

Live coding bringing scientific computing to life

Dr. Ben Swift

Australian National University

ben.swift@anu.edu.au











THE AUSTRALIAN NATIONAL UNIVERSITY



Live (interactive) programming

tight feedback loop between programmer **action** and system **reaction**, between **intuition** and **validation**

not so common in HPC (for a variety of socio-technical reasons) although people are starting to talk about real-time/online—streaming, in-situ analysis etc.



(lisp)

Extempore

<https://github.com/digego/extempore>

Andrew
Sorensen



OSCON
OPEN SOURCE CONVENTION

native code
efficiency

+

dynamic
interactivity

JIT-compiled on the
node with LLVM

no-restart hot-swapping
of functions, variables

From multimedia arts to HPC

music = real-time DSP

- need **efficiency**: native code, no interpreter
- need **determinism**: no garbage collector
- need **interoperability**: native interface to C/Fortran lib

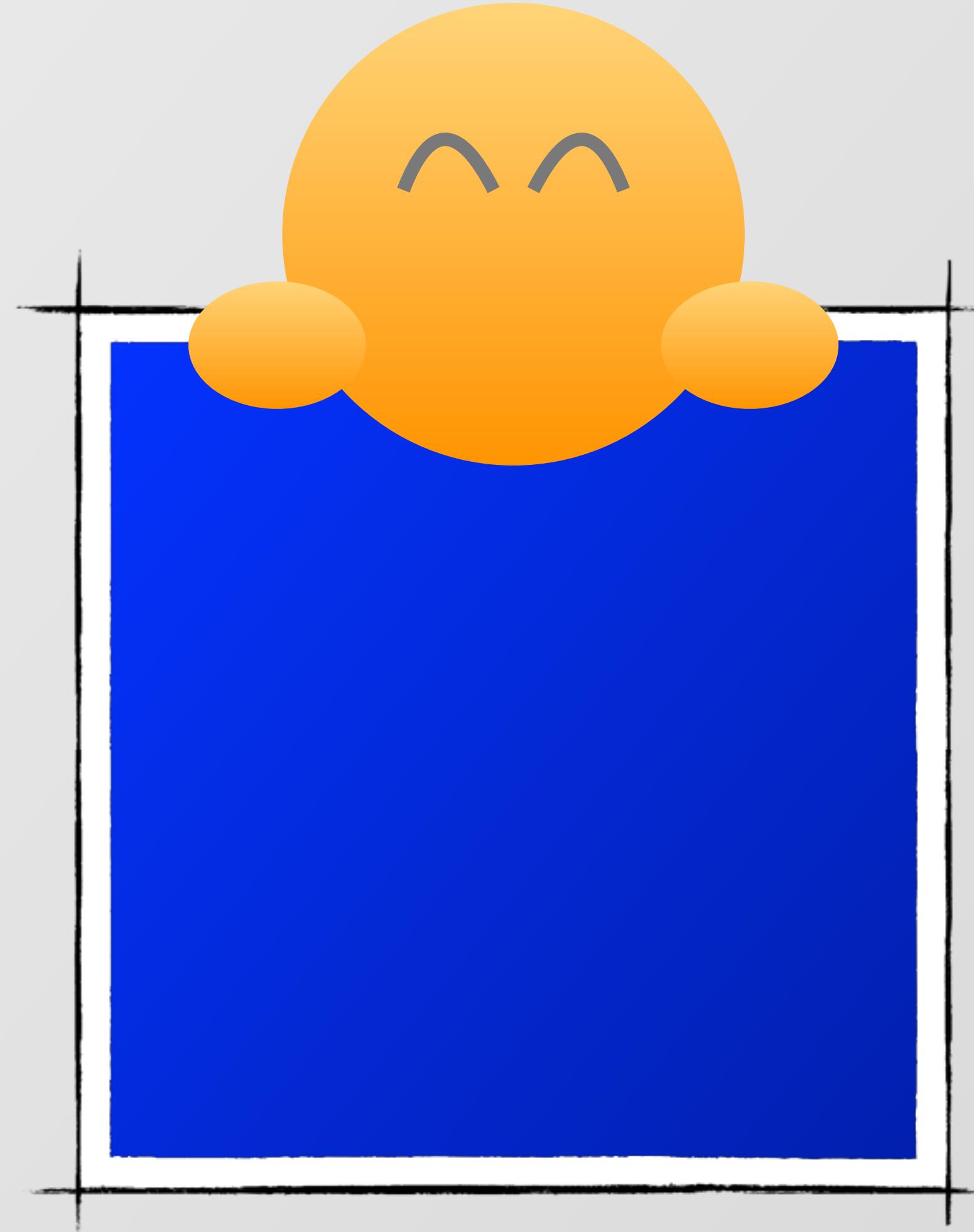
some overlap with scientific & numerical computing



(lisp)

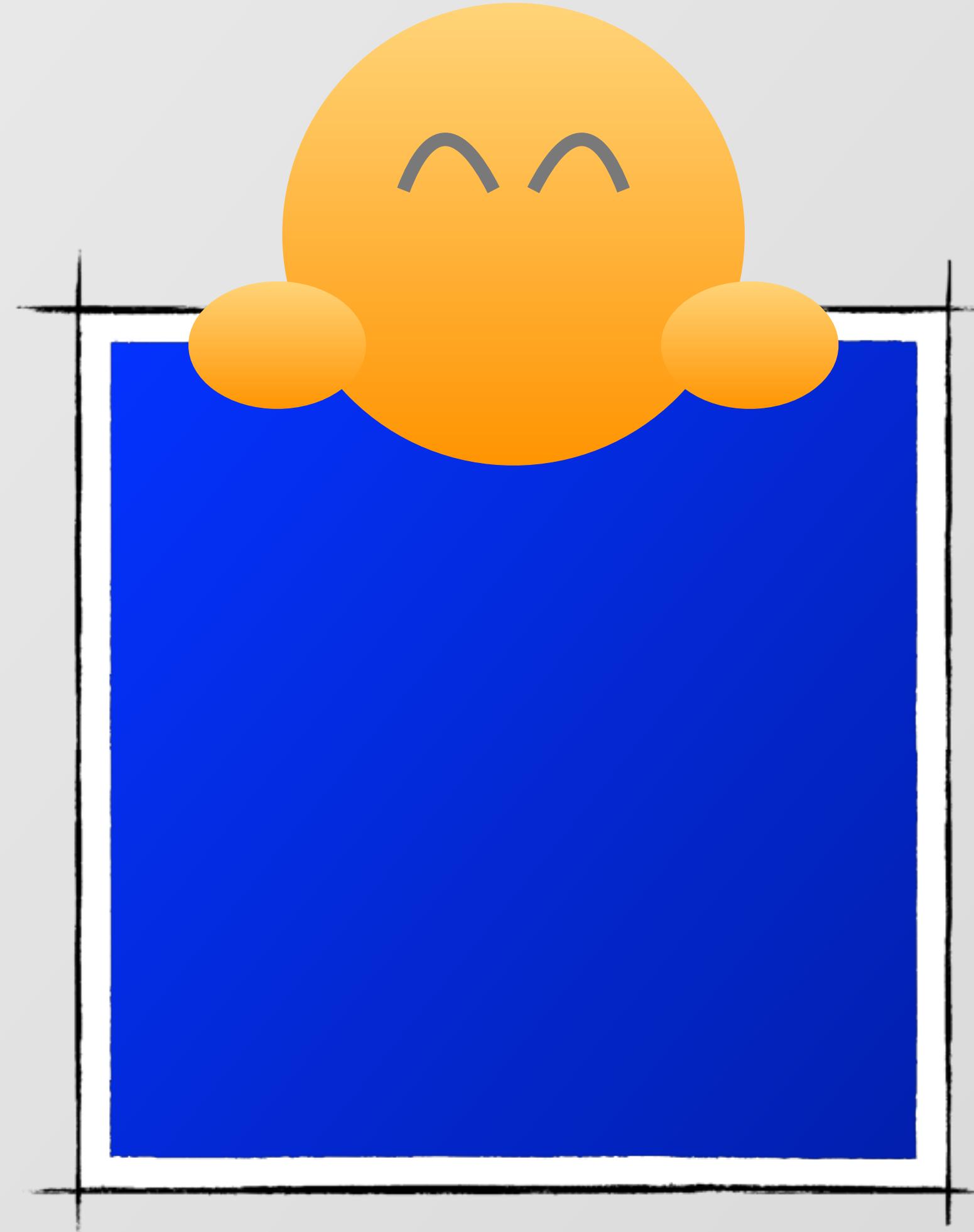
but what does *liveness*
mean in HPC?

traditional HPC workflow



source code

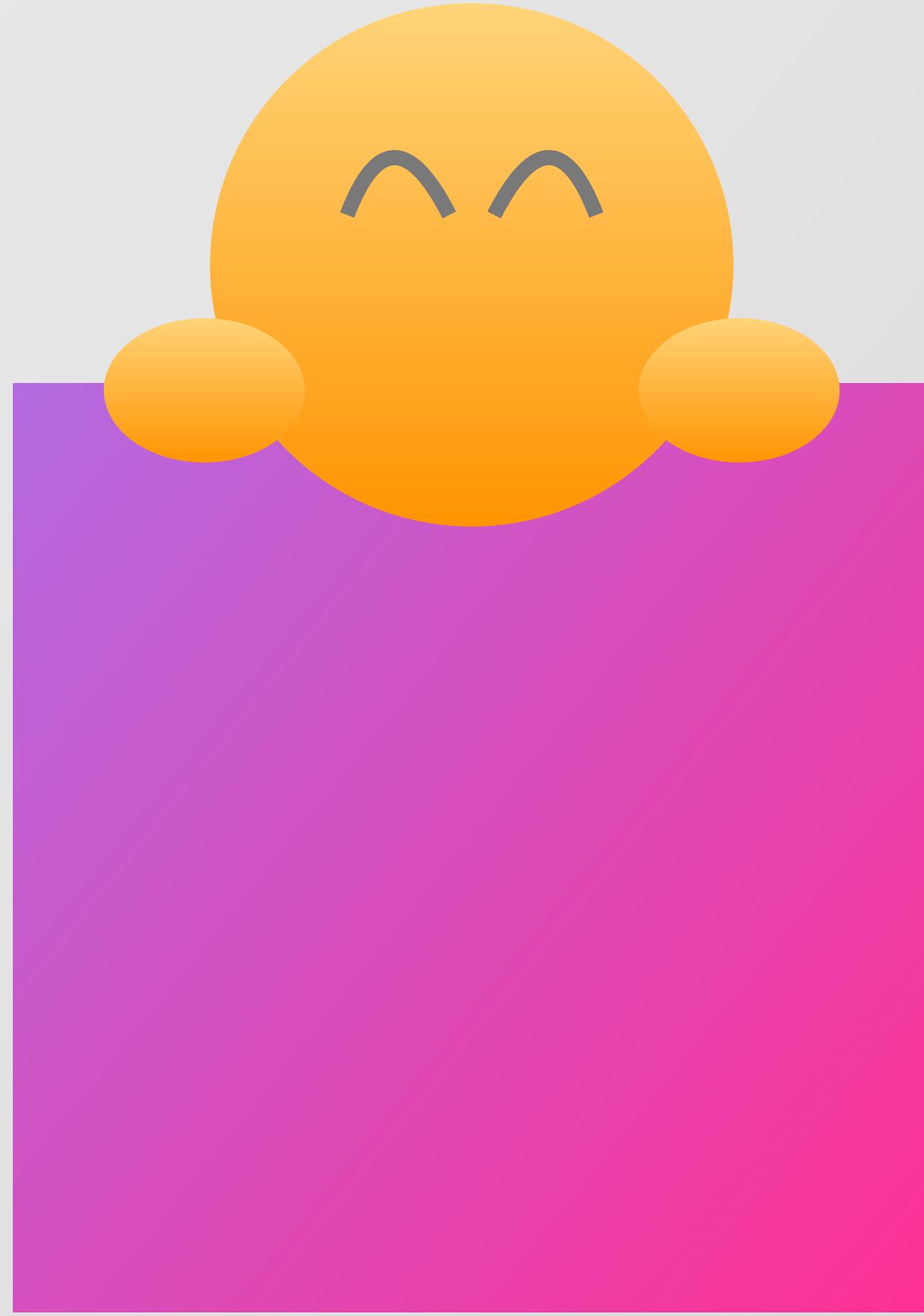
compile



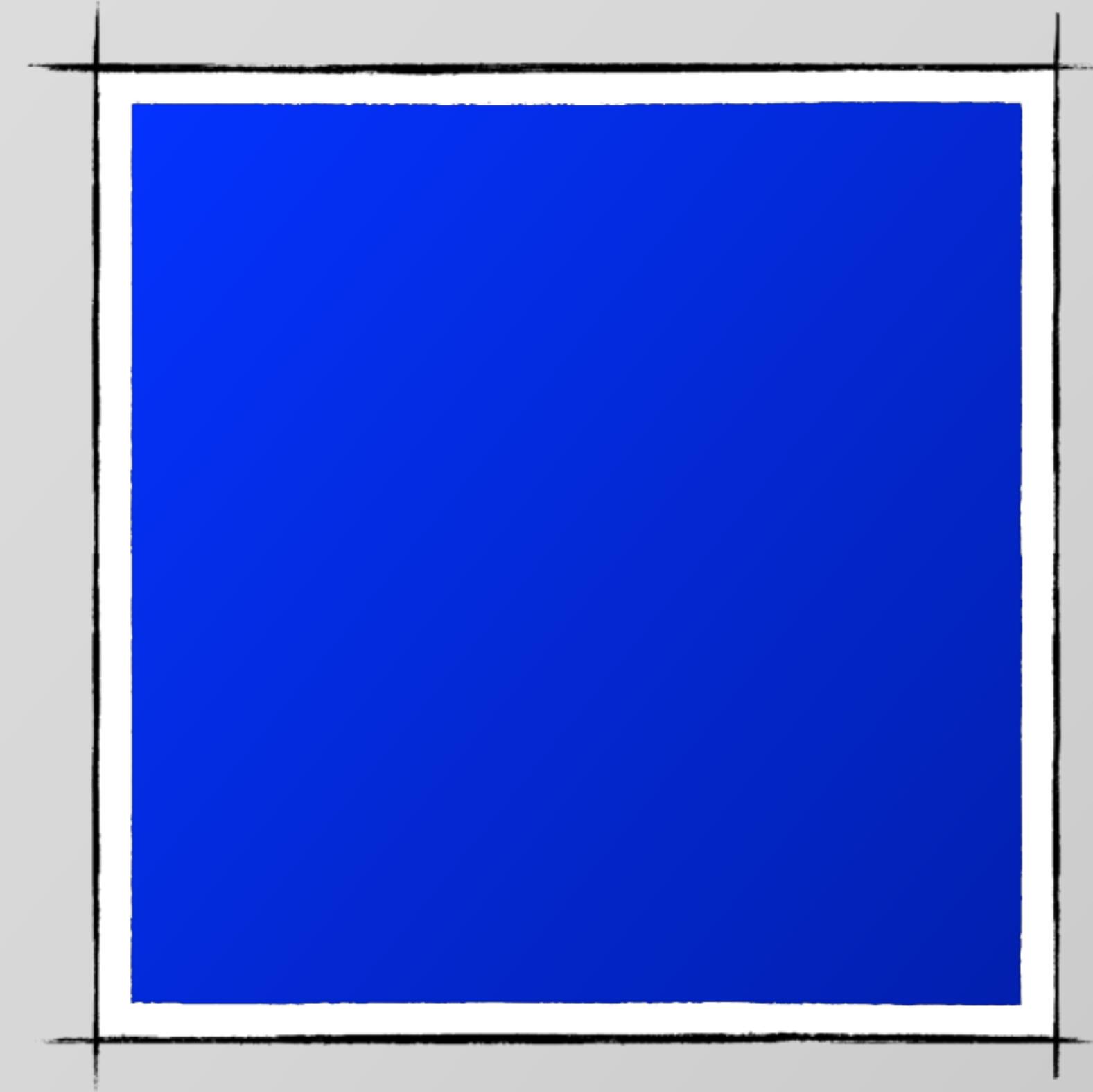
source code

run

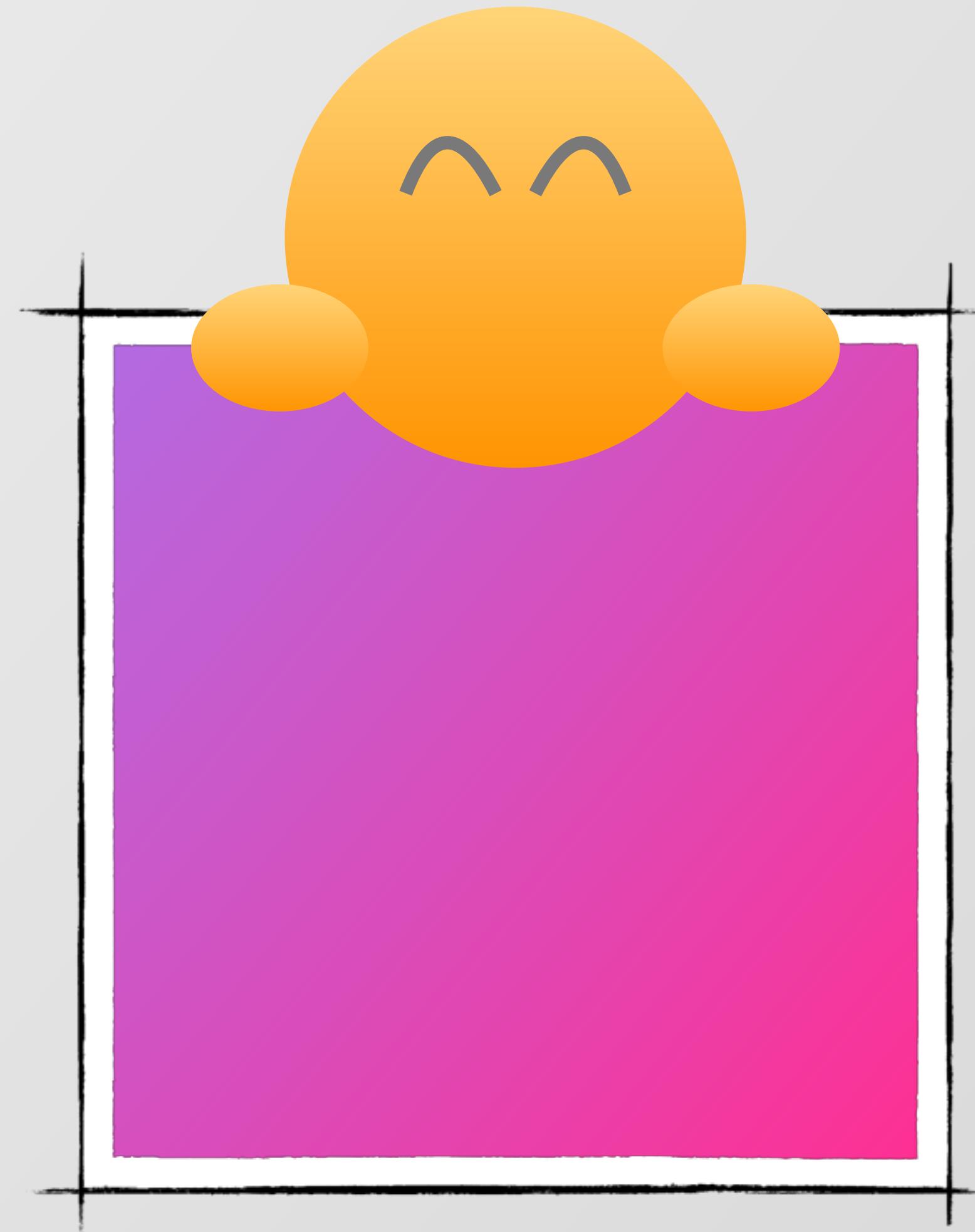
running program



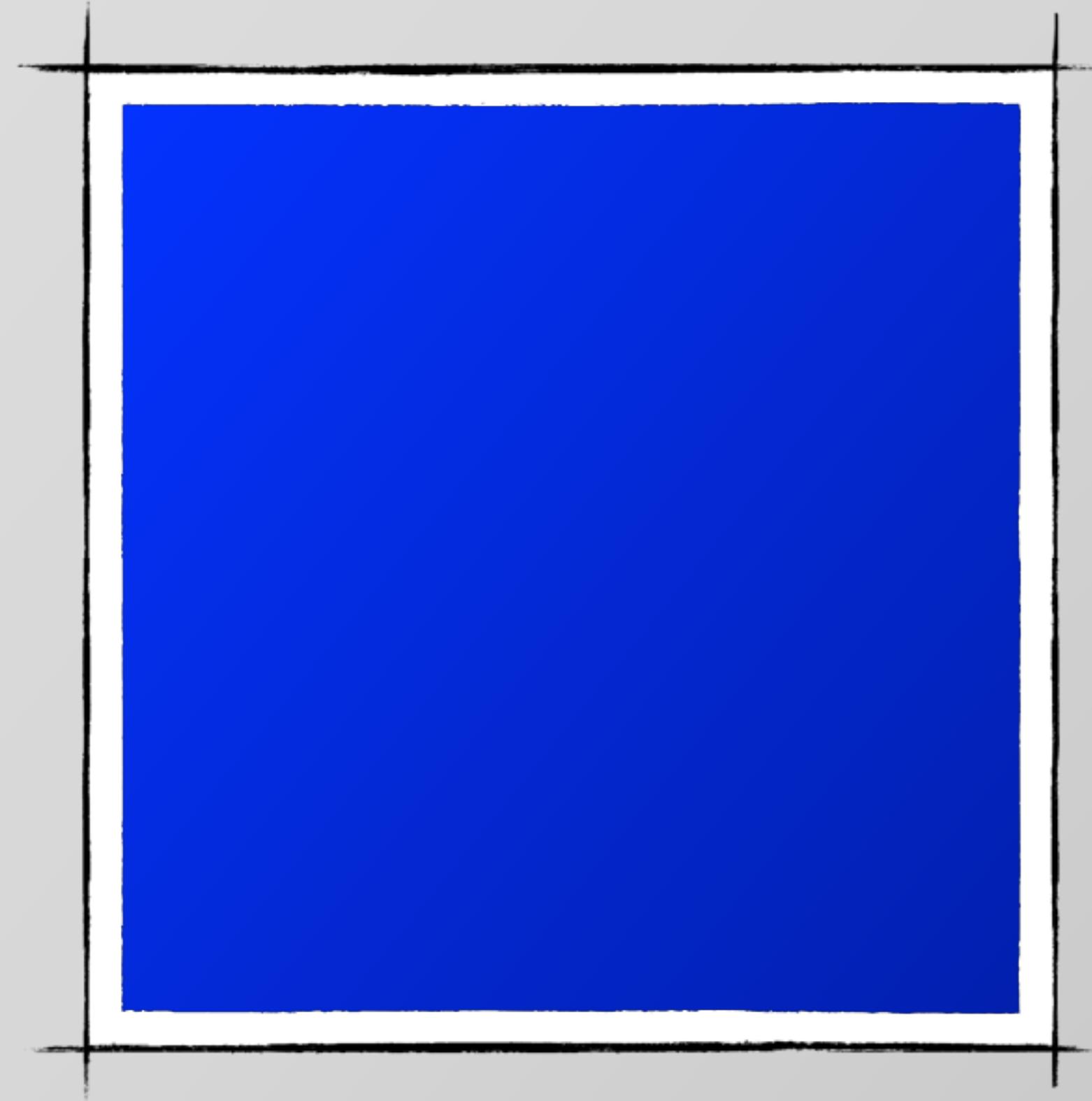
source code



running program



source code

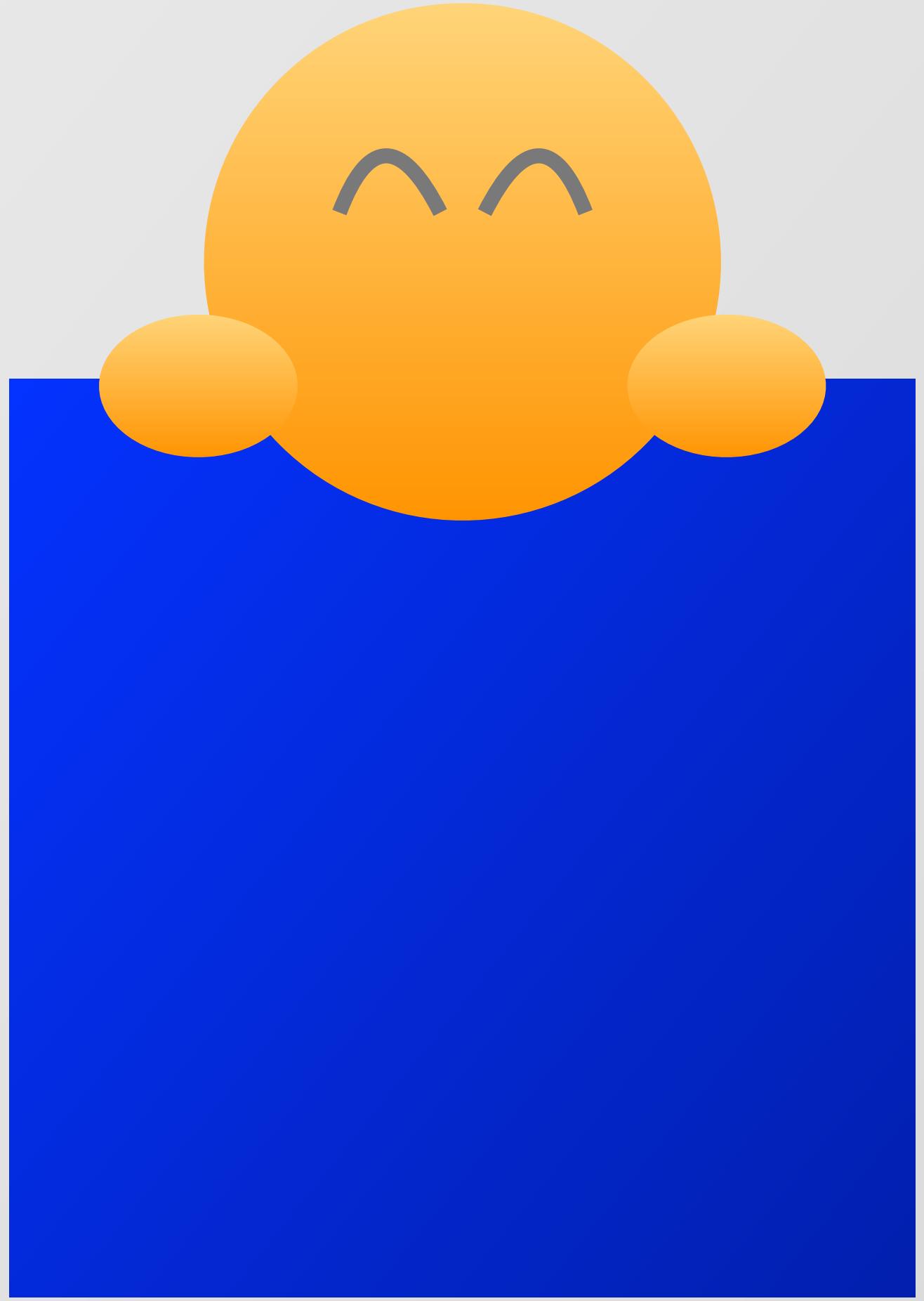


running program

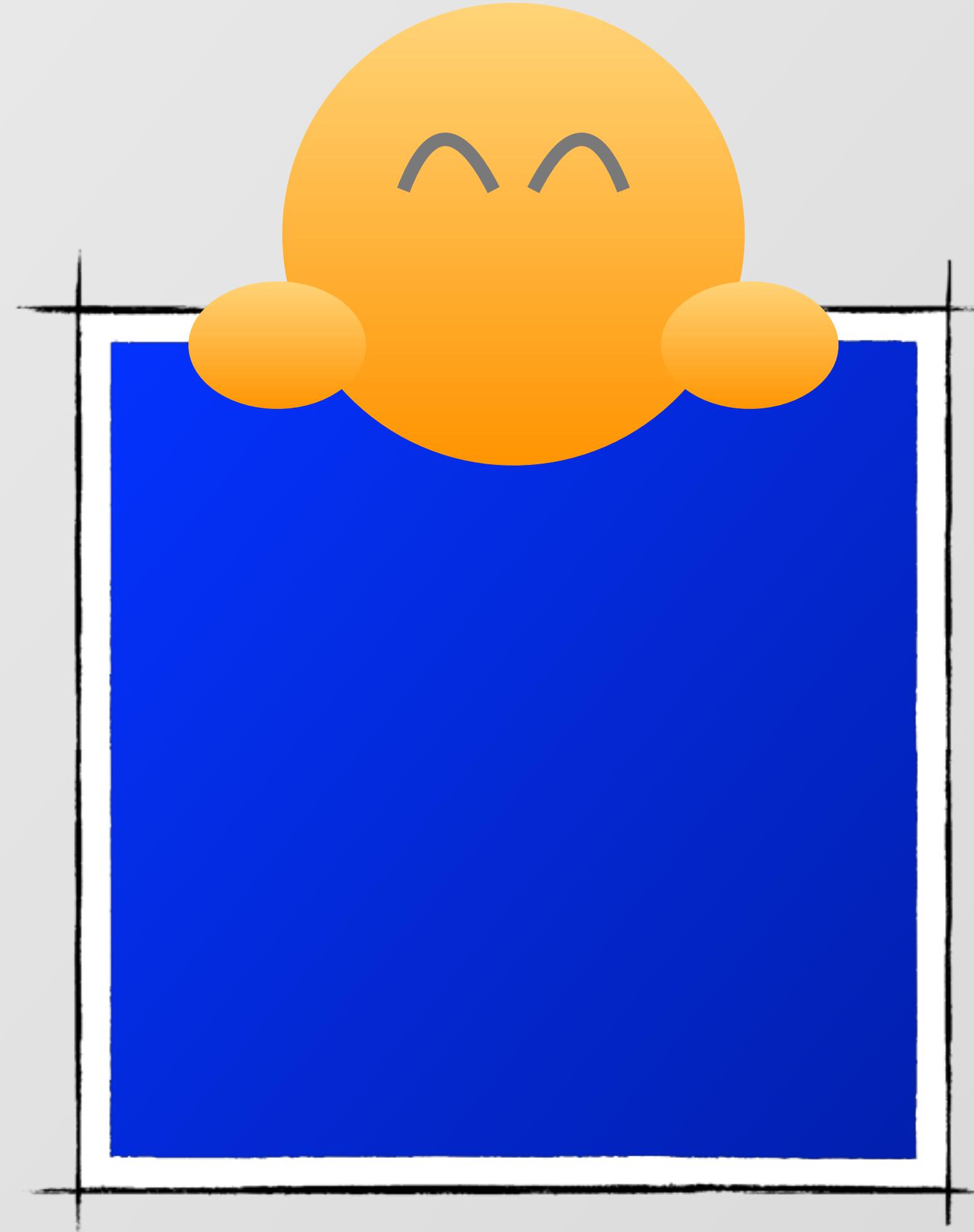
this can be tedious,
especially in the early stages

and we discard
useful program state

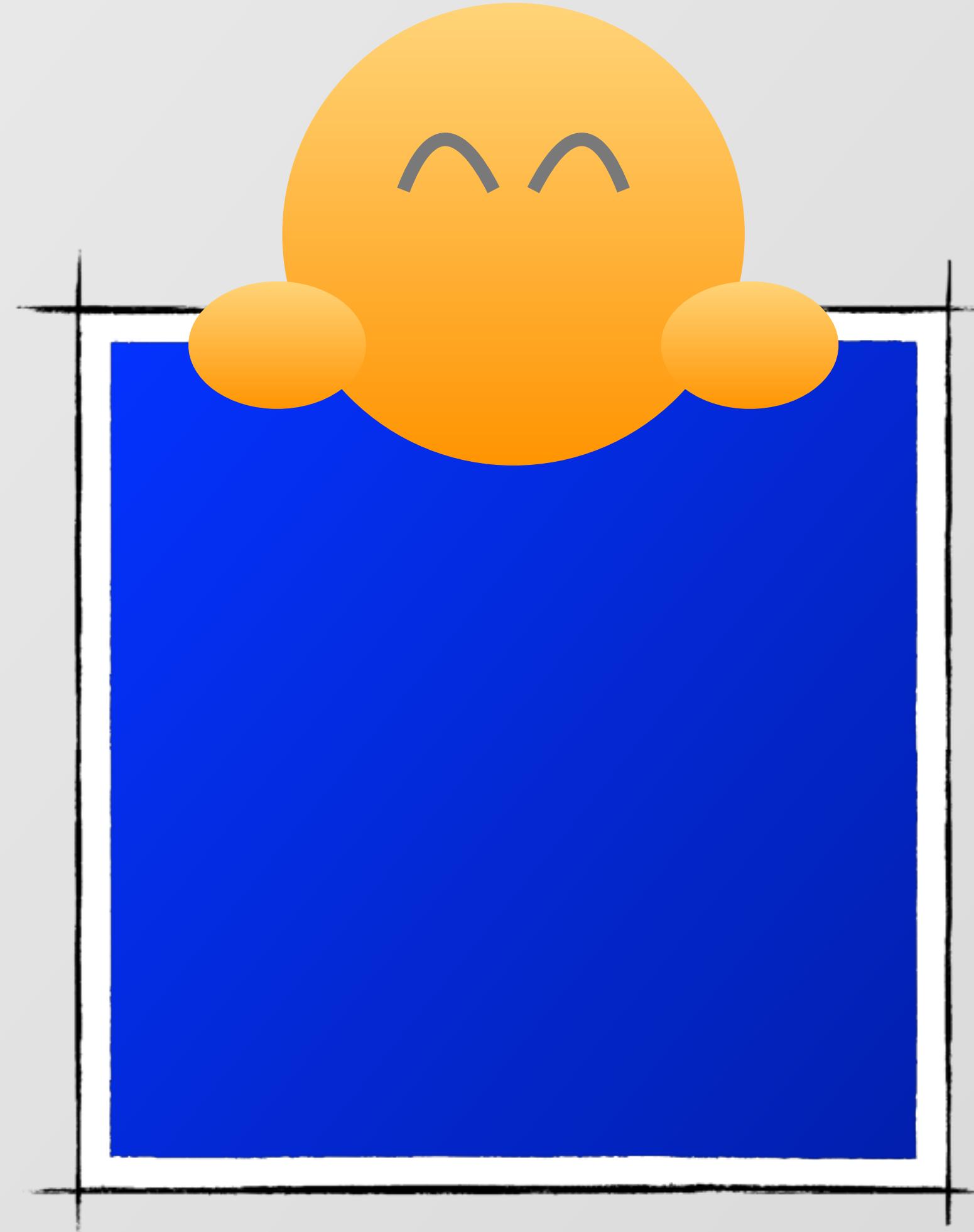
live programming
workflow



source code

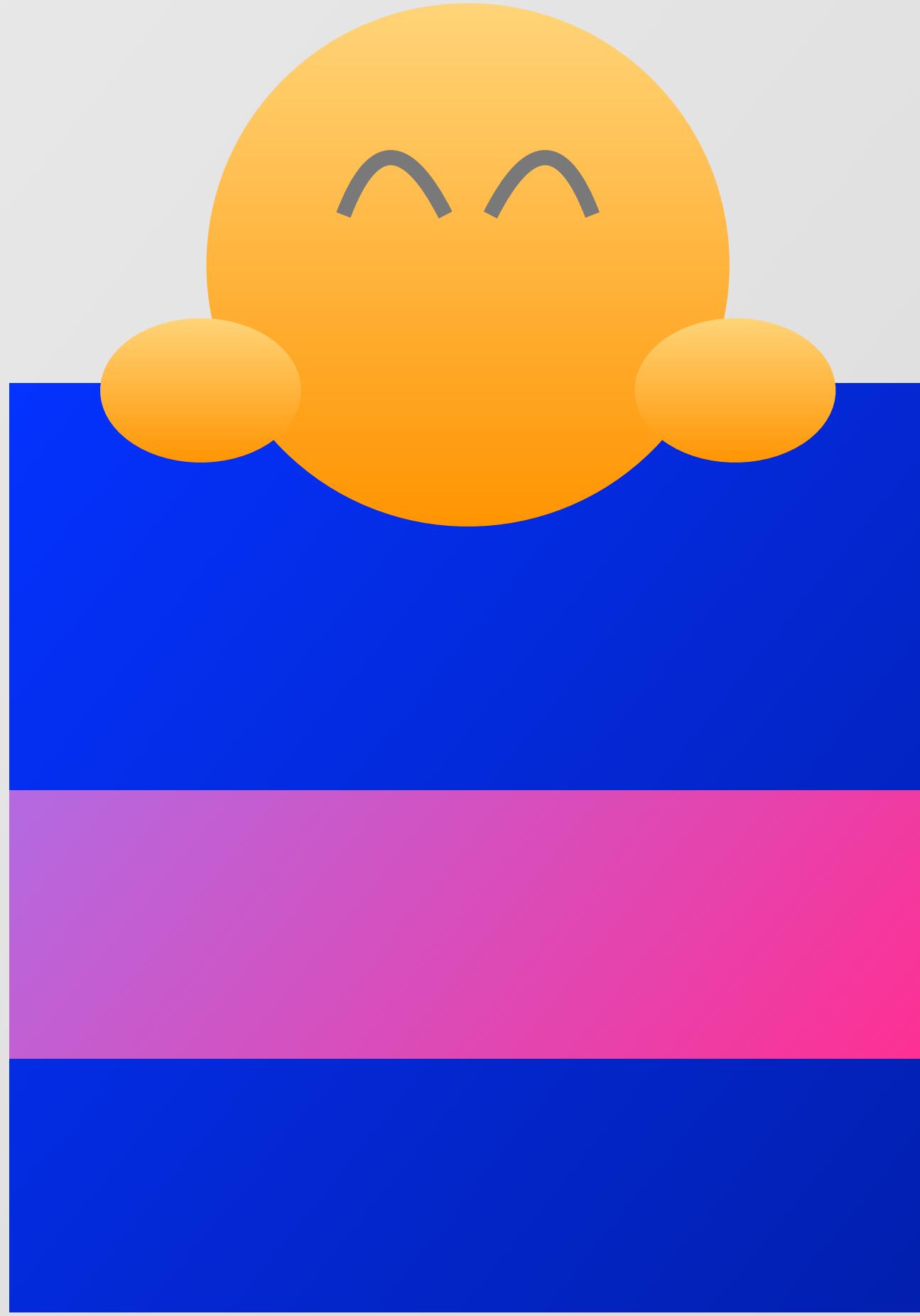


source code

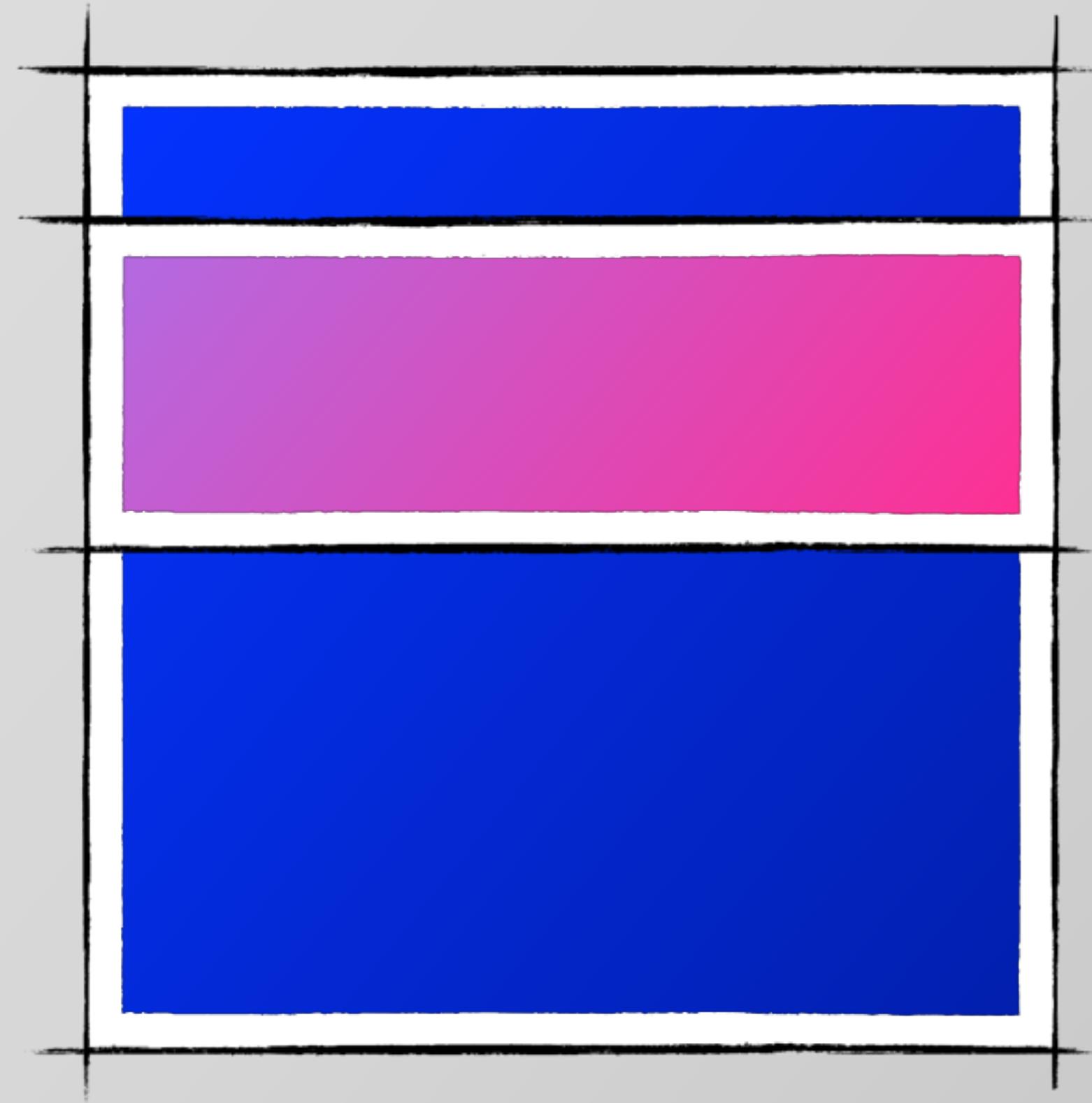


source code

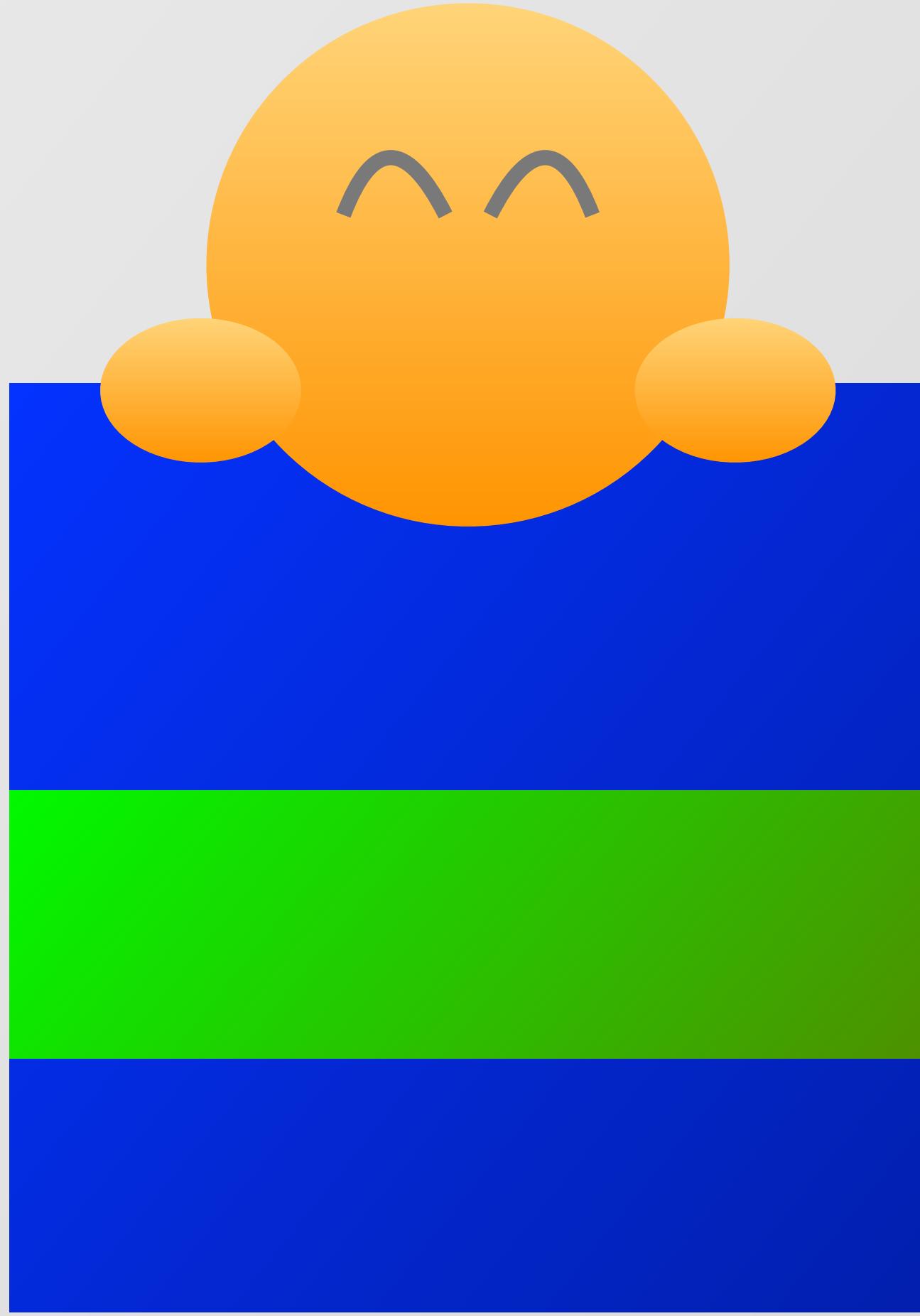
running program



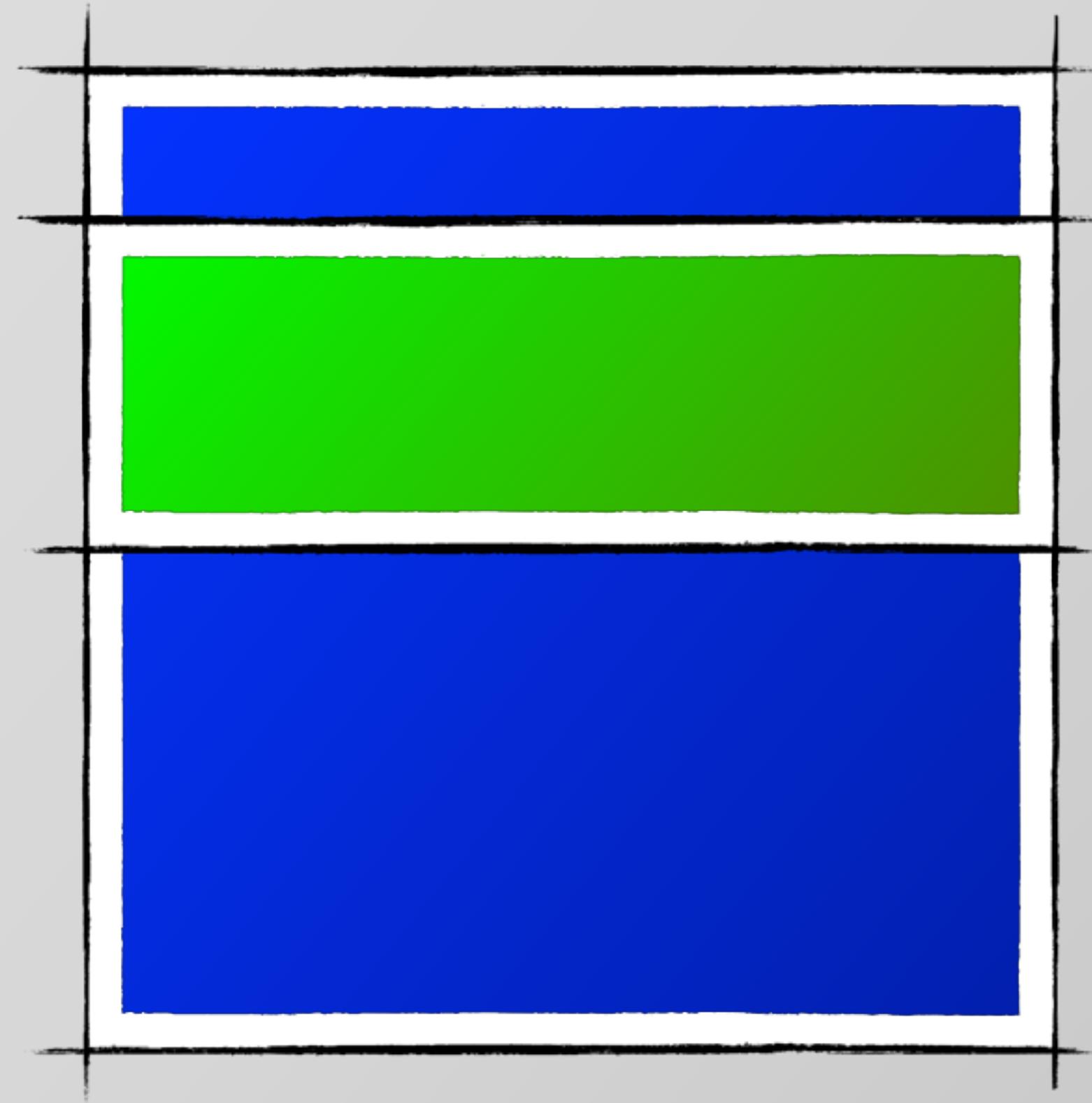
source code



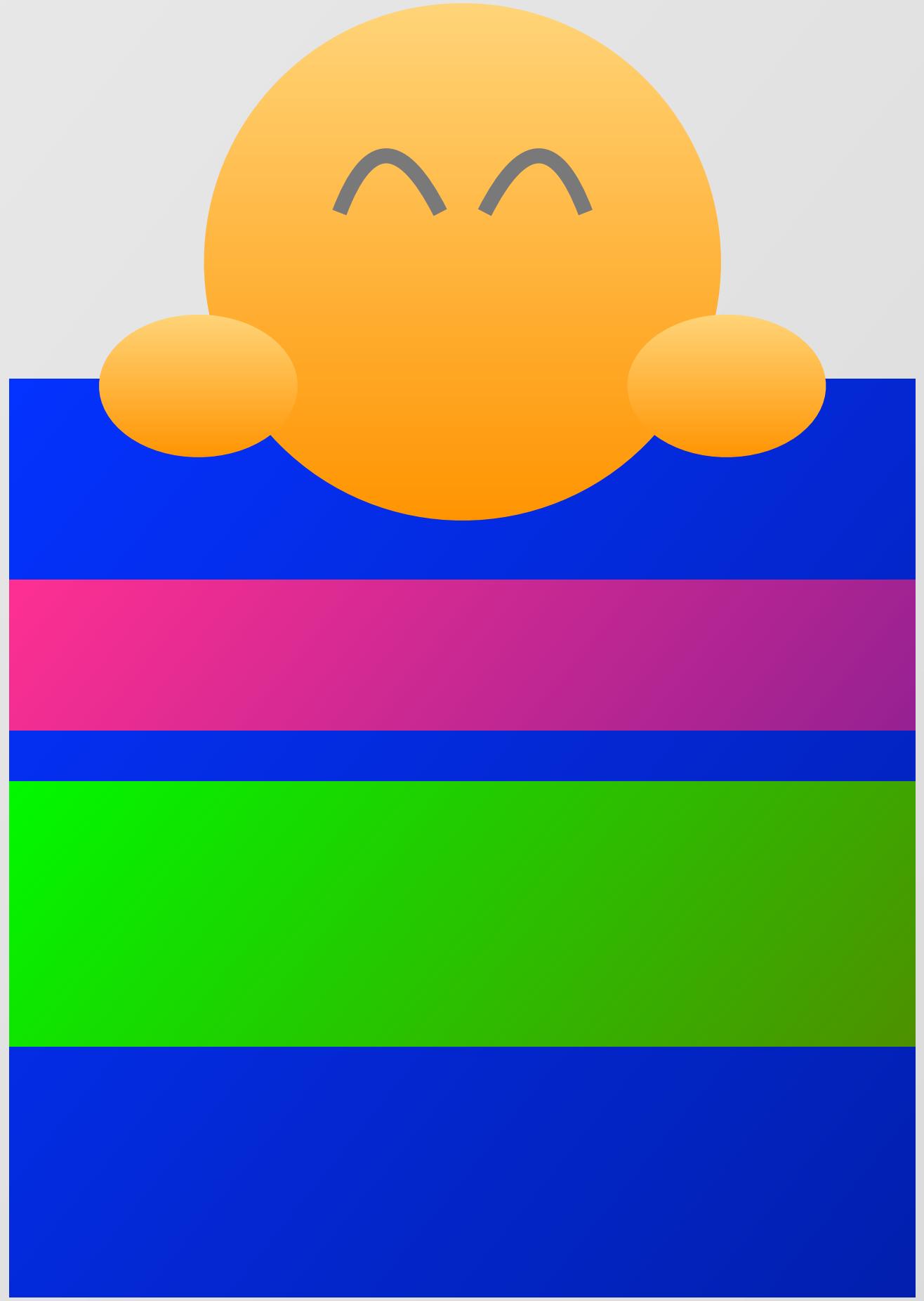
running program



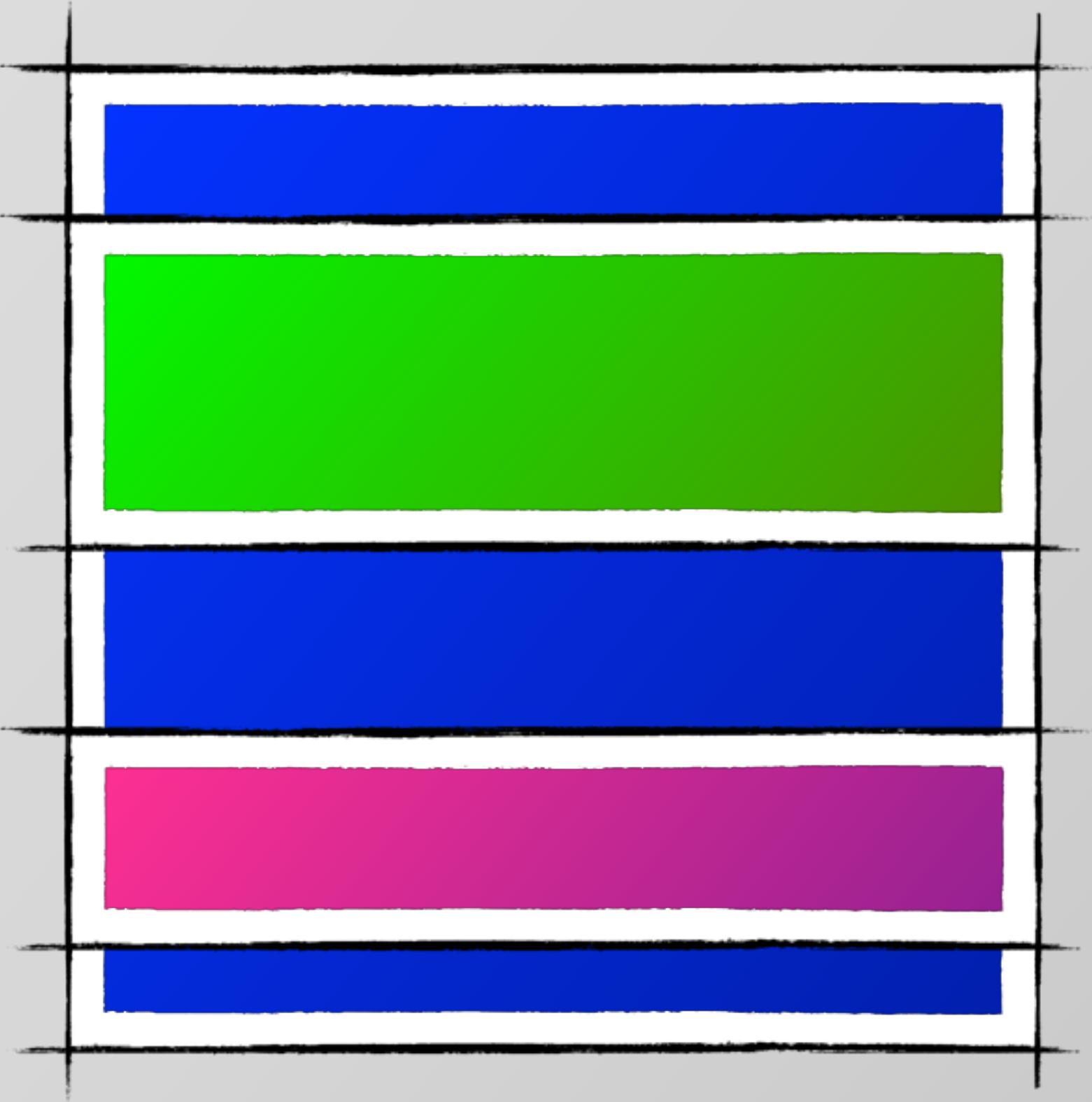
source code



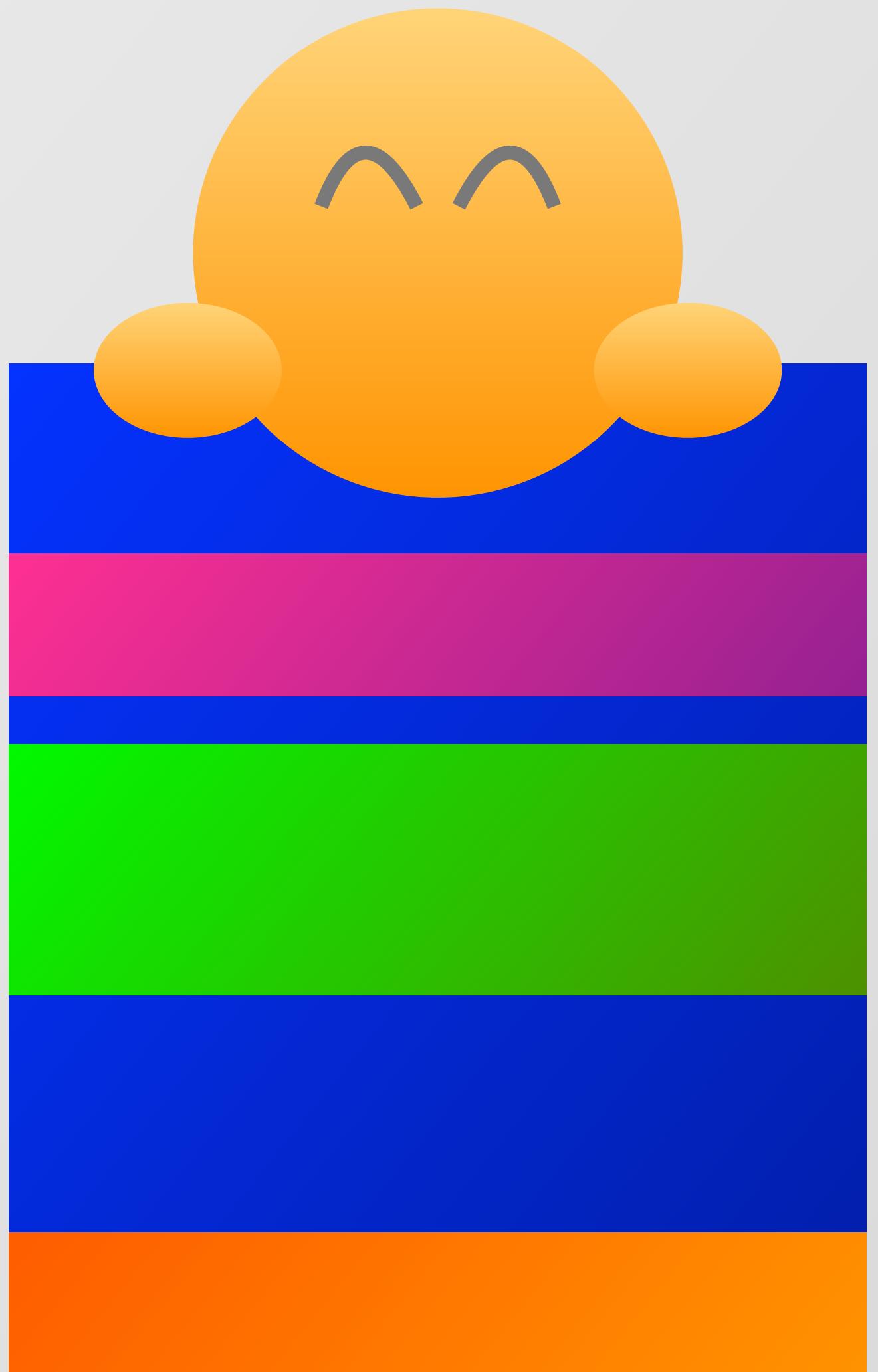
running program



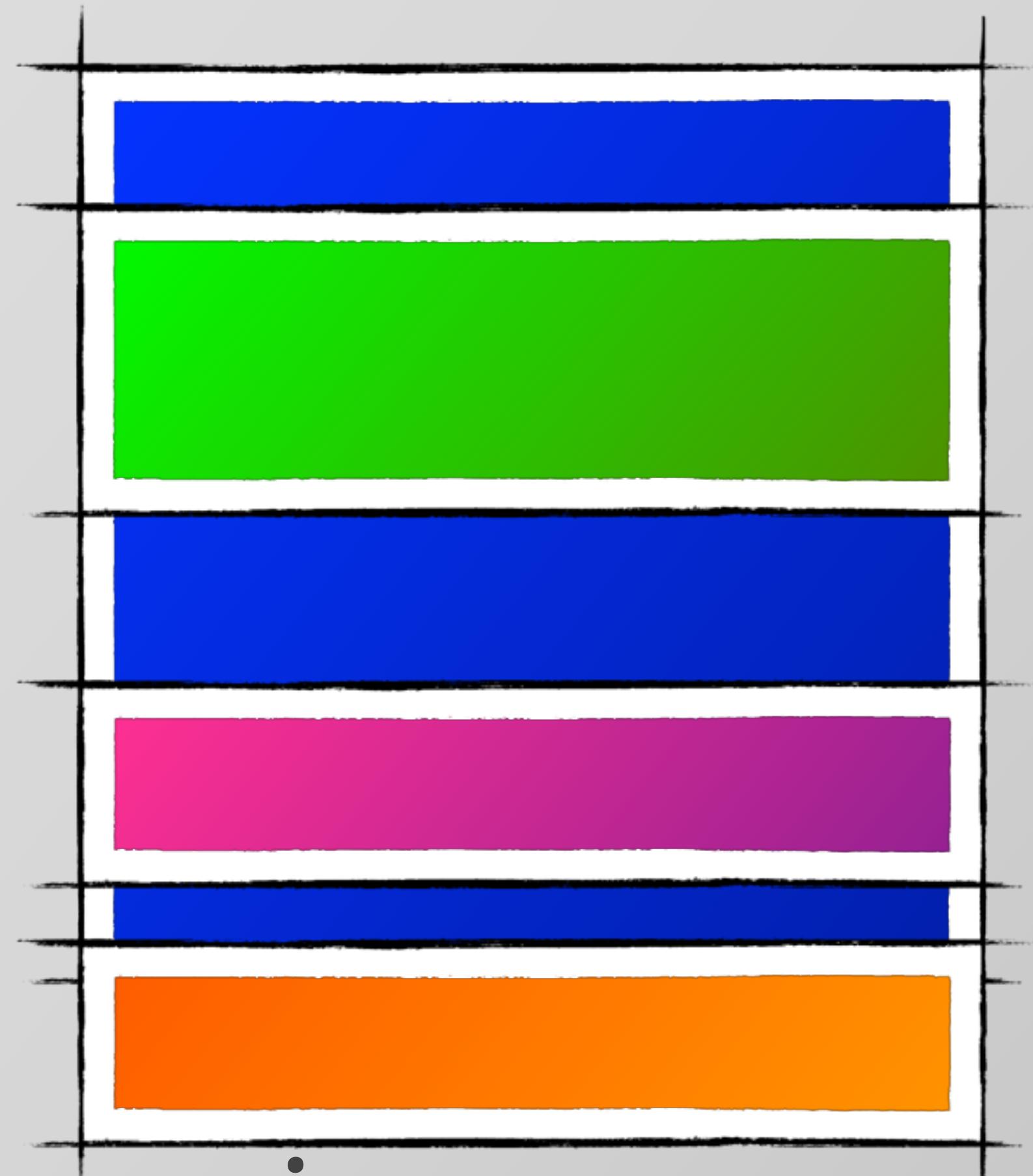
source code



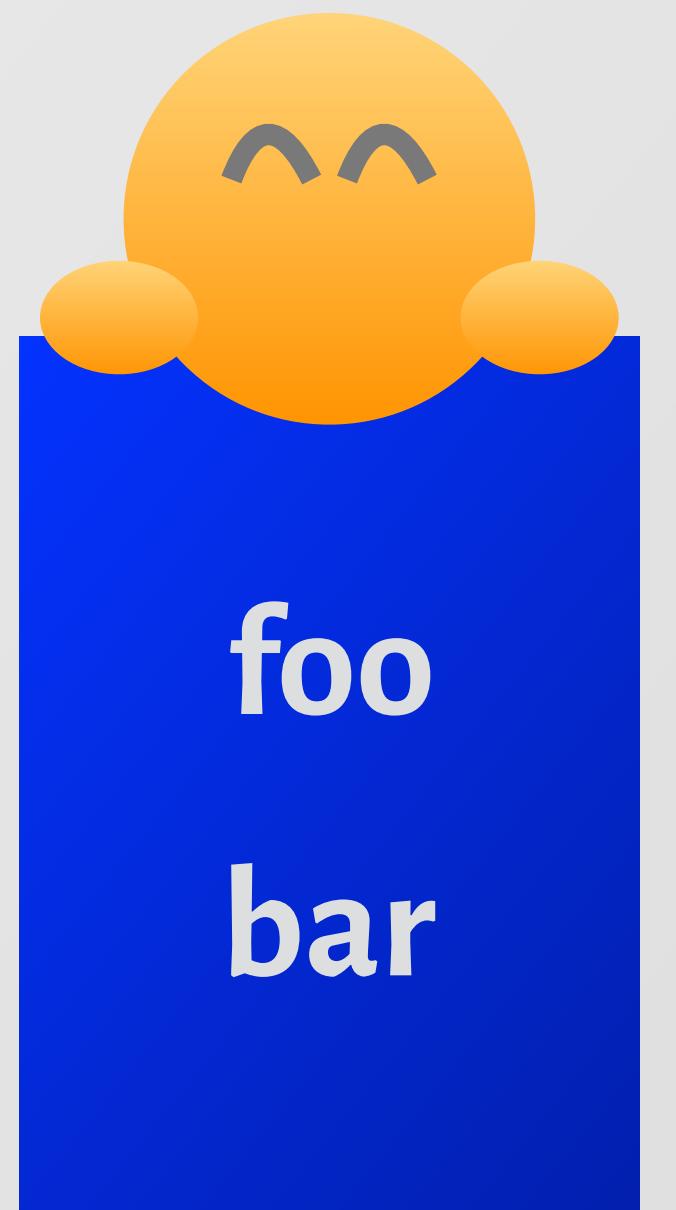
running program



source code



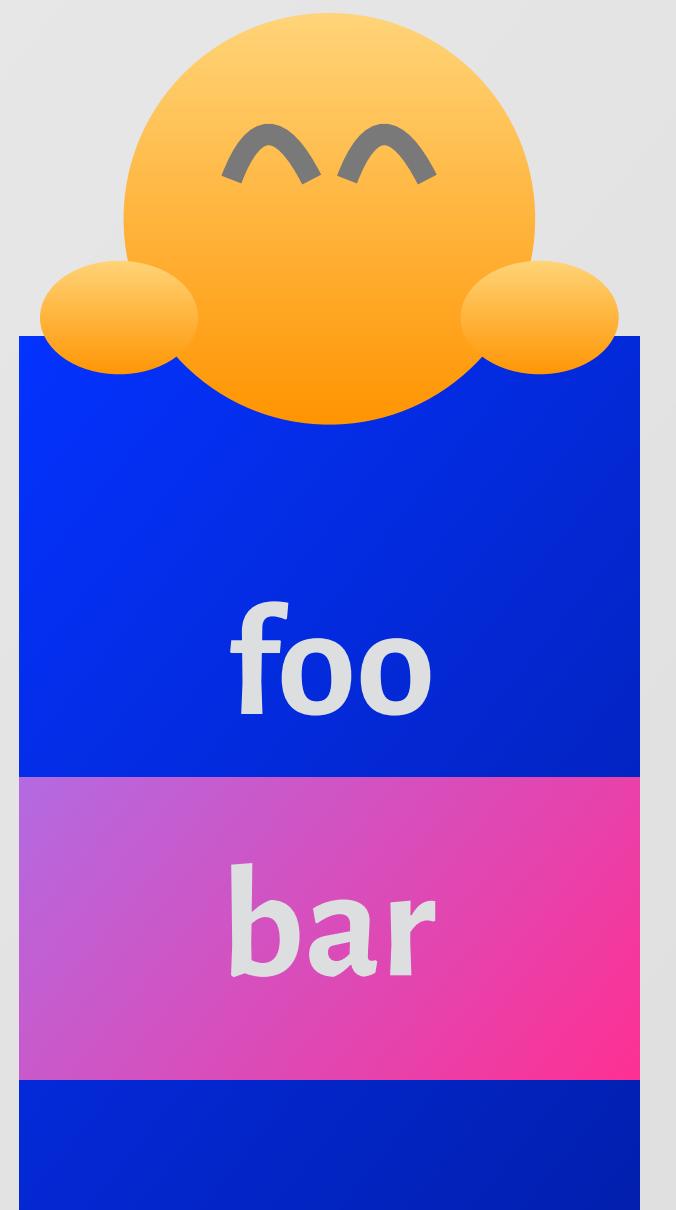
running program



source code



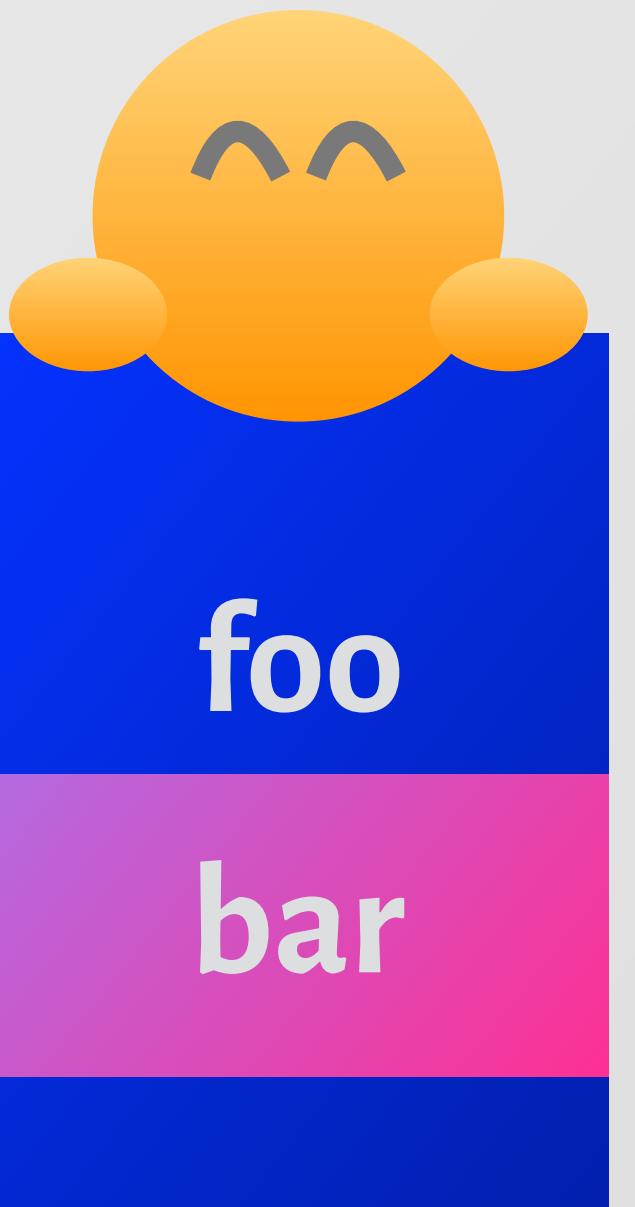
running program



source code



running program



source code

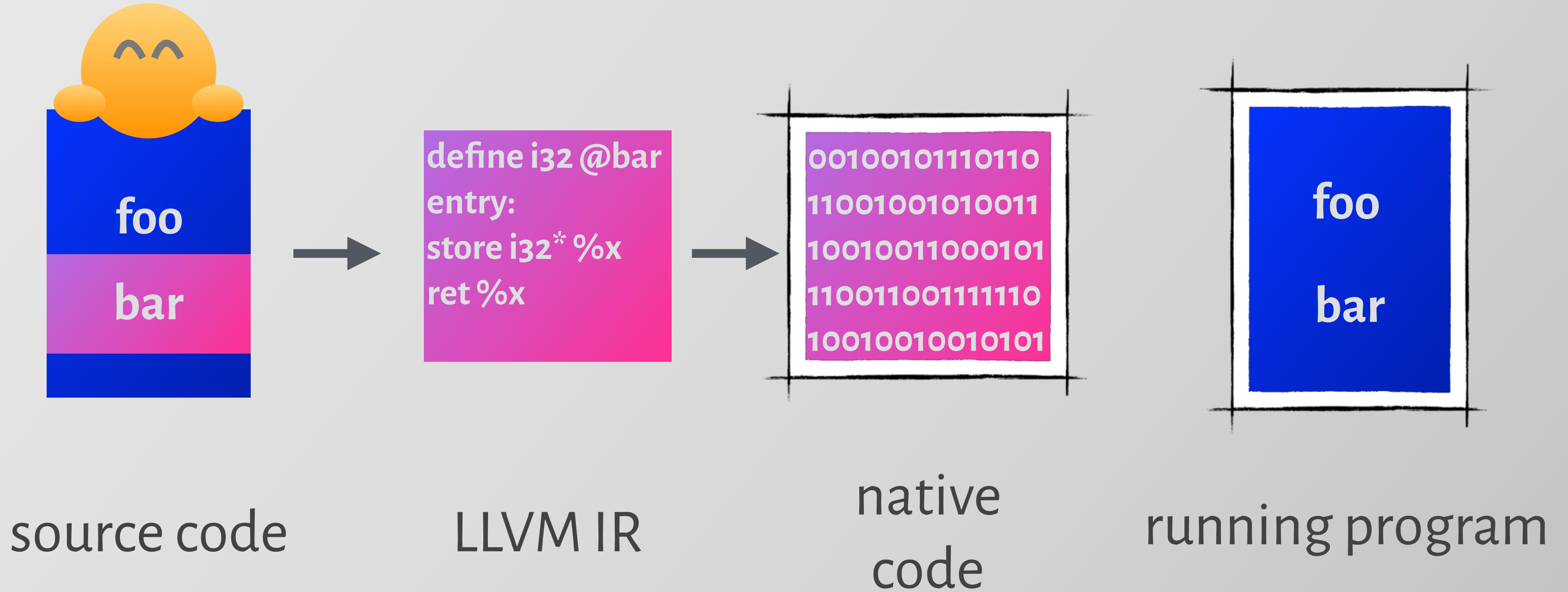


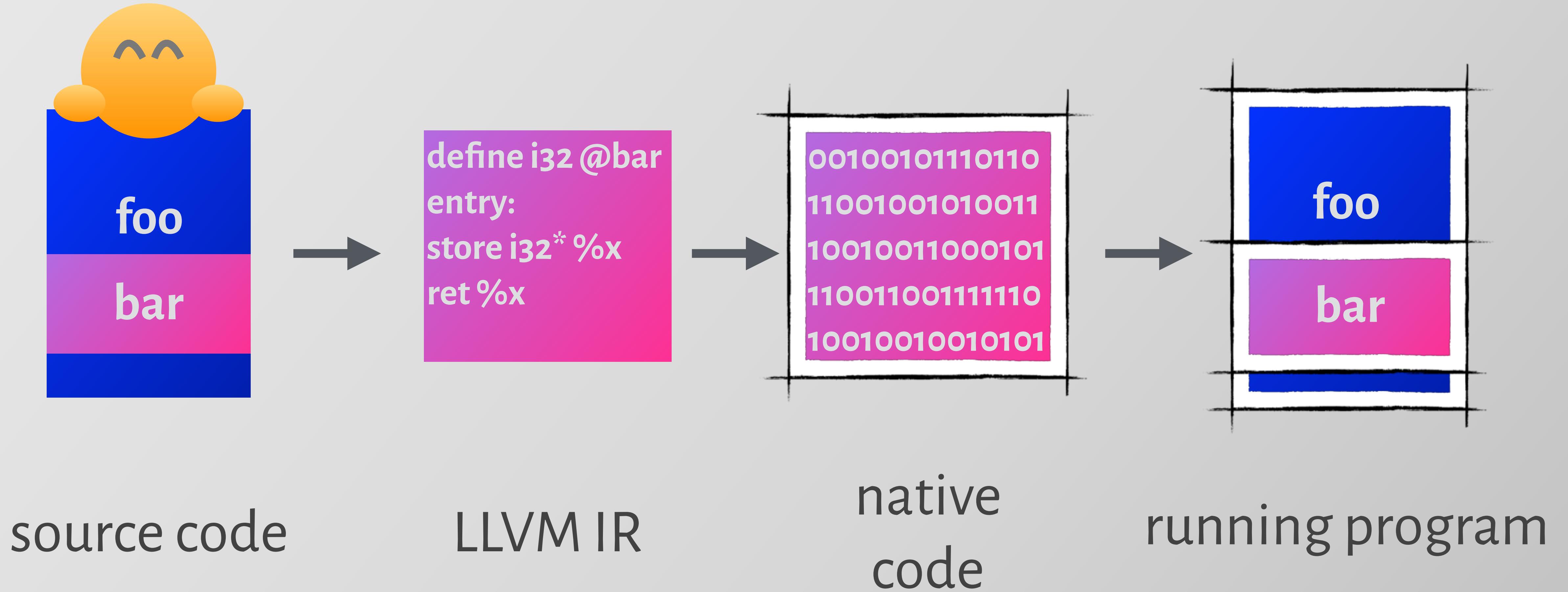
```
define i32 @bar  
entry:  
store i32* %x  
ret %x
```

LLVM IR



running program





But our HPC codes work pretty well...

Extempore is ABI compatible with C/Fortran, so you can

- call compiled C/Fortran routines from a shared library
- read and write directly to memory (pointers)
- use e.g. MPI for message passing, OpenMP for threading

*...all with **live** hot-swapping*

Live PIC: a case study

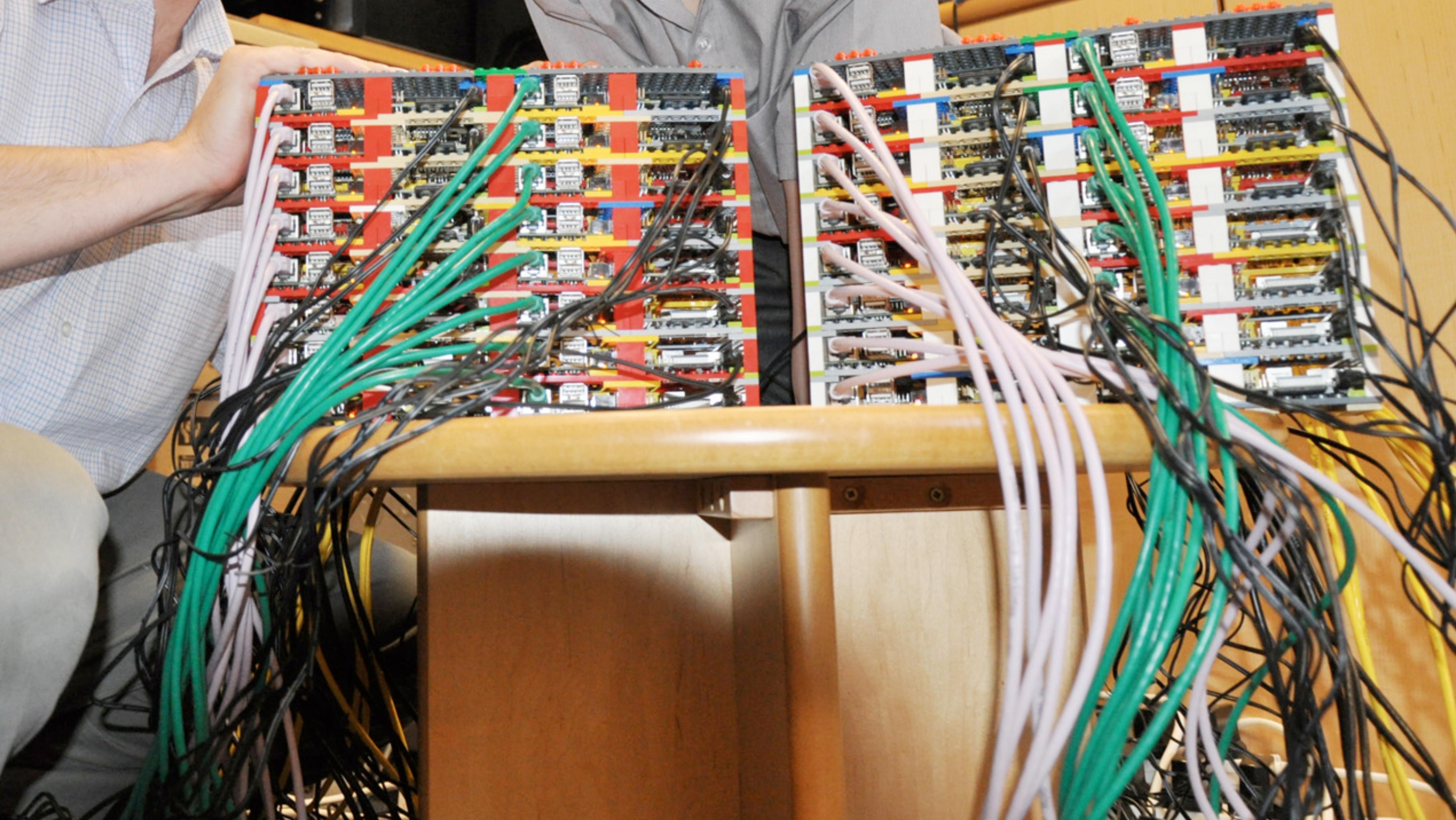
2½D electromagnetic PIC skeleton code

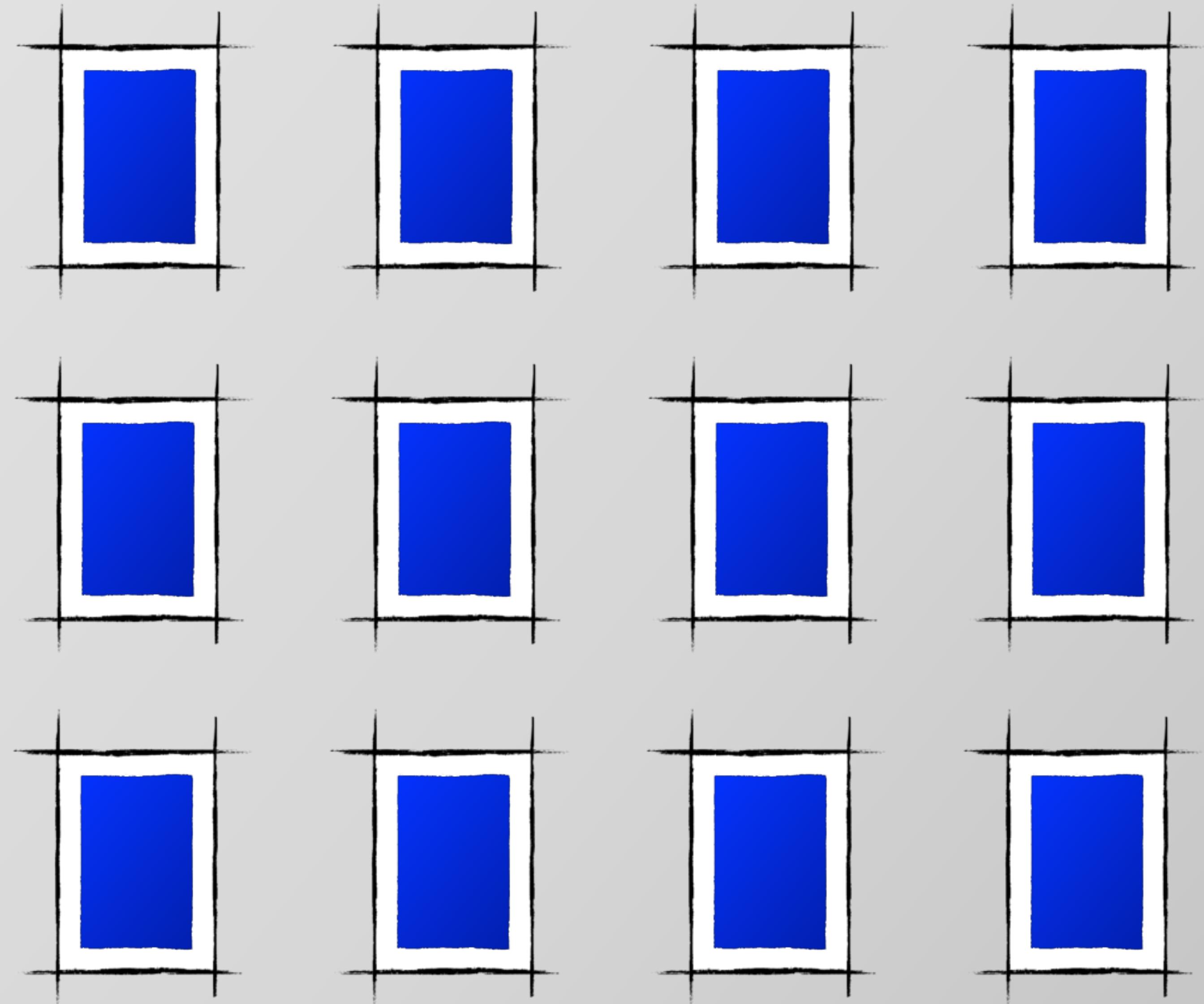
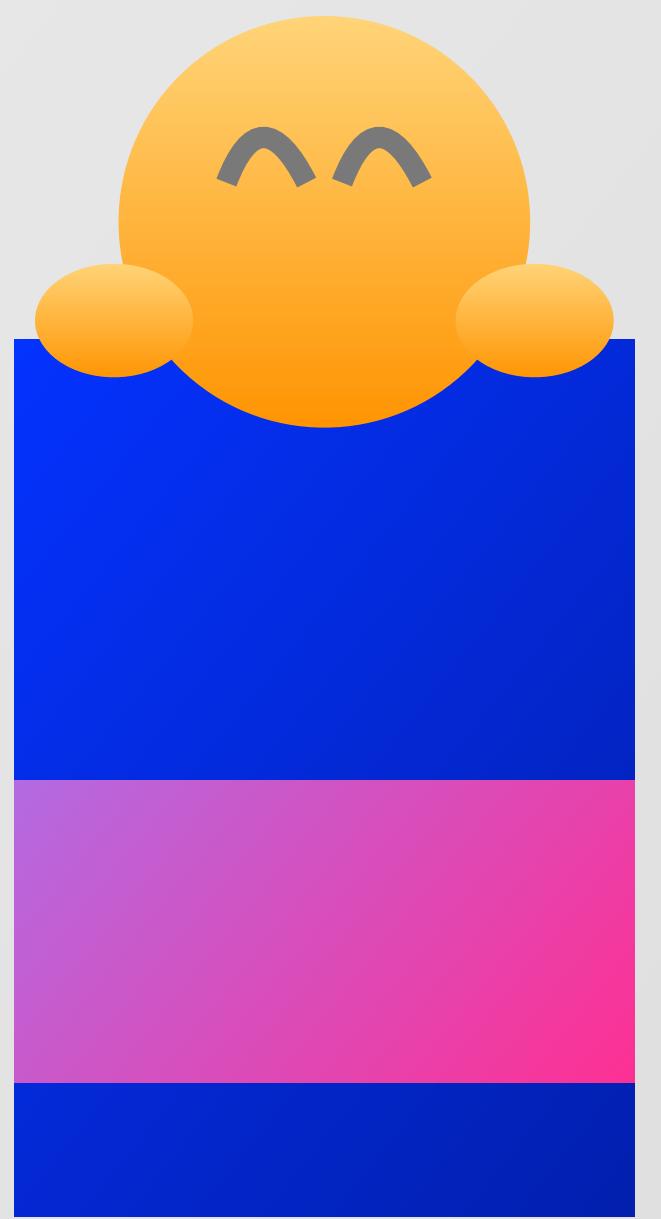
running **live**, distributed memory (MPI)

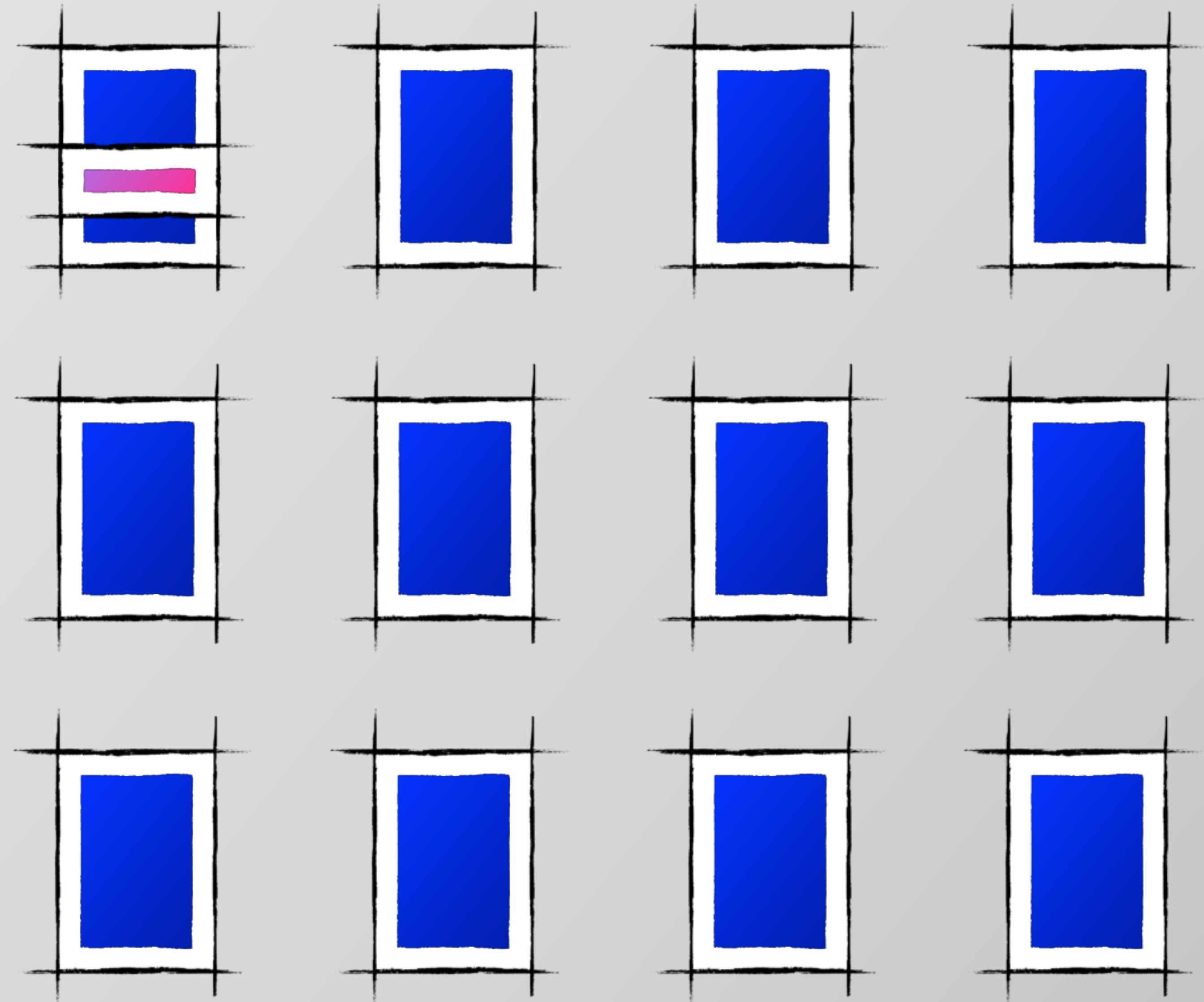
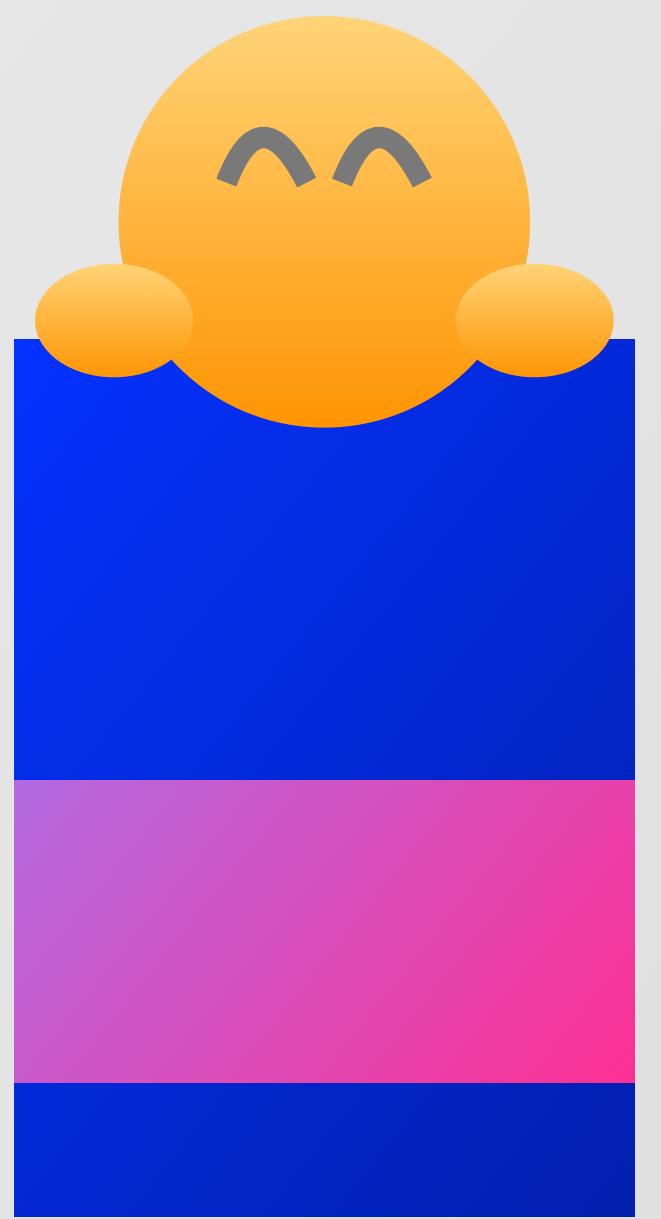
on my laptop (for practical reasons) but could be on a cluster

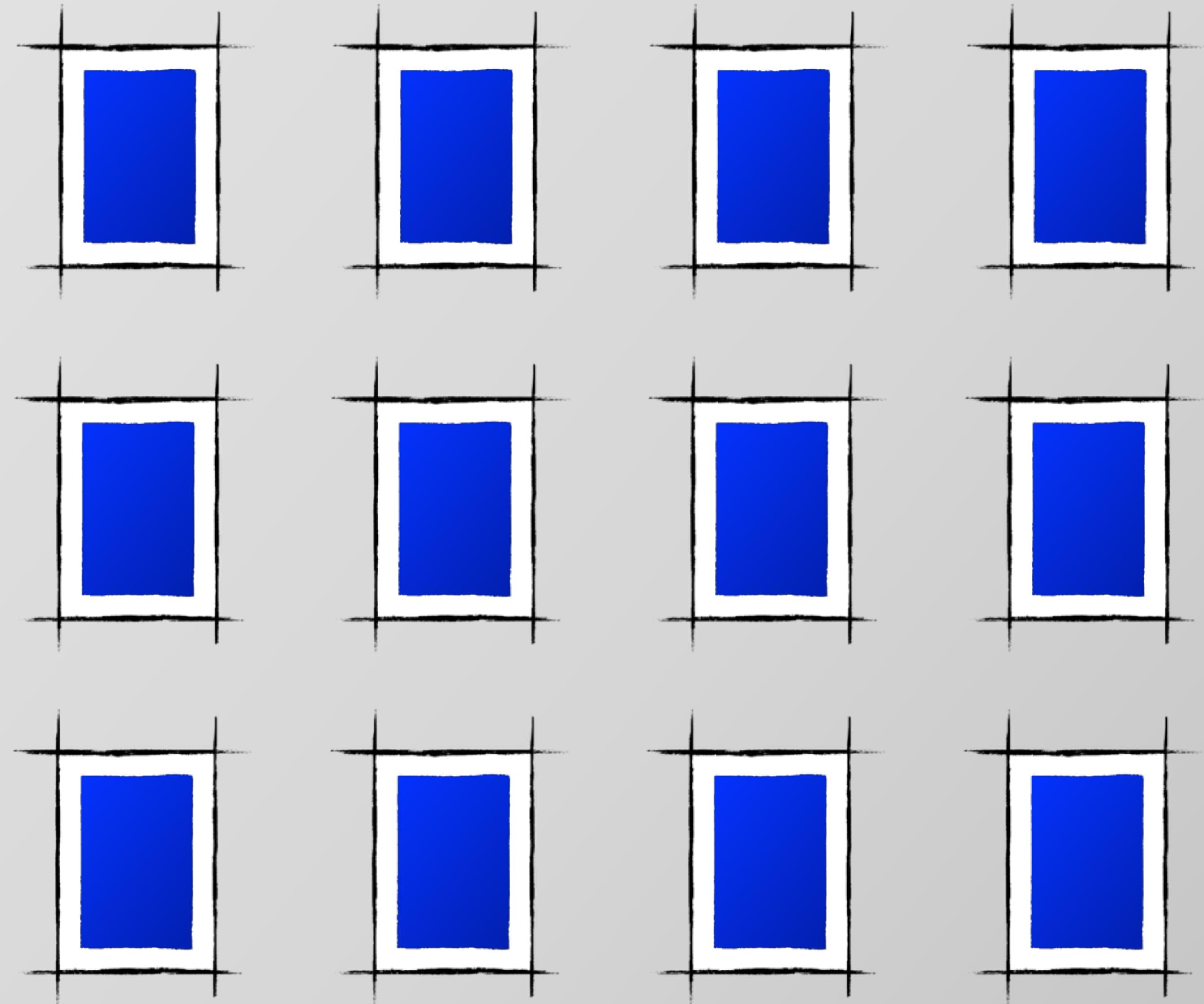
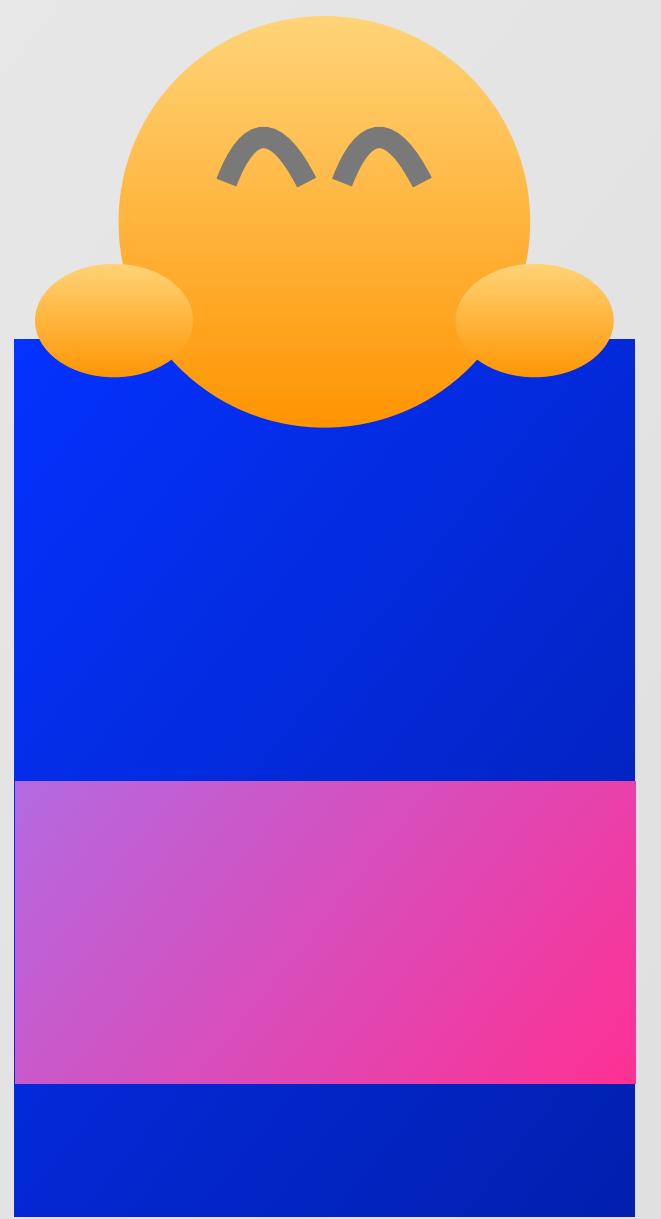
visualisation code also live, running on head node

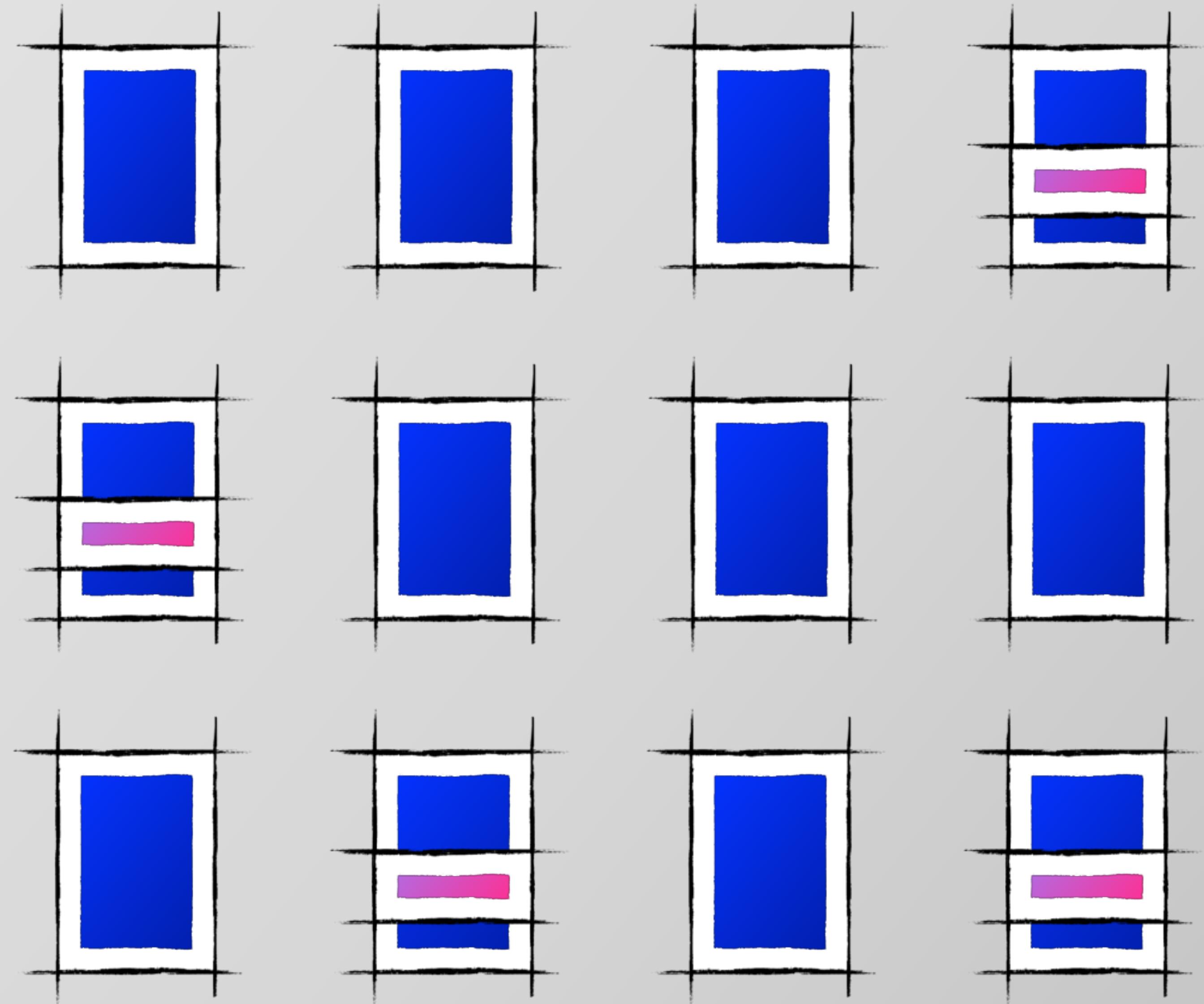
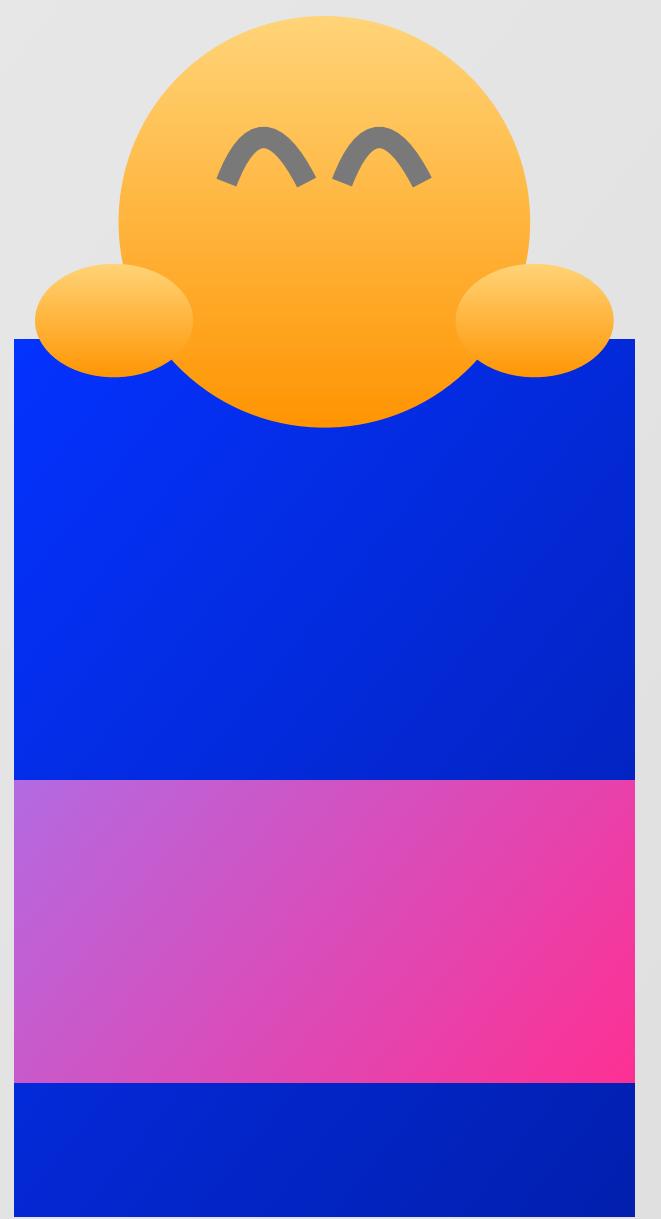


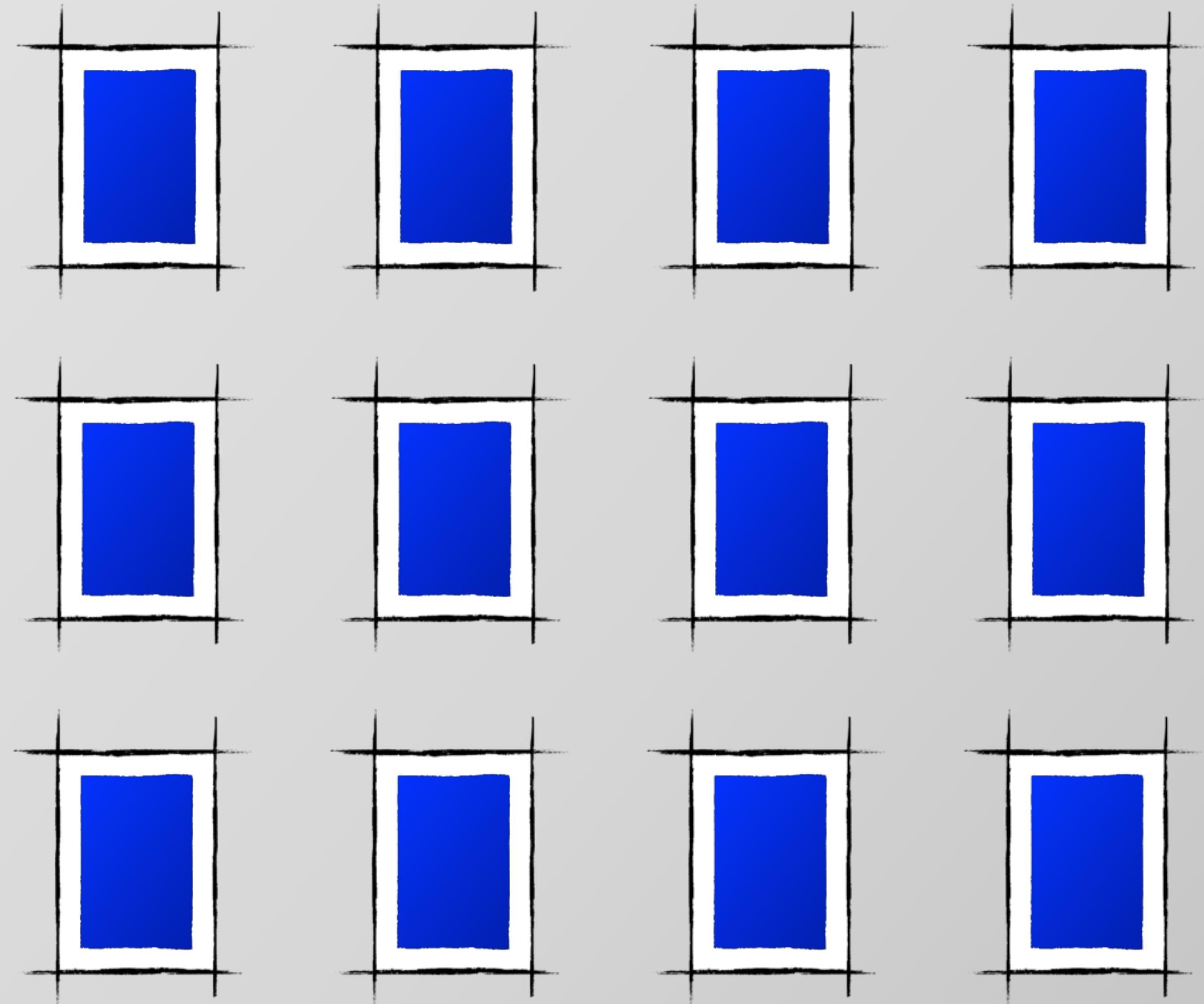
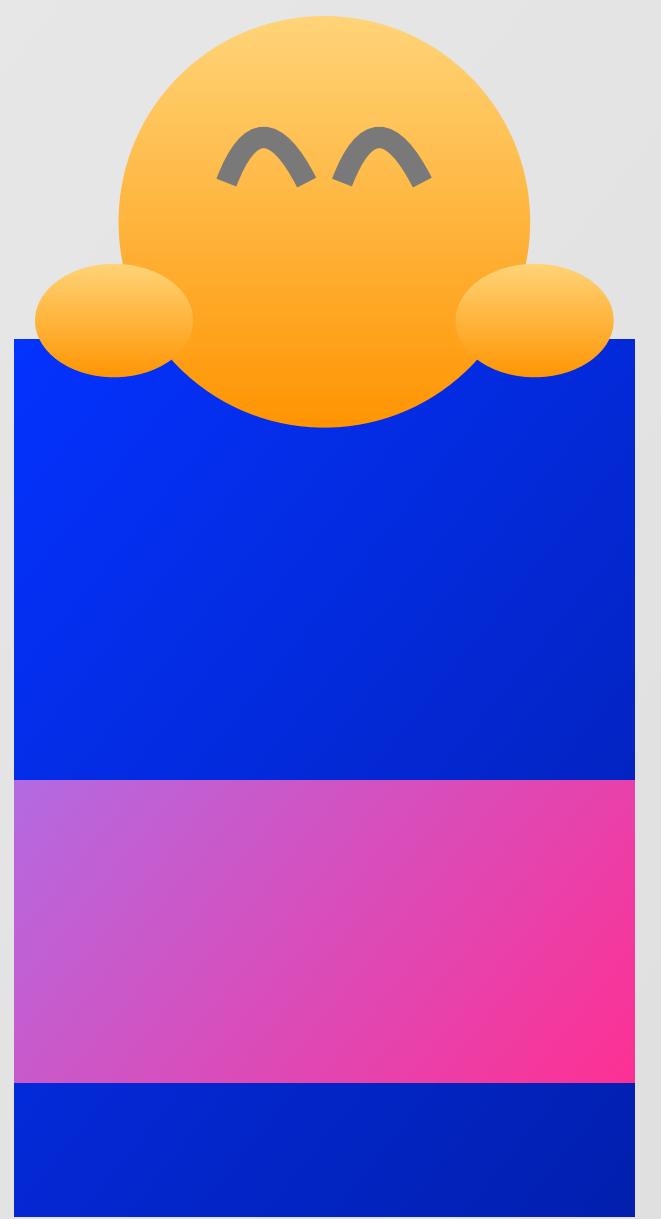


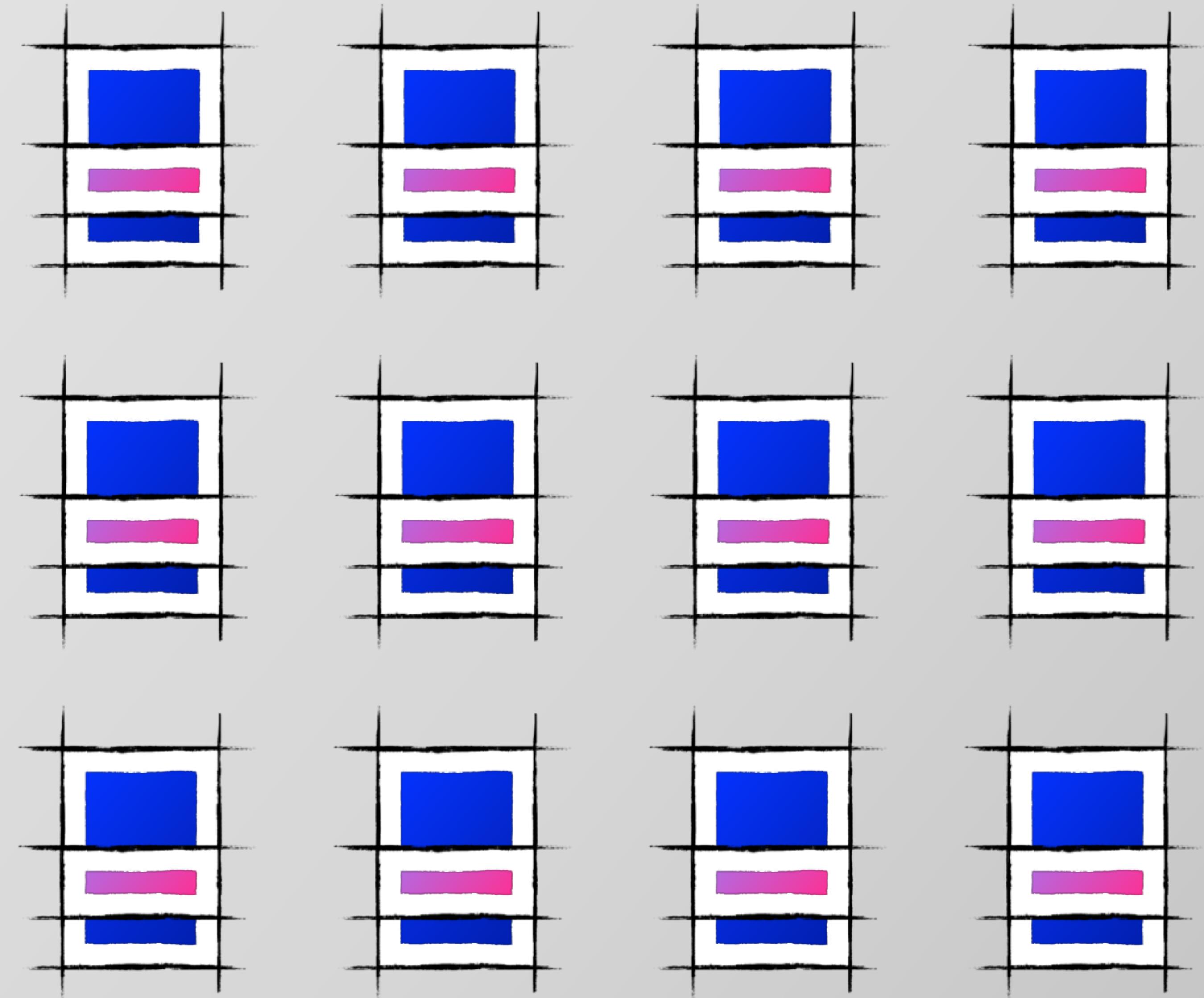
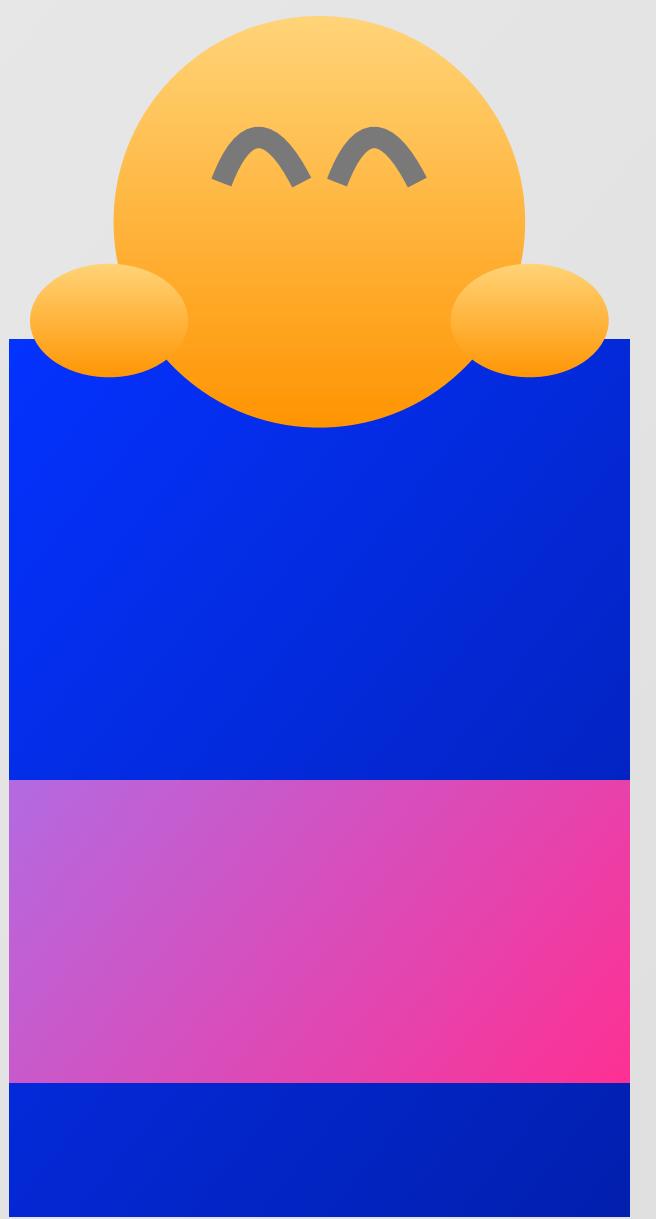


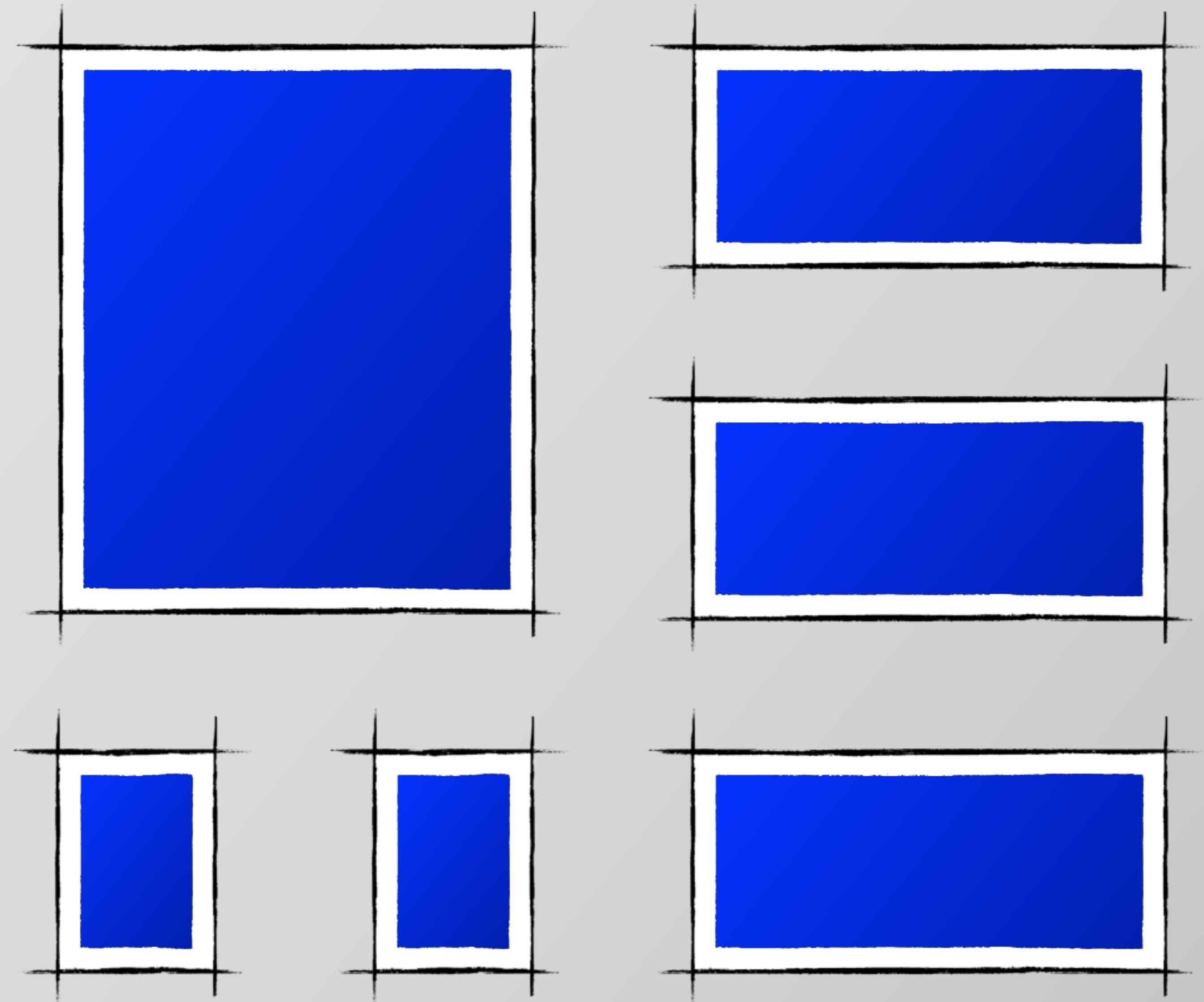
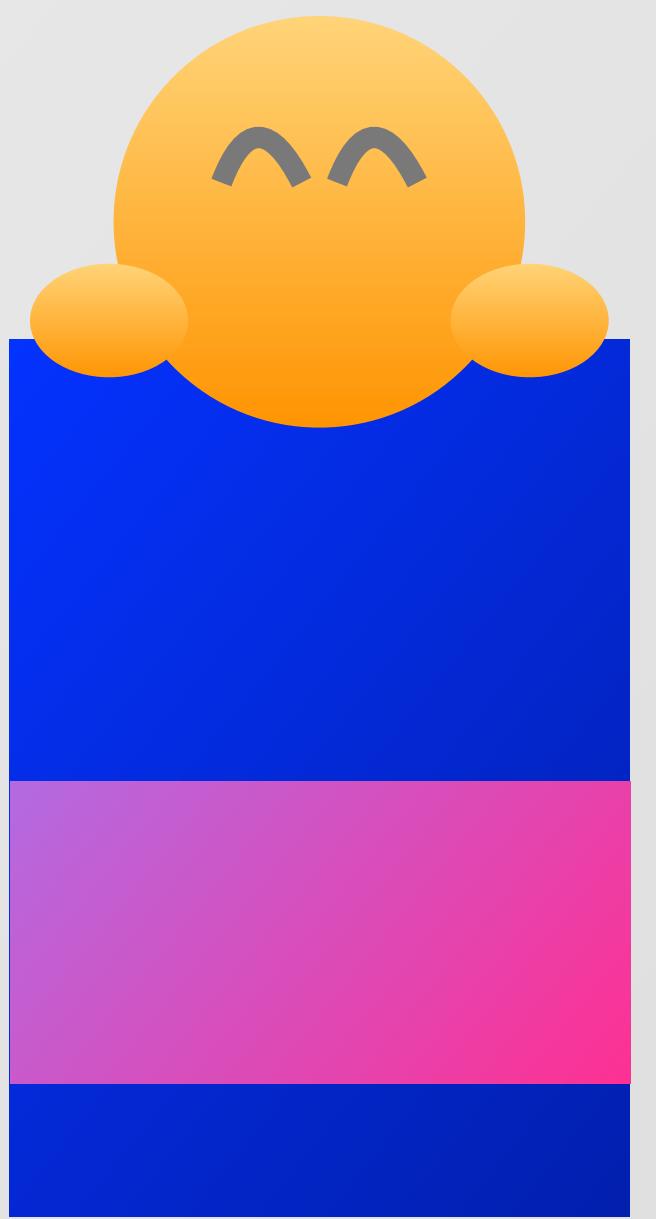


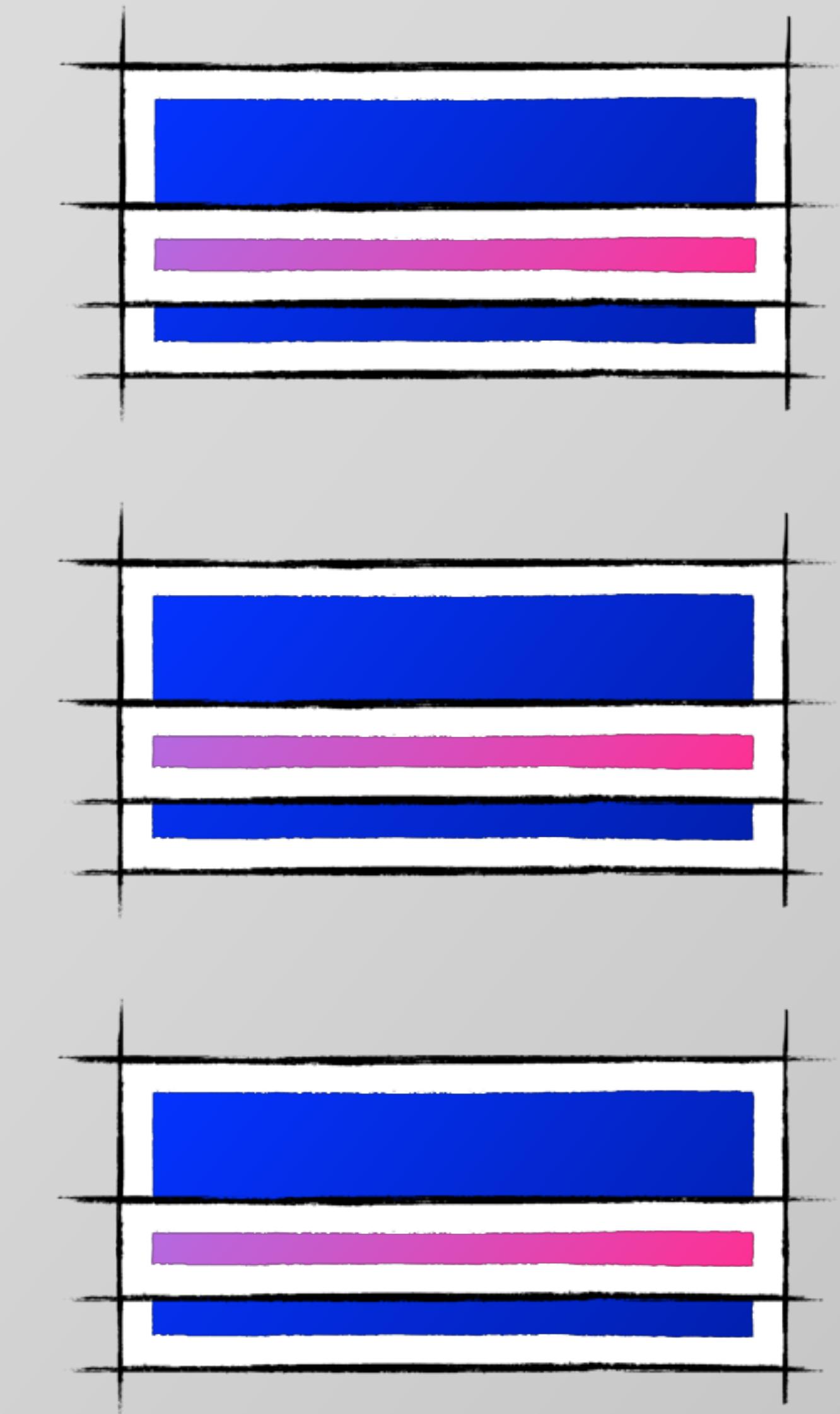
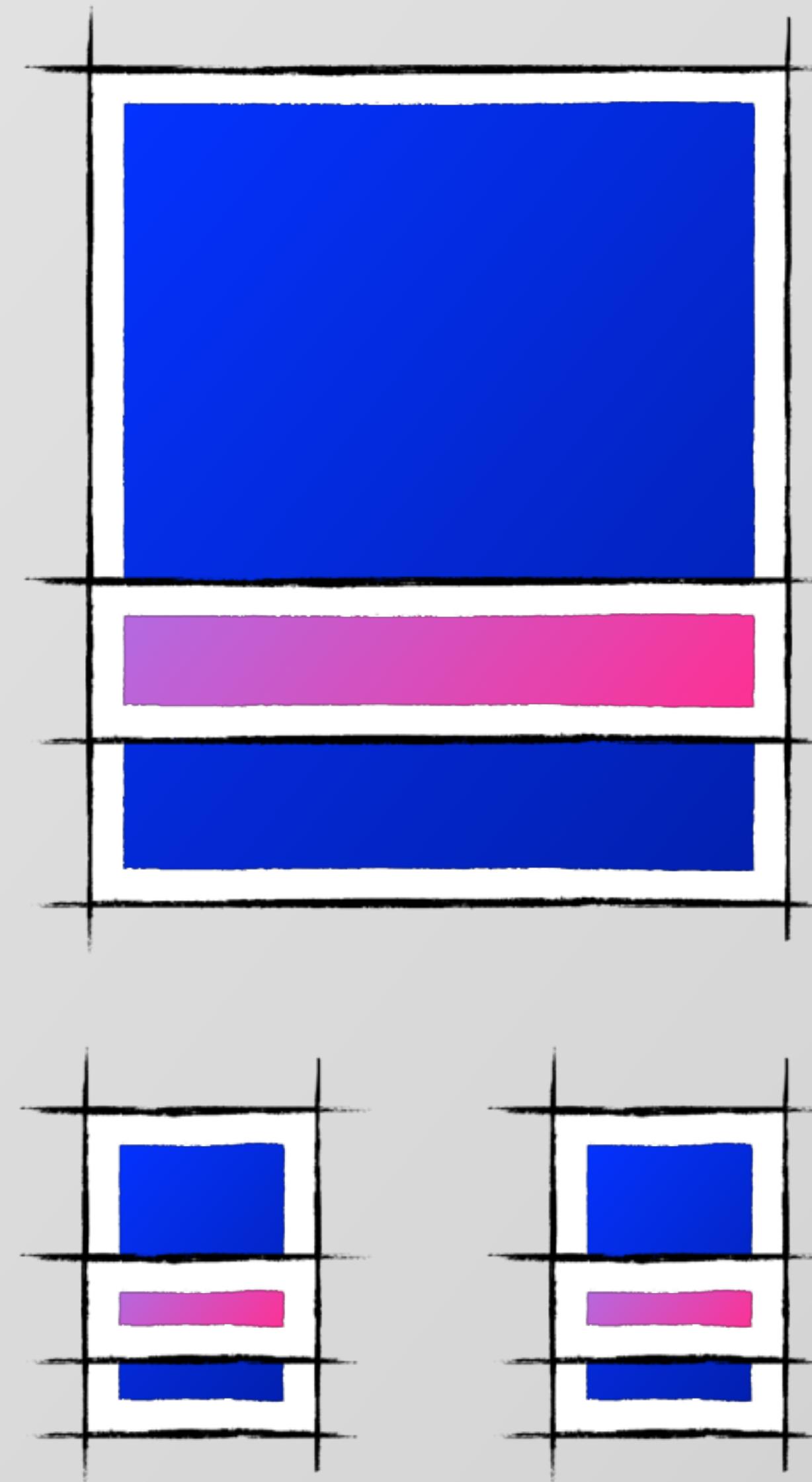
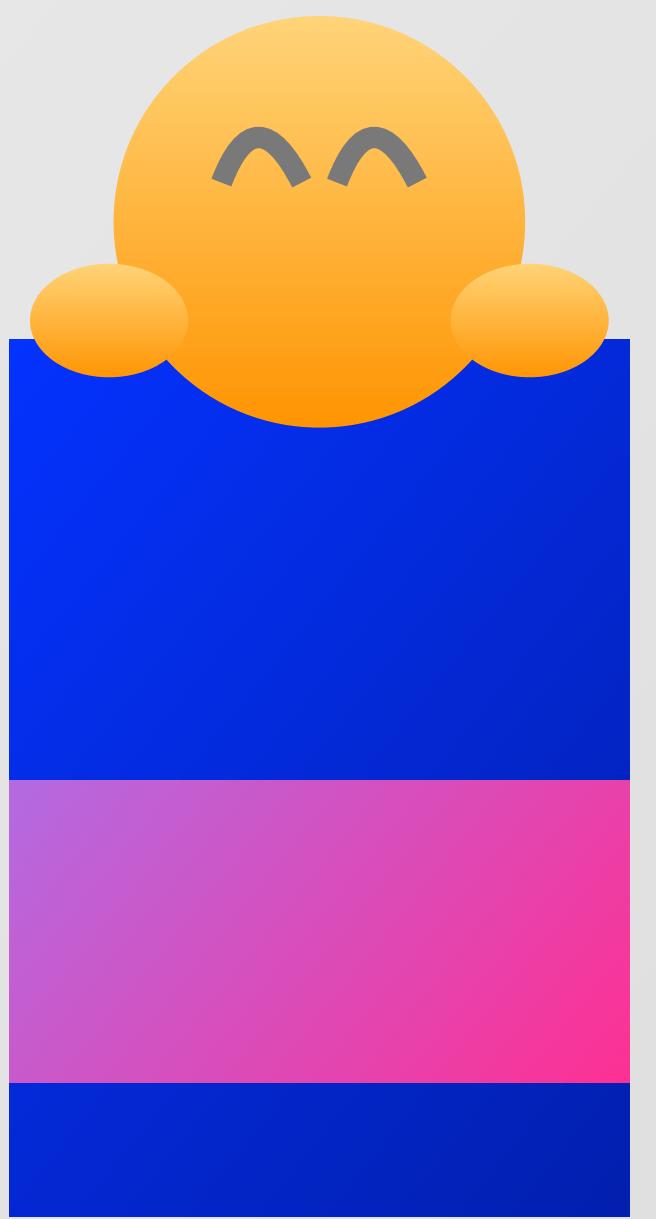












When might this be useful?

code **exploration**

in-situ & in-transit analysis

complex performance tuning, with live feedback

qualitative domain *understanding*

Open research challenges

safety (the Peter Parker principle)

reproducibility (and provenance)

balance of human & machine guidance



What we're doing at ANU

building Extempore & other live programming tools

applying live programming techniques in scientific simulation & high-performance computing

creating art + science research

teaching in live programming

modern HPC codes are
complex systems, with many
connections & dependencies

live feedback helps us explore
and understand these
relationships in our codes

For more information

ben.swift@anu.edu.au

<https://github.com/digego/extempore>

<http://benswift.me/extempore-docs>

#extempore on freenode

<http://cs.anu.edu.au>