

## How to debug a shared library (PGE)

- generally you should turn off optimization when debugging your code
  - use `-O0 -g`
- this will produce precise code to line number and variable access
  - the code will go slower, obviously, which might in extreme cases change the bug behaviour compared to `-O3`
  - you will need to experiment and become comfortable with these tools
- your experience will enable you to tradeoff these issues with your own bugs

## Adding Bungees into PGE

- so far springs, polygons and circle objects have been introduced into PGE
- recall that the spring has an at rest length  $l_0$  and the two objects are currently  $l_1$  distance apart
- a bungee is a modification of the spring object
  - it only pull objects together if  $l_1 > l_0$

## Bungee API

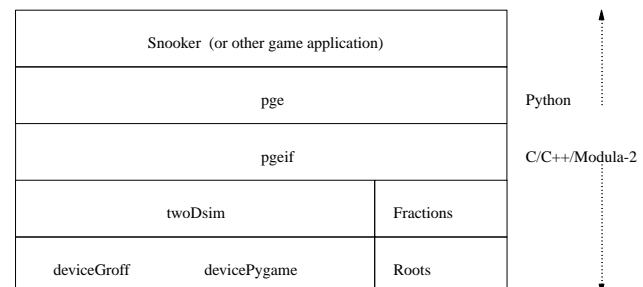
- in Python we could introduce the `bungee` method which is defined as:

`Sandpit/pge/python/pge.py`

```
#
# bungee - Pre-condition: a spring object. toBungee is
#           Post-condition: if toBungee is True convert
#                           to a bungee
#                           else convert the bungee back
#                           regular spring.
#
def bungee (self, toBungee):
    # finish this method
    # it should check self is a spring
```

## PGE Layers and associated files

- there are a number of layers in PGE



## Layers and source files to be altered

- `pge/python/pge.py`
  - the user level python API file
  - this is the only PGE visible file to the user
  
- `pge/i/pgeif.i`
  - the swig interface (python calling C/C++ definition)
  - remember to edit both sections (C/C++ section and the Python section)
  - hint look for `%{` and `%}` delimiters

## Layers and source files to be altered

- `pge/c/Gpgeif.h`
  - header file for `pgeif.c`
  - contains the external functions implemented inside `pgeif.c`
  
- `pge/c/pgeif.c`
  - its purpose is to allow, colours, polygons, circles, springs, to be given a unique integer
  - thereafter all references to objects will be achieved via the objects, id.
  - notice that inside `twoDsim.c` colours and circles are different

## Layers and source files to be altered

- `pge/c/twoDsim.c`
  - the actual game engine, which implements polygons, circles, springs
  
- `pge/c/GtwoDsim.h`
  - the header file for `pge/c/twoDsim.c` which defines all external functions

## Layers and source files to be altered

- all these files will need bungee references added to them
  
- start at the top `pge/python/pge.py` and work downwards
  
- follow per object gravity as a guide
  
- you will need to actually implement bungee's inside `twoDsim.c` (alter the behaviour of a spring)

## Layers and source files to be altered

- hint
  - add an extra field to spring `isBungee` and set it to `FALSE` by default in `twoDsim_spring`

## Hints on implementing bungees in twoDsim

- extend the spring object to contain the bungee field
- notice the functions `calcSpringFixed` and `calcSpringMoving` calculate the forces for the spring
- this can be adapted to check for the bungee properties
  - keep the code as clean as possible!

## Debugging a shared library

- the debugger `gdb` is very powerful, and it makes sense to use it when debugging the game engine
- often in `pge` a bug will be found to occur at a particular frame
  - for example there was a bug in objects falling
  - objects which were connected to a spring fell at a different speed to an object unconnected!

## Debugging the object falling bug!

- firstly the game was run and observed
  - secondly the game was inspected using `pgeplayback`
  - this confirms that at the initial frame (1) all is good, the objects are at the same position
  - in the second frame the objects have fallen to different `y` positions!

## Debugging the object falling bug!

- now we add a debug hook into

c/deviceIf.c:35

```
# include "GDynamicStrings.h"
# include "GNetworkOrder.h"
# include "GM2RTS.h"

# include "Ggdbif.h"

typedef unsigned int deviceIf_Colour;

# define whiteCID 1
```

## Debugging the object falling bug!

c/deviceIf.c:546

```
/*
   frameNote - emit a note to indicate a new frame has co
 */

void deviceIf_frameNote (void)
{
    checkOpened ();
    if (device == groff)
    {
        RawIO_Write (ffile, (unsigned char *) "fn", 2);
        NetworkOrder_writeCard (ffile, nextFrame);
    }
    if (nextFrame == 1)
        gdbif_sleepSpin ();
    nextFrame += 1;
}
```

## Debugging the object falling bug!

- now pge needs to be rebuilt and run again
- when running it will issue a console message

```
process 1234 is waiting for you to:
attach 1234
```

## Debugging the object falling bug!

```
$ cd
$ cd Sandpit/build-pge
$ gdb _pgeif.so
```

- you can now single step pge and print out variable values
- you might find it useful to use emacs which will give you a windowed interface to gdb

## Tutorial

- use these slides to add bungees into your version of pge
  
- write some simple test code in Python to create a bungee spring
  - ensure that it also has a fps counter on screen
  - write down the fps
  
- now see if you can rebuild pge using some of the optimisation techniques discussed in the slides
  - does the fps change?