

# GAMBIT-light



1. What is GAMBIT-light?
2. Why do we need it?
3. *For users:* how does it work?
4. *For GAMBIT developers:* how do we maintain it?
5. Future plans

# 1. What is **GAMBIT**-light?

- **GAMBIT-light: GAMBIT without all the physics**
- A lightweight yet powerful tool for statistical fits and optimisation tasks
- *What GAMBIT-light is not:* A full-blown tool for global fits in <your discipline> — for that you'd want more of the full GAMBIT functionality
- Key design principles:
  - Users should never need to modify and rebuild any GAMBIT code
  - Minimise the extra maintenance work for GAMBIT developers



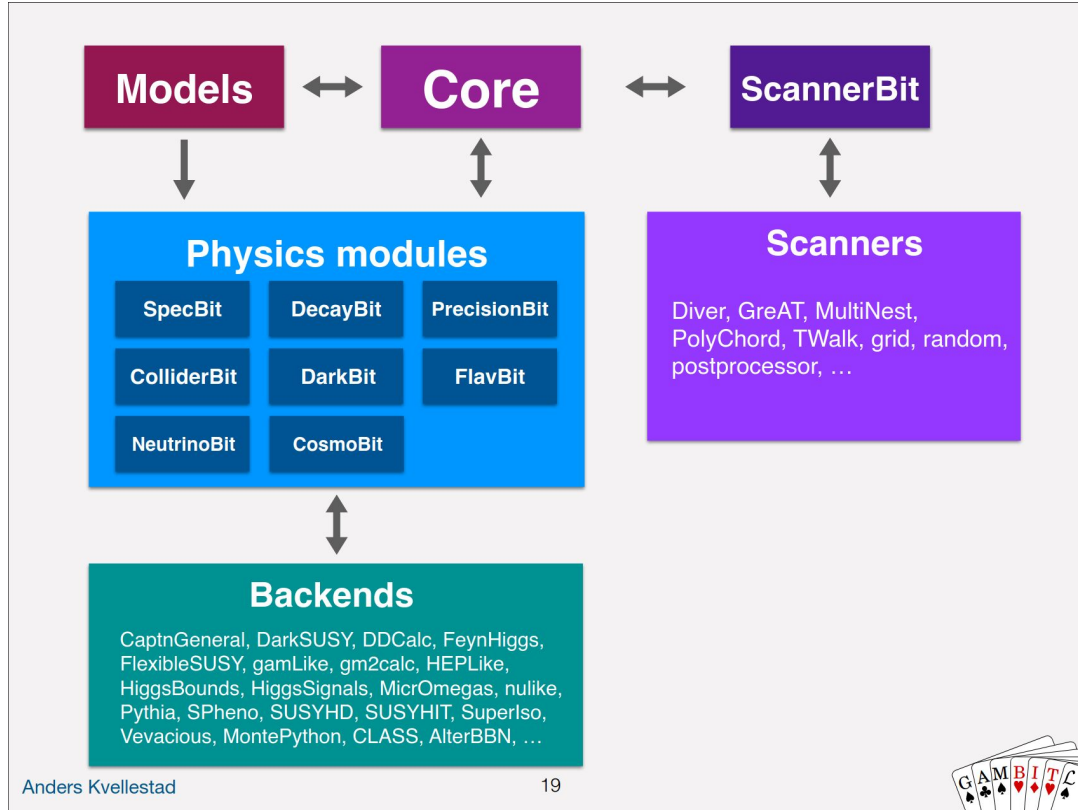
**Maiken Pedersen**  
(UiO)

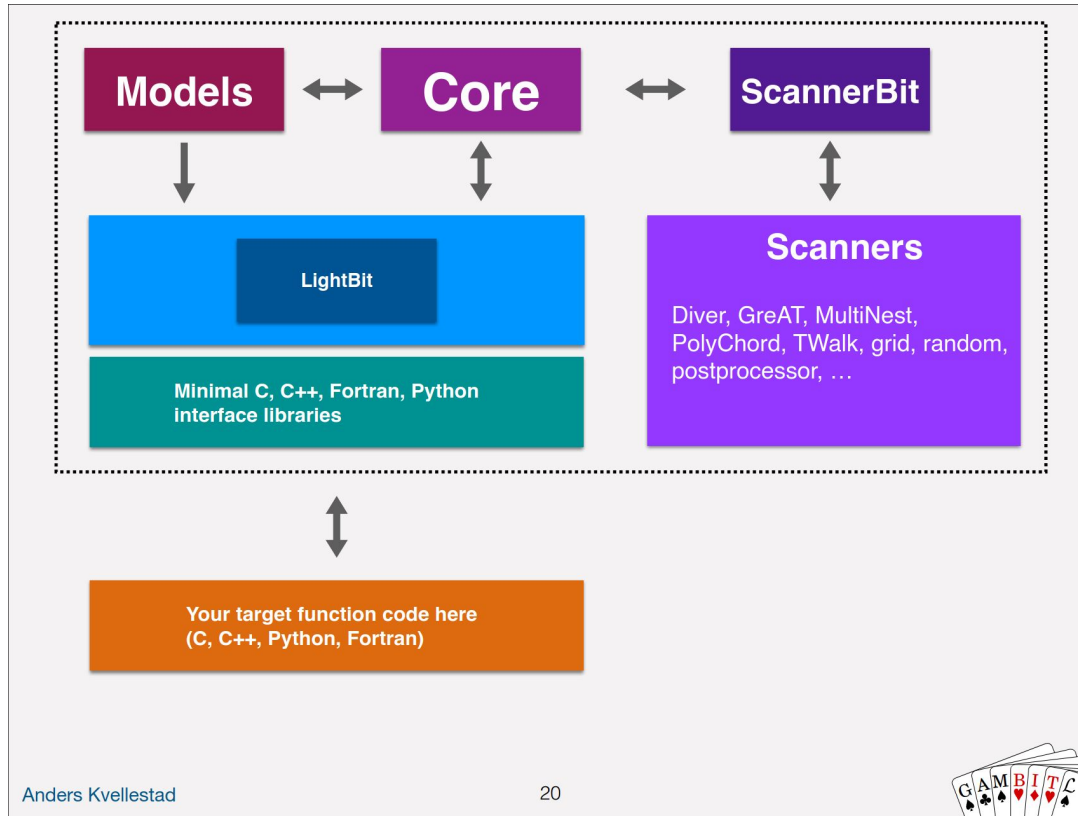


**Marcin Krotkiewski**  
(UiO)

*Brainstorming, early  
code drafts, testing:*

**Janina Renk,  
Fabio Zeiser,  
Andreas Mjøs,  
++**





## **2. Why do we need it?**

- Background:
  - We designed GAMBIT to be very general and physics-agnostic
  - We put a lot of effort into the main code framework (Core, ScannerBit, Printers, CMake system, ...)
  - → GAMBIT *can* be used for optimisation/fits outside particle/astro/cosmo
- Practical experience:
  - GAMBIT is a particle physics power tool → fairly heavyweight
  - Considerable threshold for non-experts to pick up and use/modify
  - In particular: frequent and slow recompilation kills the flow of the early development/experimentation stage of projects



- External motivation for GAMBIT-light:
  - Help projects outside particle/astro/cosmo use GAMBIT
  - In particle/astro/cosmo:  
suitable for quick experimentation, MSc projects, etc.
- Internal motivation for GAMBIT-light
  - Increase visibility and impact of Core & ScannerBit work
  - Increase visibility for Core & ScannerBit papers  
(GAMBIT-light should *not* have a separate code paper – users should cite the main GAMBIT & ScannerBit papers)
  - Sandbox for quick experimentation

**3. For users: how does it work?**

## 1. Build GAMBIT once

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DWITH_MPI=On -DCMAKE_CXX_COMPILER=g++-11 -DCMAKE_C_COMPILER=gcc-11
make -jN scanners # where N is the number of cores to use for the build, e.g. 4
cmake ..          # this step is needed for GAMBIT to detect the built scanners
make -jN gambit
```

## 2. Develop your target/likelihood function code

```
1 # To import gambit_light_interface, first append the directory containing
2 # gambit_light_interface.so to sys.path. (Alternatively, add this directory
3 # to the PYTHONPATH environment variable.)
4 import sys
5 import os
6 current_dir = os.path.dirname(os.path.abspath(__file__))
7 sys.path.append(os.path.join(current_dir, "../lib"))
8 import gambit_light_interface as gambit_light
9
10
11 # User-side log-likelihood function, which can be called by GAMBITE-light
12 def user_loglike(input_names, input_vals, output):
13
14     # Make a dictionary of the inputs?
15     input = {input_names[i]: input_vals[i] for i in range(len(input_names))}
16
17     # Error handling: Report an invalid point using gambit_light.invalid_point.
18     # gambit_light.invalid_point("This input point is no good.")
19
20     # Error handling: Report a warning using gambit_light.warning.
21     gambit_light.warning("Some warning.")
22
23     # Error handling: Report an error using gambit_light.error.
24     # gambit_light.error("Some error.")
25
26     # Error handling: Error handling, alternative to using gambit_light.error: Throw an exception.
27     # raise Exception("Some exception.")
28
29     # Compute loglike
30     loglike = input["param_name_1"] + input["param_name_2"] + input["param_name_4"]
31
32     # Save some extra outputs
33     output["py_user_loglike_output_1"] = 1
34     output["py_user_loglike_output_2"] = 2
35     output["py_user_loglike_output_3"] = 3
36
37     return loglike
38
```

## 2. Develop your target/likelihood function code (C++/C/Fortran: build as shared library)

```
1 #include "gambit_light_interface.h"
2
3 // User-side log-likelihood function, which can be called by GAMBIT-light.
4 double user_loglike(const std::vector<std::string>& input_names,
5                    const std::vector<double>& input_vals,
6                    std::map<std::string, double>& output)
7 {
8
9     // Make a map of the inputs?
10    std::map<std::string, double> input;
11    for (size_t i = 0; i < input_names.size(); i++)
12    {
13        input[input_names[i]] = input_vals[i];
14    }
15
16    // Error handling: Report an invalid point using gambit_light_invalid_point.
17    // gambit_light_invalid_point("This input point is no good.");
18
19    // Error handling: Report a string warning using gambit_light_warning.
20    gambit_light_warning("Some warning.");
21
22    // Error handling: Report an error using gambit_light_error.
23    // gambit_light_error("Some error.");
24
25    // Error handling, alternative to using gambit_light_error: Throw a runtime_error.
26    // throw std::runtime_error("Some runtime_error.");
27
28    // Compute loglike
29    double loglike = input.at("param_name_2") + input.at("param_name_3");
30
31    // Save some extra outputs
32    output["cpp_user_loglike_output_1"] = 1;
33    output["cpp_user_loglike_output_2"] = 2;
34    output["cpp_user_loglike_output_3"] = 3;
35
36    return loglike;
37 }
38
39 GAMBIT_LIGHT_REGISTER_LOGLIKE(user_loglike)
40
```

### 3. Configure GAMBIT run with a YAML file

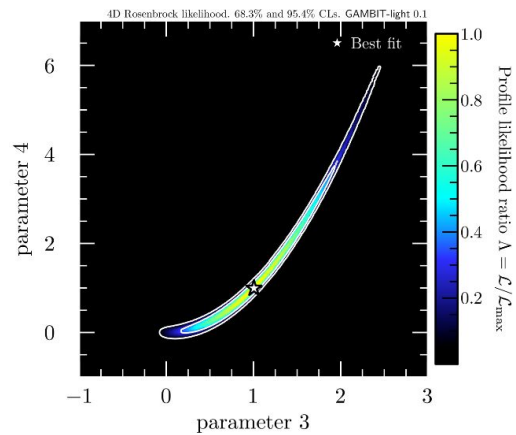
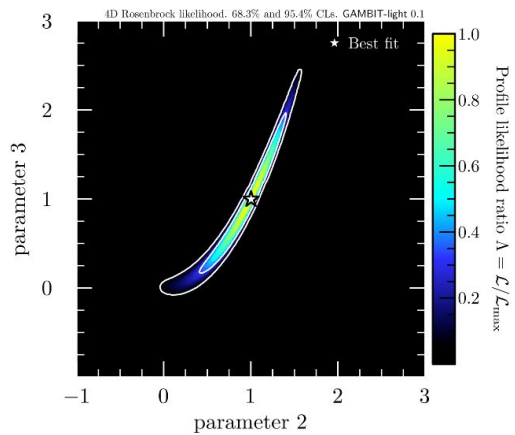
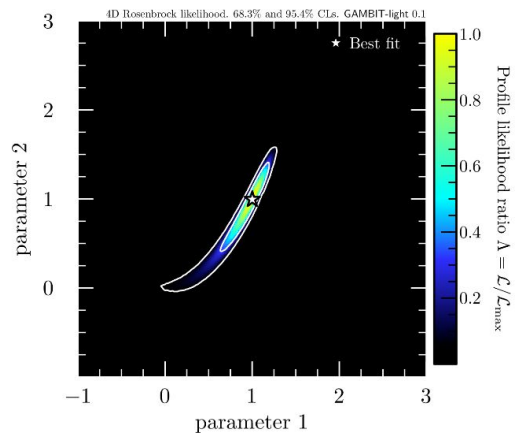
```
1  UserModel:
2
3    p1:
4      name: param_name_1
5      prior_type: flat
6      range: [0.0, 5.0]
7
8    p2:
9      name: param_name_2
10     prior_type: flat
11     range: [0.0, 5.0]
12
13    p3:
14     name: param_name_3
15     fixed_value: 3.0
16
17    p4:
18     name: param_name_4
19     same_as: UserModel::p1
20
21    p5-p7:
22     name: param_name_
23     prior_type: flat
24     range: [-1.0, 1.0]
25
26  UserLogLikes:
27
28    py_user_loglike:
29     lang: python
30     user_lib: gambit_light_interface/example_python/example.py
31     func_name: user_loglike
32     output:
33     - py_user_loglike_output_1
34     - py_user_loglike_output_2
35     - py_user_loglike_output_3
36
37    cpp_user_loglike:
38     lang: c++
39     user_lib: gambit_light_interface/example_cpp/example.so
40     func_name: user_loglike
41     input:
42     - param_name_2
43     - param_name_3
44     output:
45     - cpp_user_loglike_output_1
46     - cpp_user_loglike_output_2
47     - cpp_user_loglike_output_3
```

## 4. Run GAMBIT

```
mpiexec -np 4 ./gambit -f yaml_files/your_configuration_file.yaml
```

**5. Modify your own code, rerun GAMBIT, modify your own code, rerun GAMBIT, ...**





## 6. Analyse output samples (saved in HDF5 or ascii format)

## Also: user-supplied prior transformation

### Python

```
40 # User-side prior transform function, which can be called by GAMBIT-light.
41 def user_prior(input_names, input_vals, output):
42
43     for i,v in enumerate(input_vals):
44         output[i] = v * 10.
45
```

### C++

```
43 // User-side prior transform function, which can be called by GAMBIT-light.
44 void user_prior(const std::vector<std::string>& input_names,
45               const std::vector<double>& input_vals,
46               std::vector<double>& output)
47 {
48     for (size_t i = 0; i < input_vals.size(); i++)
49     {
50         output[i] = input_vals[i] * 10.;
51     }
52 }
53
54 GAMBIT_LIGHT_REGISTER_PRIOR(user_prior)
```

## Also: user-supplied prior transformation

```
1  UserModel:
2
3  p1:
4    name: param_name_1
5  p2:
6    name: param_name_2
7  p3:
8    name: param_name_3
9  p4:
10   name: param_name_4
11  p5-p7:
12   name: param_name_
13
14  UserPrior:
15
16   lang: python
17   user_lib: gambit_light_interface/example_python/example_prior_transform.py
18   func_name: user_prior
19
20  UserLogLikes:
21
22  py_user_loglike:
23   lang: python
24   user_lib: gambit_light_interface/example_python/example.py
25   func_name: user_loglike
26   output:
27     - py_user_loglike_output_1
28     - py_user_loglike_output_2
29     - py_user_loglike_output_3
30
31  cpp_user_loglike:
32   lang: c++
33   user_lib: gambit_light_interface/example_cpp/example.so
34   func_name: user_loglike
35   input:
36     - param_name_2
37     - param_name_3
38   output:
39     - cpp_user_loglike_output_1
40     - cpp_user_loglike_output_2
41     - cpp_user_loglike_output_3
42
```

**4. For GAMBIT developers: how do we maintain it?**

- Will describe the current setup – we can change this as needed
- [github.com/GambitBSM/gambit\\_light](https://github.com/GambitBSM/gambit_light) is a fork from our main repo [github.com/GambitBSM/gambit](https://github.com/GambitBSM/gambit):

The screenshot shows the GitHub repository page for 'gambit'. The repository is in the 'master' branch and has 166 branches and 30 tags. The 'About' section describes it as 'Global And Modular BSM Inference Tool'. The 'Tool' section lists various backends like CollierBit, Core, CondoBit, DarkBit, DecaBit, Elements, ExampleBit\_A, ExampleBit\_B, FluxBit, Loge, Modes, NeuroBit, ObjectwiseBit, PrecisionBit, Printers, ScannerBit, SpecBit, UbiS, cmake, config, contrib, doc, gum, yam1\_files, igitron, BUILD\_OPTIONS.md, CMakeLists.txt, README.md, HISTORY, README.md, and README\_OSX.md. The 'Releases' section shows 30 tags and a 'Create a new release' button. The 'Packages' section shows 'No packages published'. The 'Contributors' section shows 33 contributors. The 'Languages' section shows a bar chart with C++ 35.0%, Python 3.0%, Fortran 1.0%, CMake 2.0%, and Other 2.0%. The 'README.md' section is partially visible at the bottom.

The screenshot shows the GitHub repository page for 'gambit\_light'. The repository is in the 'master' branch and has 9 branches and 28 tags. The 'About' section describes it as 'A lightweight version of GAMBIT'. The 'Tool' section lists various backends like CollierBit, Core, Elements, LightBit, Loge, Models, Printers, ScannerBit, UbiS, cmake, config, contrib, doc, gambit\_light\_interface, yam1\_files, igitron, BUILD\_OPTIONS.md, CMakeLists.txt, README.md, and README\_OSX.md. The 'Releases' section shows 26 tags and a 'Create a new release' button. The 'Packages' section shows 'No packages published'. The 'Contributors' section shows 33 contributors. The 'Languages' section shows a bar chart with C++ 35.0%, Python 3.0%, Fortran 1.0%, CMake 2.0%, and Other 2.0%. The 'README.md' section is partially visible at the bottom.

### GAMBIT-light

**GAMBIT-light** is a powerful yet easy-to-use tool for computationally difficult statistical fits and optimisation tasks. The user can provide their target/likelihood function as a Python, C, C++ or Fortran library.

#### Some features of GAMBIT-light:

- A unified interface to a collection of powerful, MPI-parallelised sampling/optimisation algorithms
- Simple interface for connecting user-supplied target/likelihood functions (and sampling priors, if needed) as runtime plugins
- Outputs in different the formats (binary or text)
- Systems for exception handling and logging
- Safe shutdown and resuming of aborted runs
- Run configuration via a simple YAML file

GAMBIT-light is a spin-off project from **GAMBIT** (the Global And Modular BSM Inference Tool), <https://gambitbsm.org/>, a software tool for large-scale statistical fits in particle physics and astrophysics.

#### Citation(s)

- Please cite the following GAMBIT papers if you use GAMBIT-light:
- GAMBIT Collaboration: P. Athron, et al., **GAMBIT**: The Global and Modular Beyond-the-Standard-Model Inference Tool, Eur. Phys. J. C 77 (2017) 784, arXiv:1705.07908
  - GAMBIT Scanner Workgroup: G. D. Martinez, et al., Comparison of statistical sampling methods with **ScannerBit**, the GAMBIT scanning module, Eur. Phys. J. C 77 (2017) 761, arXiv:1705.07959

- Will describe the current setup – we can change this as needed
- **github.com/GambitBSM/gambit\_light** is a fork from our main repo **github.com/GambitBSM/gambit**
- Only some updates on **gambit** are relevant for **gambit\_light** (Core, ScannerBit, Printers, ...)
- The branch **gambit:gambit\_light\_sync** contains
  - a list of the **gambit** files that should be identical on **gambit\_light**
  - a GitHub Actions workflow that autogenerates pull requests on **gambit\_light** whenever some of these files are updated on **gambit**
- Workflow is currently set to sync **gambit:gambit\_light\_sync** → **gambit\_light:gambit\_light\_sync**
- *Example:*
  - On **gambit**: a merge **master** → **gambit\_light\_sync** will generate a pull request on **gambit\_light** with relevant updates for **gambit\_light:gambit\_light\_sync**
  - On **gambit\_light**: when PR is checked and merged, we can merge **gambit\_light\_sync** → **master**
- Using the branches **gambit\_light\_sync** is just to make syncing a bit more manual for now – in the future we can directly sync **gambit:master** → **gambit\_light:master**

- All files are either **fully synced** or **not synced at all** – no partial syncing
- Most files on **gambit** don't exist on **gambit\_light**
- Many files on **gambit\_light** (e.g. CMakeLists.txt) are completely detached from the corresponding file on **gambit**
- All such *non-synced* **gambit\_light** files can be modified directly on the **gambit\_light** repo
- For *synced* files where we need small modifications between **gambit** and **gambit\_light**, we can implement the changes on **gambit** with

```
#ifdef GAMBIT_LIGHT
...
#endif
```

- From the GitHub perspective, the file is fully synced between **gambit** and **gambit\_light**. (But it will generate different behaviour when compiled on **gambit\_light**.)

```
master - gambit_light / Utils / include / gambit / Utils / yaml_parser_base.hpp ↑ Top
Code Blame 149 lines (123 loc) · 4.11 KB Raw Download Edit View
83     template<typename TYPE, typename... args> TYPE getValue(args... keys) const
84     {
85         const YAML::Node node = getVariadicNode(keyValuePairNode, keys...);
86         if (not node) inFile_error().raise(LOCAL_INFO, "no inifile entry for [" + stringifyVariadic(keys...) + "];");
87         return NodeUtility::getNode<TYPE>(node);
88     }
89
90     template<typename TYPE, typename... args> TYPE getValueOrDefault(TYPE def, const args... keys) const
91     {
92         const YAML::Node node = getVariadicNode(keyValuePairNode, keys...);
93         if (not node)
94             {
95                 return def;
96             }
97         return NodeUtility::getNode<TYPE>(node);
98     }
99     /// {}
100
101     /// getters for model/parameter section
102     /// {}
103     template<typename TYPE> TYPE getModelParameterEntry(str model, str param, str key) const
104     {
105         if (not parametersNode[model][param][key]) inFile_error().raise(LOCAL_INFO, model + "." + param + "." + key + "not found in inifile.");
106         return parametersNode[model][param][key].as<TYPE>();
107     }
108     bool hasModelParameterEntry(str model, str param, str key) const;
109     /// Return list of model names (without "adhoc" model)
110     const std::set<str> getModelNames() const;
111     const std::vector<str> getModelParameters(str model) const;
112     /// {}
113
114     /// getter for options
115     const Options getOptions(str key) const;
116
117 protected:
118
119     /// Read in the actual YAML file
120     YAML::Node filename_to_node(str);
121
122     /// Do the basic parsing of the YAML file
123     void basicParse(YAML::Node, str);
124
125     /// Print a node to file
126     void printNode(YAML::Node, str, bool);
127
128 private:
129
130     YAML::Node YAMLNode;
131     YAML::Node keyValuePairNode;
132     YAML::Node parametersNode;
133     YAML::Node priorsNode;
134     YAML::Node printerNode;
135     YAML::Node scannerNode;
136     YAML::Node logNode;
137     #ifdef GAMBIT_LIGHT
138     YAML::Node userModelNode;
139     YAML::Node userPriorsNode;
140     YAML::Node userLogLikesNode;
141     #endif
142 };
143
144
145 }
146
147 }
148
149 #endif /* defined(__yaml_parser_base_hpp__) */
```



- This is an experiment – we'll have to tweak things to find the best system for easy development + minimal duplicated maintenance
- Revisit and evaluate at the next face-to-face GAMBIT meetings

## **5. Future plans**

- Code development
  - Keep testing the **gambit** → **gambit\_light** syncing
    - First big test: the PR for Python scanners
  - Polish the **gambit\_light** examples and documentation
  - PR for initial code updates on **gambit**
  - Document **gambit\_light** in the “GAMBIT 2” paper
  - Make **first public release**
  - Next: absorb GAMBIT bugfixes and improvements as they arrive (fast-slow, continual learning, ...)
  
- Some ongoing projects that will use GAMBIT-light:
  - With the nuclear physics group in Oslo: **unfolding of gamma spectra**  
*(Andreas Mjøs, Erlend Lima, Lasse Braseth, Ann-Cecilie Larsen, Morten Hjorth-Jensen + me)*
  - With the Norwegian Institute of Public Health: **optimisation of Monte Carlo simulations of disease spread**  
*(Ida-Marie Johanson, Jørgen Midtbø, Francesco Di Ruscio, Yat Hin Chan, Birgitte Freiesleben de Blasio + Are and me)*



# Computador Inteligente

**Letras**

- 1) Aprenda as Letras
- 2) Escrivendo Letras
- 3) Preenche as Letras
- 4) Jogo de Letras
- 5) Organize Letras

**Música**

- 11) Aprenda as Notas
- 17) Preenche as Notas
- 18) Tempo de Música
- 19) Jogo de Canção
- 20) Abecedário

**Palavras**

- 4) Aprenda as Palavras
- 7) Identifique
- 8) Preenche as Palavras
- 9) Substitua as Palavras
- 26) Corrigir as Palavras

**Jogos**

- 21) Jogo 1
- 22) Jogo 2
- 23) Jogo 3
- 24) Jogo 4
- 25) Jogo 5

**Aritmética**

- 15) Aprenda as Números
- 12) Preenche os Números
- 13) Organize os Números
- 14) Adição
- 16) Subtração



25  
FUNÇÕES INTERESSANTES

