Dynamic Data Structures and Algorithms.

ENGF0002: Design and Professional Skills

Prof. Mark Handley, University College London, UK

Term 1, 2018


Stoke Newington 73
© EPA

Today's topics

- Understand how dynamic data types are implemented.
- Explore in detail how to implement linked lists.
- Introduce Object Oriented Programming
- Implement a data type that maintains a sorted set of data using a tree.

A Node class for a linked list.

```
1   class Node():
2       def __init__(self, value):
3           self.value = value
4           self.next = None
```

A class defines a new data type.

- It can contain data.
- It can contain functions that act on that data

A Node class for a linked list.

```
1   class Node():
2       def __init__(self, value):
3           self.value = value
4           self.next = None
```

The `__init__()` function is special. It is called when a new instance of the class is created.

```
68      n1 = Node(1)
```

Helper functions make the API more obvious

```
1   class Node():
2       def __init__(self, value):
3           self.value = value
4           self.next = None
5
6       def get_next(self):
7           return self.next
8
9       def get_value(self):
10          return self.value
```

A test for node initialization:

```
68      n1 = Node(1)
69      assert(n1.get_value() == 1)
70      assert(n1.get_next() == None)
```

Adding to a linked list is fast

```
1   class Node():
2       def __init__(self, value):
3           self.value = value
4           self.next = None
```

Appending to an existing node just requires updating the `next` node field.

```
12      def append(self, node):
13          if self.next != None:
14              raise ValueError("Append to non-empty node")
15          self.next = node
```

Adding anywhere in a linked list is fast

```
1   class Node():
2       def __init__(self, value):
3           self.value = value
4           self.next = None
```

```
30      def insert_after(self, value):
31          #insert a value after the current node
32          node = Node(value)
33          node.next = self.next
34          self.next = node
```

```
116     n1 = Node(1)
117     n2 = Node(2)
118     n1.append(n2)
119     n3 = Node(3)
120     n2.append(n3)
121     n1.insert_after(42)
122     assert(n1.list() == [1,42,2,3])
```

Deletion anywhere in a linked list is fast

```
1   class Node():
2       def __init__(self, value):
3           self.value = value
4           self.next = None
```

```
26      def delete_next(self):
27          if self.next != None:
28              self.next = self.next.get_next()
```

```
122     assert(n1.list() == [1,42,2,3])
123     n1.delete_next()
124     assert(n1.list() == [1,2,3])
```

Indexing into a linked list requires O(n) time

```
1   class Node():
2       def __init__(self, value):
3           self.value = value
4           self.next = None
```

```
51      def find_by_index(self, index):
52          if index == 0:
53              return self.value
54          if self.next == None:
55              raise IndexError("Attempt to index past end of list")
56          return self.next.find_by_index(index - 1)
```

```
124     assert(n1.list() == [1,2,3])
125     assert(n1.find_by_index(0) == 1)
126     assert(n1.find_by_index(1) == 2)
127     assert(n1.find_by_index(2) == 3)
```

Visualization Diversion

Bomber Programme

Bug 1: Bomb doesn't come back when it misses buildings

Cause of bug: missing test for bomb hitting ground.

```
235         def check_bomb(self):
236             if not self.bomb.falling:
237                 return
238             # did the bomb hit a building?
239             for building in self.buildings:
240                 if building.is_inside(self.bomb.position):
241                     self.bomb.explode()
242                     building.shrink()
```

Missing tests are common causes of bugs in code. This one is pretty obvious, but they're often not obvious, and only hit very rarely. Big source of obscure runtime errors.

Bug 2: Can't bomb right hand building

Cause of bug: plane doesn't wrap far enough right. Wrong constant used to move the plane.

```
168         ''' move the plane however much it moves during one frame '''
169         def move(self):
170             self.position.move(-4 * speed, 0)
171             if self.position.getX() < -self.width:
172                 self.position.move(CANVAS_WIDTH, 40)
```

Fix: move the plane by `CANVAS_WIDTH + self.width`.

Bug 3: Too many buildings

Symptom: If you fix Bug 2, the plane hits a building off the right side of the screen. **Cause**: magic embedded constant is wrong

```
219         def create_buildings(self):
220             #remove any old buildings
221             while len(self.buildings) > 0:
222                 building = self.buildings.pop()
223                 building.cleanup()
224 
225             #create the new ones
226             for building_num in range(0, 1200//SPACING):
227                 height = self.rand.randint(10,500) #random number between
228                 self.buildings.append(Building(self.canvas, building_num, 
229                                                self.building_width))
```

Fix: replace `1200` with `CANVAS_WIDTH`

Lesson: don't embed random magic numbers - try to use named constants

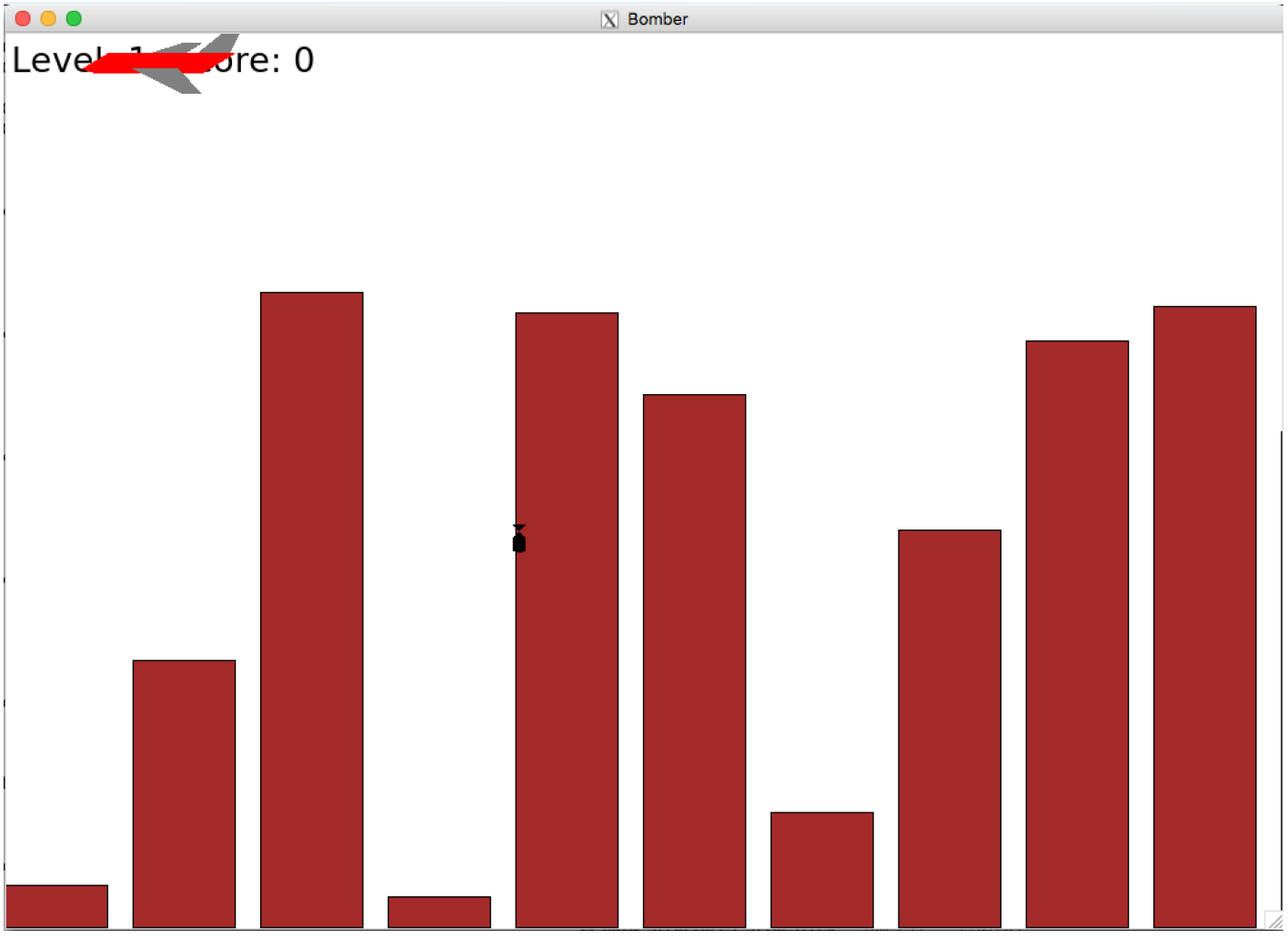
Bug 4: Plane can't land

Cause: test preventing going off the bottom of screen is wrong due to bad assumptions about what the plane position represents.

```
172                 self.position.move(CANVAS_WIDTH, 40)
173                 #ensure we don't go off the bottom of the screen
174                 if self.position.getY() > CANVAS_HEIGHT:
175                     self.position.Y = CANVAS_HEIGHT
```

Fix: `Plane.position.getY()` returns the top of the plane. Test should test the bottom of the plane.

Bug 5: Bomb scrapes the left side of a building without exploding



Bug 5: Bomb scrapes the left side of a building without exploding

Cause: test only tests whether top left corner of bomb is inside a building.

```
238             # did the bomb hit a building?
239             for building in self.buildings:
240                 if building.is_inside(self.bomb.position):
241                     self.bomb.explode()
242                     building.shrink()
```

Fix: also test if the top right corner of the bomb is in a building