

Implementing a callback in PGE

- in the PGE API there are callbacks defined
 - for example the `at_time` method in the `event` class

■ `$HOME/Sandpit/pge/python/pge.py`

```
def at_time (t, p):  
    global idcount, id2func, slow_down_factor  
    idcount += 1  
    pgeif.create_function_event (t / slow_down_factor, idcount, 0)  
    id2func[idcount] = p  
    return idcount
```

at_time

- informs the physics engine to create a call to function, p, at time, t, in the future
 - it returns a integer reference for the timed function.
- pre-condition: t is a time seconds (float) in the future.
 - p, is a function which takes two parameters the first parameter is the event and the second is an unused parameter
 - the second parameter is only there to allow coexistence with other call back functions
- post-condition: function p is placed into the timer list and a timer id (integer) is returned

at_time

- notice that the function `p` is recorded into the dictionary `id2func` and is referenced by an integer `idcount`
- the game engine shared library (`twoDsim.c`) will, at the appropriate time, issue a request to the python module to execute a function number
 - it does this by issuing a `function_event` with a function id which is the integer used to lookup the python function held in the dictionary
 - see the method `_process` in class `event` in `pge.py`

Implementing `impulse_exceeds`

- you can use the ability of `twoDsim.c` to request python calls a function to implement your `impulse_exceeds` callback
- hint you might want introduce changes to the API in class object

Implementing impulse_exceeds

`$HOME/Sandpit/pge/python/pge.py`

```
#
# impulse_exceeds - on collision if the impulse generated by this object
#                   (and the other object in collision)
#                   exceeds, value, call, function.
#                   function is a python function which takes a single
#                   parameter (param). Param is the other object in
#                   collision.
#
def impulse_exceeds (self, value, function):
    ...
```

Implementing `impulse_exceeds`

- you could propagate this call through the various layers down to `twoDsim.c` and implement this test within the game engine

Implementing impulse_exceeds

- hint you might want to add new fields to the object structure

- ```
unsigned isImpulseActive; /* a boolean indicating the exceed test is active. */
long double impulseThreshold; /* if the impulse exceeds the threshold call function.
unsigned int impulseCallback; /* the integer function number. */
```

## Implementing `impulse_exceeds`

- hint search for `hasCallbackLength` in `twoDsim.c`
  - the spring object can be set to snap if the spring exceeds a value
  - it will call a function (should the spring exceed a length)
- very similar to exceeding an impulse
  - you can use similar code to implement the impulse callback