Where the bug lies…

The Prefect Programmer

- Mostly, programmers write hundreds of lines of code, then compile and execute it and everything just works.

Yeah. Exactly…

- Usually you don‘t even get to compile things…
- But once things compile, you normally get segmentation faults as first reaction.
- There are several ways to improve on that situation. One of them is reacting to unforseen problems before even they exist.

Three Constructs:

- When the program flow encounters an unusual problem and that is cought with a conditional for instance, you have more or less 3 possibilities:
- Abort the program
- Throw an exception
- Only display a warning

The Message

- Use this when you can deal with the problem easily (simply skip something for instance).
- Notify the user by writing a message to the output stream:

```
COUT(1) << „An Error has occured in blahblah…“
        << std::endl;
```

The Exception

- If the problem is more serious and you want to abort the current task (loading a level), but not the program, the exception is just right.
- Avoid using them if you can think of a better way. But a parser is a very good example to use them anyway.
- With `catch (std::exception& ex) { … }` you can deal with exceptions.
- Throwing one:

```
ThrowException(FileNotFound, „Could not find
config.ini“);
```

An example:

```
try
{
   parseScript();
}
catch (std::exception& ex)
{
   // Display it too:
   COUT(1) << ex.what() << std::endl;
   // Continue with the program considering
   // that the script could not be parsed.
   parsingFailed_ = true;
}
actAccordingly();
```

Assertion

- Sometimes you are just sure that nothing can go wrong because of the program code flow.
- But what if it does go wrong after all?

  → You made a programming mistake.
- Best tatics would be to immediately abort the program and display the code position where the problem occured.
- Unfortunately you mostly assume that the error doesn‘t happen. The result is segmentation fault, maybe at a completely different position.
- `assert(condition);` can guarantee that the program doesn‘t continue if `condition` is false. It also displays the right code position at run time.

Placing asserts

```
int myFunction(unsigned int position)
{
    assert(position < myArray.size());
    return myArray_[position];
}
```

‚position‘ is an internal parameter that always is within the array boundaries. At least according to your design ideas…