************************************************/
/*                                                                           */
/*        ▄▄▄▄▄▄▄  ▄▄▄▄▄▄▄▄  ▄▄▄▄▄▄▄▄▄ ▄▄▄     ▄▄▄▄▄▄▄  ▄▄▄ ▄▄▄▄▄▄▄▄  ▄▄▄▄▄▄▄▄▄        */
/*        ▄▄▄  ▄▄▄ ▄▄▄▄  ▄▄▄ ▄▄▄▄    ▄▄▄▄    ▄▄▄▄  ▄▄▄▄ ▄▄▄▄ ▄▄▄▄ ▄▄▄▄  ▄▄▄           */
/*        ▄▄▄▄▄▄▄▄ ▄▄▄▄  ▄▄▄ ▄▄▄▄▄▄▄  ▄▄▄▄    ▄▄▄▄▄▄▄▄ ▄▄▄▄ ▄▄▄▄  ▄▄▄ ▄▄▄▄▄▄▄        */
/*        ▄▄▄  ▄▄▄ ▄▄▄▄  ▄▄▄ ▄▄▄▄    ▄▄▄▄    ▄▄▄▄  ▄▄▄▄ ▄▄▄▄ ▄▄▄▄ ▄▄▄▄  ▄▄▄           */
/*        ▄▄▄  ▄▄▄ ▄▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄▄▄ ▄▄▄ ▄▄▄ ▄▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄▄▄        */
/*                                                                           */
/*        ▄▄▄  ▄▄▄ ▄▄▄▄  ▄▄▄ ▄▄▄    |                   |          */
/*        ▄▄▄  ▄▄▄ ▄▄▄▄▄ ▄▄▄ ▄▄▄    |  OOP Semester 1 2023    |          */
/*        ▄▄▄  ▄▄▄ ▄▄▄ ▄▄ ▄▄▄ ▄▄▄    |  Camille, Rose, Alex    |          */
/*        ▄▄▄  ▄▄▄ ▄▄▄ ▄▄ ▄▄▄ ▄▄▄    |  Pokemon Showdown remake  |          */
/*         ▄▄▄▄▄▄▄ ▄▄▄  ▄▄▄▄▄ ▄▄▄    |                   |          */
/*                                                                           */
/*       This is the base class for all objects that need rendering          */
/*     It is an abstract class that holds the sprites for each object        */
/*                                                                           */
/*****************************************************************************/
```

Variables:

- Declare local variables as close as reasonably possible to its first usage.
- Local variables should be declared and initialised on the same line.
- Only use static variables for simple types. Static classes are forbidden.
- Cast variables using c++ style casts, using the syntax ‘static_case<type>(variable)’
- Use ++i instead of i++ unless you specifically need the value before incrementing.
- Use const wherever possible
- For numbers larger than integers, use the <cstdint> library and specify ‘int64_t’ etc.
- Do not use unsigned integers.
- Use ‘nullptr’ for null pointers (address 0) and ‘\0’ for null characters (0 ascii.)
- Use sizeof(variable) instead of sizeof(type). This prevents bugs when changing a variable’s type

Classes

- Class and struct names should have the first letter of each word capitalised and don't include underscores.
- Use classes whenever possible. Only use structs if its whole use is carrying data.
- Avoid calling virtual functions in constructors.
- When inheriting classes, only inherit publicly.
- Overload operators according to the official use of the operator for primitive types (e.g overloading + should be addition or concatenation of classes)
- All data in classes should be private unless they are constant
- When writing classes, put variables and functions in the order "public, protected, then private".
- Indent the public private and protected labels by one space from the beginning of the line
- If a list of constructor initializers is longer than 80 characters, indent each extra line by 4 spaces from the beginning of the line and line up each line for neatness

Functions:

- Function names should start with a capital letter and each subsequent word should be capitalised.
- The line before any function or class should contain a comment describing what it is and its usage
- Keep functions short; try to keep functions under 40 lines
- Only overload functions if its purpose is immediately recognisable
- Don't output to output variables, make them return a value instead.
- Optional arguments can be used for all functions but virtual functions
- Switch statements should always have a default value. If the default value should never run, make it an error.
- Function parameters may wrap to the next line, however they should be indented to the first function parameter.

Loops & control statements:

- Always put a space between a keyword and an expression (e.g. if (condition), while (condition))
- In a statement with multiple inline expressions (like for loops) always put a space between a semicolon and the following condition, however do not put a space between a semicolon and a closing bracket or a second semicolon.
- Place the opening curly bracket on the same line as the statement.
- Curly brackets may be omitted if the statement only contains one line of instructions.
    - This does **not** apply to multi condition statements like "if … else" and "do … while". They must not omit any curly brackets.
- Everything other than a statement's condition and the opening bracket must go on a new line immediately underneath the statement.
- Never pad a statement using newlines
- Put each following condition statement on the same line as the closing curly bracket

Formatting:

General

- Use descriptive names that aren't abbreviated or shortened. Only use an acronym if they're listed on Wikipedia. Prioritise readability over saving horizontal space.
- All file names should be in lowercase with each word separated by an _. C++ files should have the .cpp extension and header files should have the .hpp extension.
- Keep file names as specific as possible.
- Comments should be '/*' comments. Indent between your comment and the comment symbol for both the start of the comment and end of comment.
    - E.g. "/* comment */"
- It is recommended that whitespace is added after the end of a comment so that the '*' symbol ends at character 80. However, it is not required.
- A line can be at most 80 characters long unless a comment or string literal cannot be conveniently split (i.e. URLs.)

Whitespace:

- No trailing white space or spaces on empty lines.
- **Indent using spaces and should be 2 spaces wide**
- Limit the use of vertical white space by not introducing unnecessary blank lines.
    - Think of white space like a paragraph break. Use it to separate differing concepts but don't overuse them
    - Never have more than 1 blank line before or after a function
    - Do not start or end a function, object, or namespace with a blank line

Floats

- Floating point literals should always include a decimal and digits on both sides of it, even if it's a whole number (1.0 not 1)

Pointers

- The symbols for pointers, addresses, and accessing objects/pointers to objects should **not** be padded with spaces. (Class->var not class -> var)
- When declaring pointer variables, stick the pointer symbol to the variable name, not the type

TODO comments

- Todo comments should be formatted as follows: "//TODO (name) feature to be added. Date to be added by"
- Try to honour the due date of your todo comments. If for any reason you can't, put an update comment underneath it with a new due date e.g. "//update: function more complicated than expected. New due date: 13th may 11am".

Variables, Booleans, and operators

- Variable names should be in all lowercase, with words separated by underscores.
- Constant names should be prefixed with ‘k’.
- If a Boolean expression has to be spread across multiple lines, always put the next Boolean symbol at the end of the line
- Don’t surround a return expression with brackets unless it would normally have brackets around it (e.g for maths expressions; return (1 + 2) * 4; is ok but return (integer) isn’t.”
- Use = to initialise basic variables on declaration and use {} for data types like string
- Surround unary operators, variable assignments, and Boolean operators like +, *, ||, and = with spaces
    - This rule does not apply to the ++ and — operator
