

How to start implementing frozen bubble in PGE

- start by examining the bagatelle example
`examples/bagatelle/bagatelle.py`
- remove triangles, silos and pegs

How to solve the bubble freezing and unfreezing?

- these are code hints, they will need polishing and completing
 - this code has been used to implement the frozen bubble game

How to solve the bubble freezing and unfreezing?

```
#  
# bubble class which is used to record, colour, circle and children  
# children are bubbles which connect below the current bubble  
#  
class bubble:  
    def __init__ (self, cir, col):  
        self.circle = cir  
        self.colour = col  
        self.circle.fix ()  
        self.circle.on_collision (bubble_hits_bubble)  
        self.children = [] # no bubbles attached below this bubble
```

How to solve the bubble freezing and unfreezing?

```
#  
# addChild - adds bubble, b, as a child of self.  
#  
def addChild (self, b):  
    self.children += [b]  
#  
# removeChild - removes all bubbles in blist from the children field.  
#  
def removeChild (self, blist):  
    if self.children != []:  
        self.children = [x for x in self.children if x not in blist]
```

How to solve the bubble freezing and unfreezing?

- ```
bubbleDict = {} # allows us to obtain a bubble list from a pge circle
currentCircle = None
currentColour = None
```

## How to solve the bubble freezing and unfreezing?

```

removeChildren - remove all bubbles in blist from all children in all bubbles.

def removeChildren (blist):
 for circle in bubbleDict.keys ():
 for b in bubbleDict[circle]:
 b.removeChild (blist)
```

## How to solve the bubble freezing and unfreezing?

```
#
unfreezeList - unfreeze all bubbles in blist and return any children bubbles
of these bubbles. Remove any reference to these circles/bubbles
as they will be deleted as soon as they hit the bottom edge.
#
def unfreezeList (blist):
 todo = []
 for b in blist:
 print "bubble", b, "should be unfixed"
 b.circle.unfix ()
 if bubbleDict.has_key (b.circle):
 todo += bubbleDict[b.circle]
 del bubbleDict[b.circle]
 todo += b.children
 removeChildren (todo)
 return todo
```

## How to solve the bubble freezing and unfreezing?

```

unfreezeAll - unfreeze all bubbles attached to circle (via children)

def unfreezeAll (circle):
 global bubbleDict
 blist = bubbleDict[circle]
 while blist != []:
 blist = unfreezeList (blist)
 currentCircle.unfix ()
```



## How to solve the bubble freezing and unfreezing?

```

bubble_hits_bar - the callback for the bubble hitting the top edge.

def bubble_hits_bar (o, e):
 global bubbleDict, currentCircle

 if currentCircle != None:
 if bubbleDict.has_key (currentCircle):
 print "odd the bubble is already registered"
 else:
 initChain (currentCircle, currentColour)
 currentCircle = None
```

## How to solve the bubble freezing and unfreezing?

```

initChain - create an initial bubble in a chain of one.

def initChain (circle, colour):
 global bubbleDict
 b = bubble (circle, colour)
 bubbleDict[circle] = [b] # the first bubble in a chain is keyed via a circle
```

## How to solve the bubble freezing and unfreezing?

```

updateChains - adds currentCircle/currentColour bubble to the chain specified by circle

def updateChains (circle):
 global bubbleDict
 print "updateChains"
 blist = bubbleDict[circle]
 print "blist", blist, "len (blist) =", len (blist)
 blist += [bubble (currentCircle, currentColour)]
 print "blistis now", blist, "len (blist) =", len (blist)
 # and update all circle -> chain entries
 for b in blist:
 bubbleDict[b.circle] = blist
```

## addBubble

- is only called when adding a bubble to an existing bubble chain.
- the bubble keyed by circle will be known and `currentCircle/currentColour` is the new bubble which is added onto the chain.
- you will need to implement this!

## addBubble

- Prerequisite
  - circle is a pge object which has not been deleted.
  
- Postrequisite
  - assuming that circle is already known in the bubbleDict and if the collided chain has two bubbles of the same colour then unfreeze these bubble and their children. A same colour collision is appended to the chain. A different colour collision creates a new chain with a single bubble.

## **bubble\_hits\_bubble**

```

bubble_hits_bubble - call back for a circle hitting a frozen bubble.

def bubble_hits_bubble (o, e):
 global currentCircle
 print "bubble hits bubble, currentCircle =", currentCircle
 if currentCircle != None:
 b = e.collision_between ()
 for o in b:
 if not currentCircle.is_fixed ():
 addBubble (o)
 currentCircle = None
```

## **unfix and is\_fixed**

- it would be useful to implement `unfix` and `is_fixed` in `pge`
- in order to achieve this we need to modify the following files
  - `pge/i/pgeif.i`
  - `pge/c/Gpgeif.h`
  - `pge/c/pgeif.c`
  - `pge/c/GtwoDsim.h`
  - `pge/c/twoDsim.c`

## 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 `pge/c/twoDsim.c` colours and circles are different
  
- `pge/c/GtwoDsim.h` contains the prototypes and external declarations for `pge/c/twoDsim.c`

## unfix hints

- in the five files mentioned on the previous slides
  - search for the `fix` function
  - duplicate it, and change the duplicate to `unfix`
  - adjust the implementation of the function (if necessary)
  - finally correct the comments!
  
- note that the file `pge/python/pge.py` will need a few edits to change the runtime checking
  - it currently checks to ensure that a `fixed` object has no mass
  - and a `fixed` object can not be given an acceleration/velocity
  - obviously this needs to be changed
  - in effect these checks need to be removed

## pge/python/pge.py

- removing the checking code
  - the checking code is found in many methods and will look similar to

```
def velocity (self, vx, vy):
 ...
 self._check_not_fixed ("assign a velocity")
 ...
```

## Running your pge code

```
$ cd Sandpit/build-pge
$./localrun.sh ../pge/frozenbubble/frozenbubble.py
```

- example of frozen bubble implemented with the above code in pge [frozenbubble.mp4](http://floppsie.comp.glam.ac.uk/pge/frozenbubble.mp4) (<http://floppsie.comp.glam.ac.uk/pge/frozenbubble.mp4>)