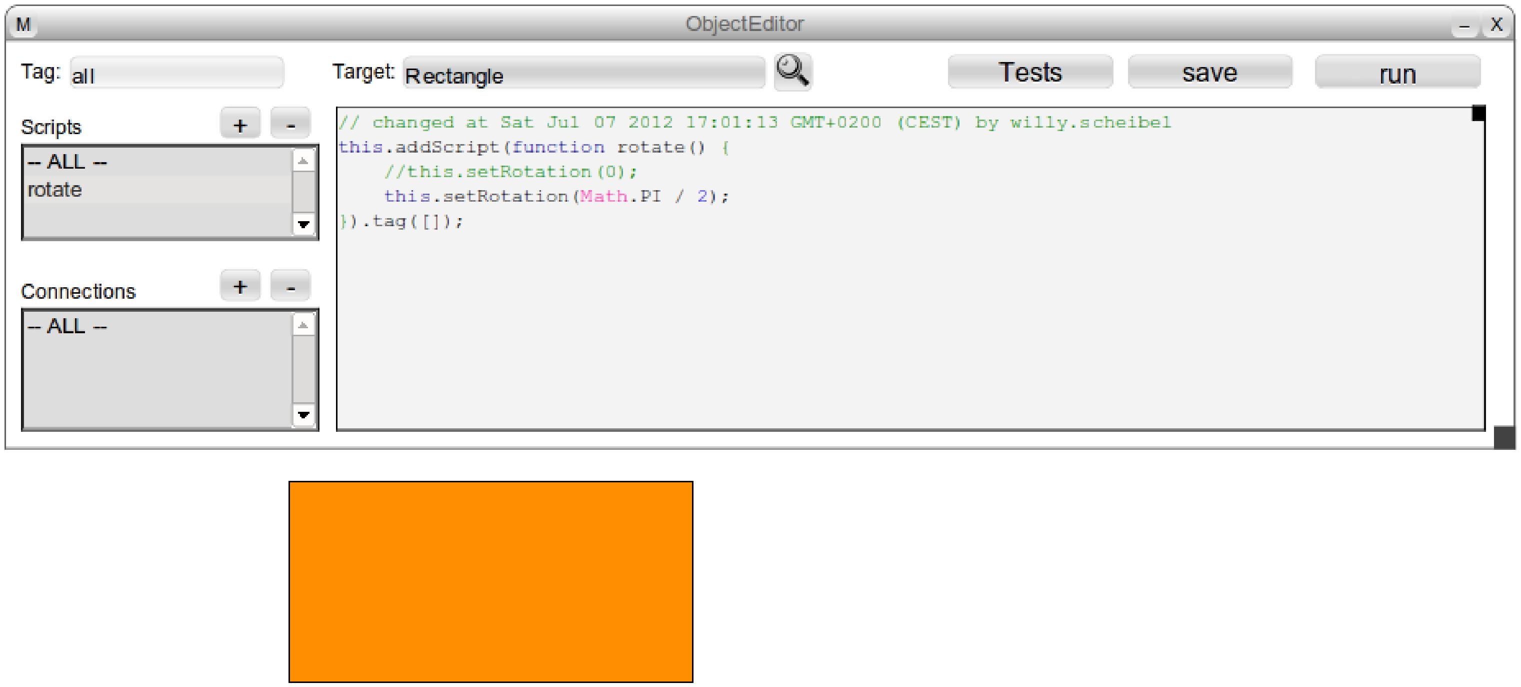
Web Based Software Development
Lively: Private Classes

Hasso-Plattner-Institut Potsdam
Software Architecture Group
Prof. Dr. Robert Hirschfeld
Roland Lux, Willy Scheibel
http://www.hpi.uni-potsdam.de/swa/
08.05.2012

Application Development in Lively

- Create “Parts” by using compound Morphs
  - Add specific behavior with added Scripts and Connections

- Parts may use Core-Classes to implement behavior

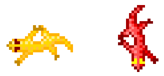


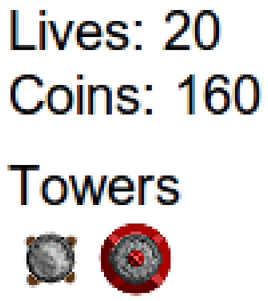
```
// changed at Sat Jul 07 2012 17:01:13 GMT+0200 (CEST) by willy.scheibel
this.addScript(function rotate() {
    //this.setRotation(0);
    this.setRotation(Math.PI / 2);
}).tag([]);
```

Original TowerDefense

- Started as a Part with Scripts
- Quickly evolved to a size which could not be handled with Scripts anymore
  - Introduction of real classes
  - Moving logic from Parts to the Class-system
- Further development of classes and only providing a window without Scripts as the Part

TowerDefense as a Part

- Uses different graphical entities
  - Tiles
  - Creeps 
  - Lifebars
  - Towers
  - Missiles
  - GUI



Lives: 20
Coins: 160
Towers

- Uses abstract entities without graphical representation
  - Levels
  - Paths
  - Directions (and additional subclasses)
  - Animations

Problems with Abstract Behavior in Parts

- Using own abstract behavior in Parts
  - By the use of strange methods (e.g. invisible Morphs)
  - By the use of the class system

- Problems with invisible Morphs
  - Counterintuitive
  - Missing metaphor

- Problems with existing class system
  - Code is versioned differently than Parts
  - A specific code version is not associated with a specific Part version

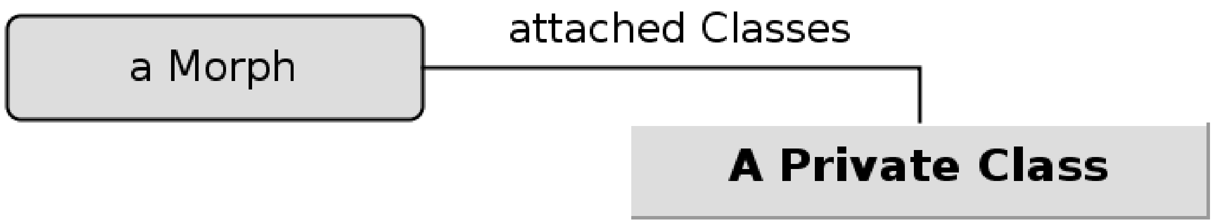
Multiple Versions of a Part

- Multiple versions of a Part can coexist within an image
    - If classes are versioned with their parts, these classes must also coexist in multiple versions within an image
    - But classes are identified by their global name

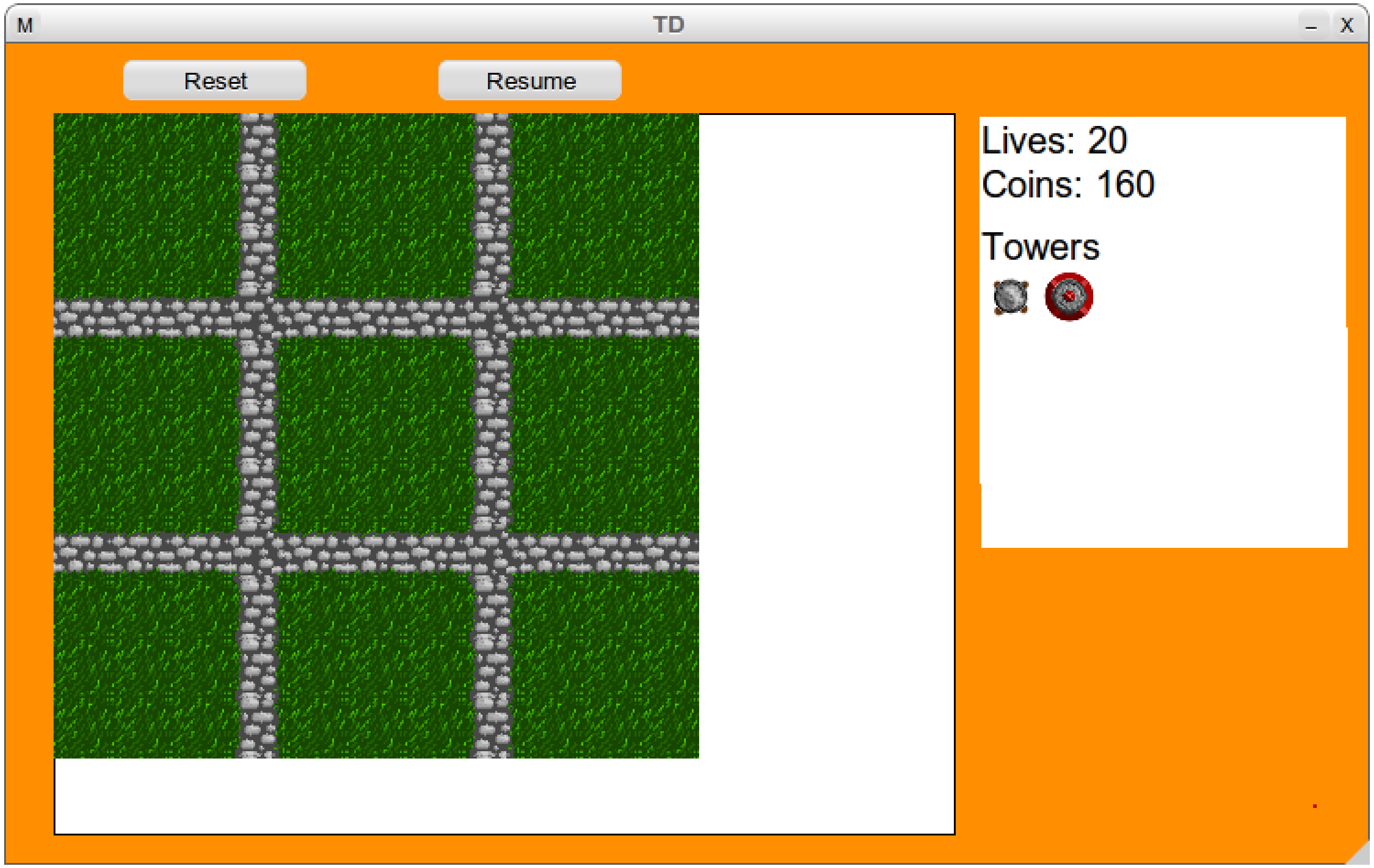
- Therefore, classes which are versioned with Parts must not have a global name

Our Solution: Private Classes

- Attached to Morphs (but can be attached to simple objects too)
- Attached private classes are saved and loaded with their Morphs
  - In PartsBin
  - In World

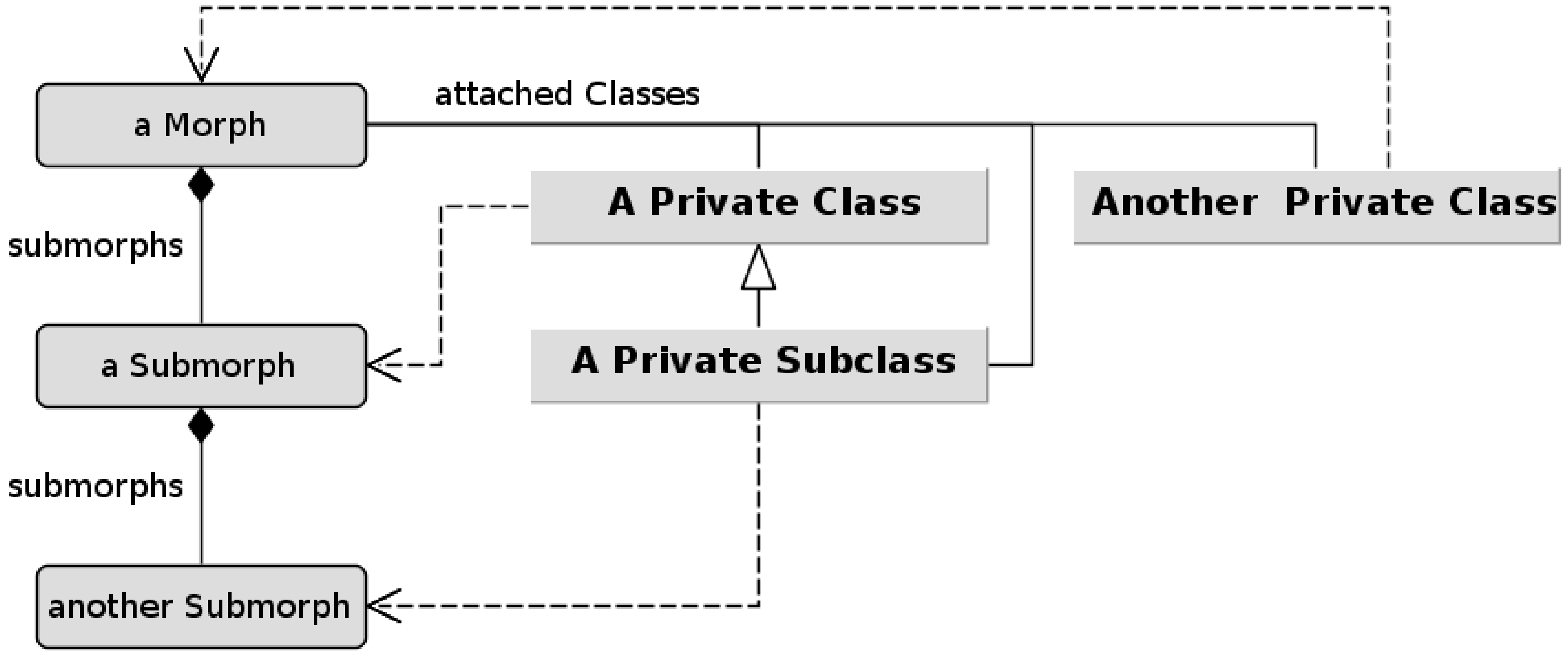


Demo



Lives: 20
Coins: 160

Features of Private Classes



Creating Private Classes

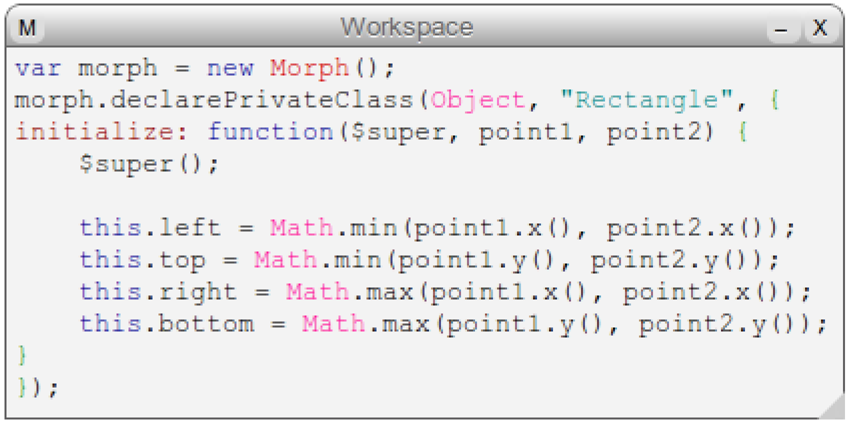
- With DoIts
  - But why would somebody want to do that?

- Simple class browser

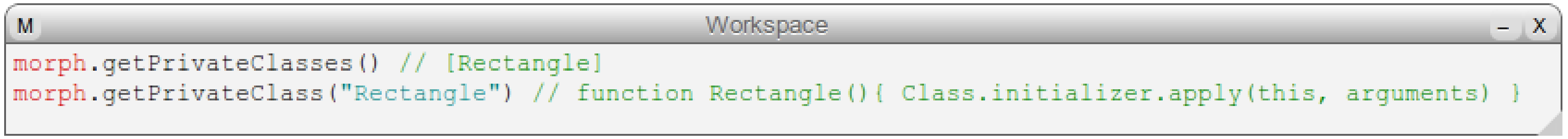
- Object editor integration

Creating Private Classes with DoIts

- Function.prototype
  - privateSubclass()
- Morph.prototype
  - getPrivateClass()
  - setPrivateClass()
  - getPrivateClasses()
  - openClassesInBrowser()



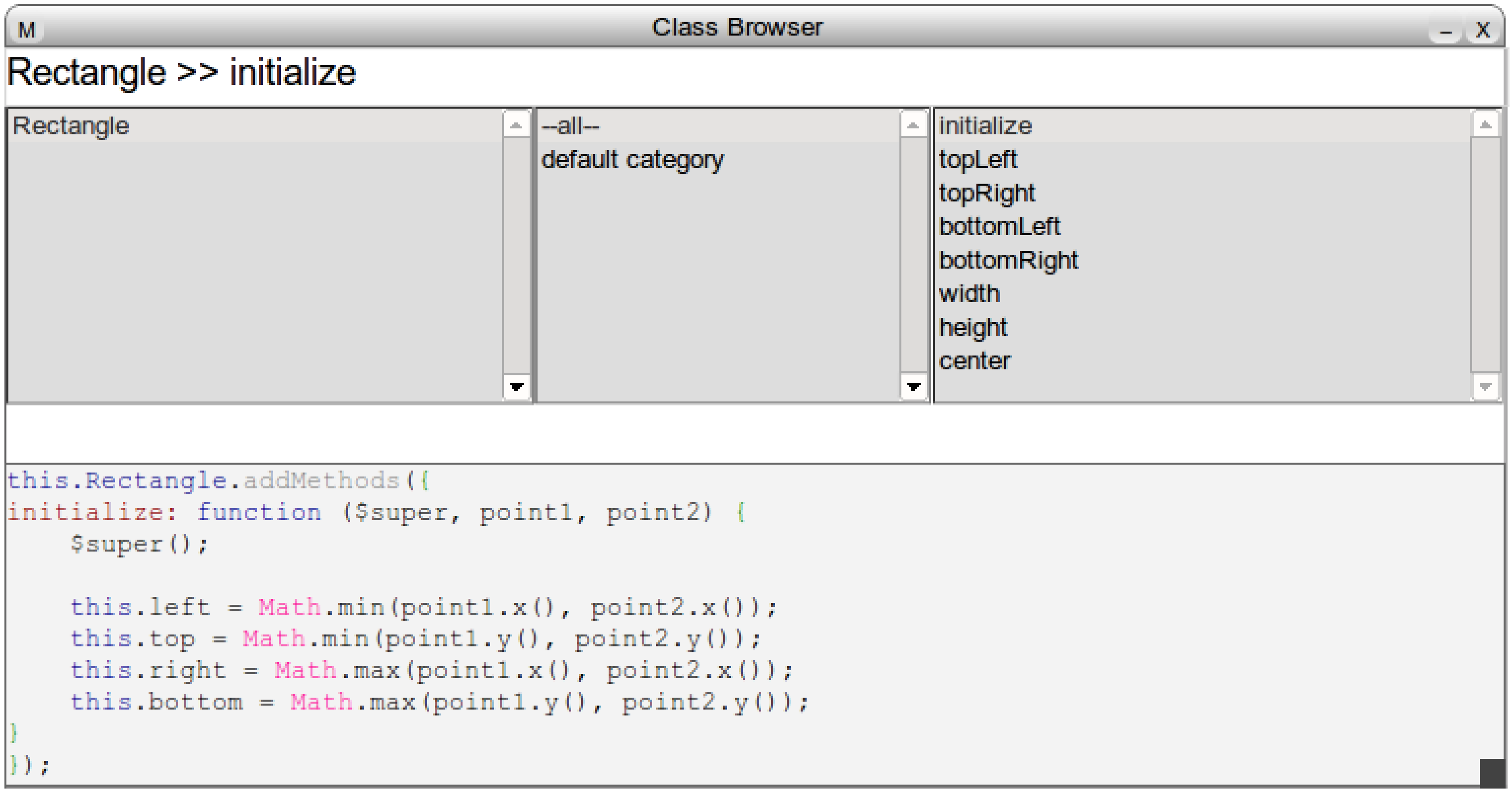
```
var morph = new Morph();
morph.declarePrivateClass(Object, "Rectangle", {
initialize: function($super, point1, point2) {
    $super();

    this.left = Math.min(point1.x(), point2.x());
    this.top = Math.min(point1.y(), point2.y());
    this.right = Math.max(point1.x(), point2.x());
    this.bottom = Math.max(point1.y(), point2.y());
}
});
```



```
morph.getPrivateClasses() // [Rectangle]
morph.getPrivateClass("Rectangle") // function Rectangle(){ Class.initializer.apply(this, arguments) }
```

Creating Private Classes in a Class Browser

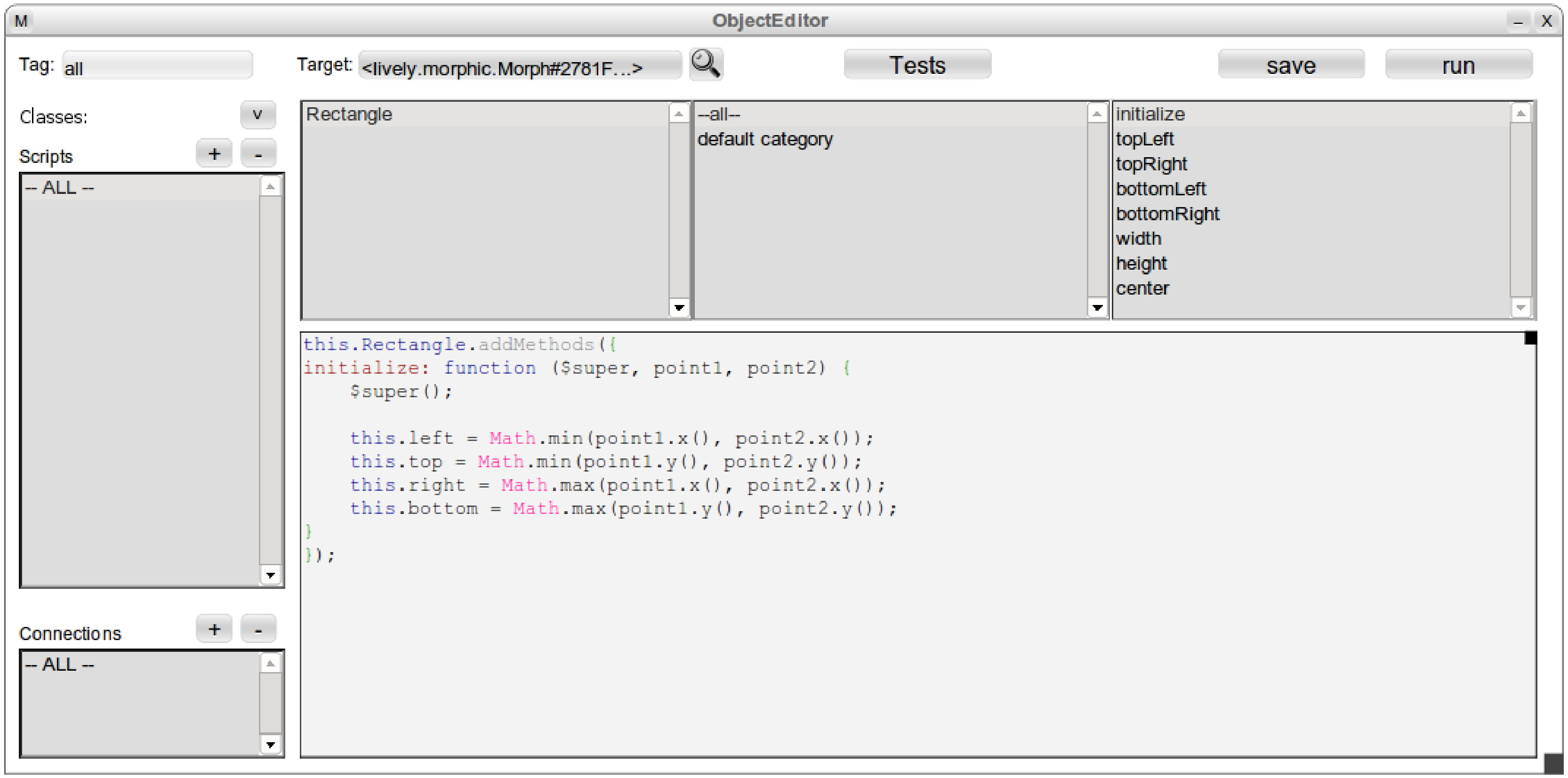
- Adding and modifying classes and methods
- Only instance side supported so far



```
this.Rectangle.addMethods({
initialize: function ($super, point1, point2) {
    $super();

    this.left = Math.min(point1.x(), point2.x());
    this.top = Math.min(point1.y(), point2.y());
    this.right = Math.max(point1.x(), point2.x());
    this.bottom = Math.max(point1.y(), point2.y());
}
});
```

Creating Private Classes in Object Editor

- Collapsible bar for classes, categories and methods
- Goal: provide one tool for application development in Lively instead of many



```
this.Rectangle.addMethods({
initialize: function ($super, point1, point2) {
    $super();

    this.left = Math.min(point1.x(), point2.x());
    this.top = Math.min(point1.y(), point2.y());
    this.right = Math.max(point1.x(), point2.x());
    this.bottom = Math.max(point1.y(), point2.y());
}
});
```

Using Private Classes

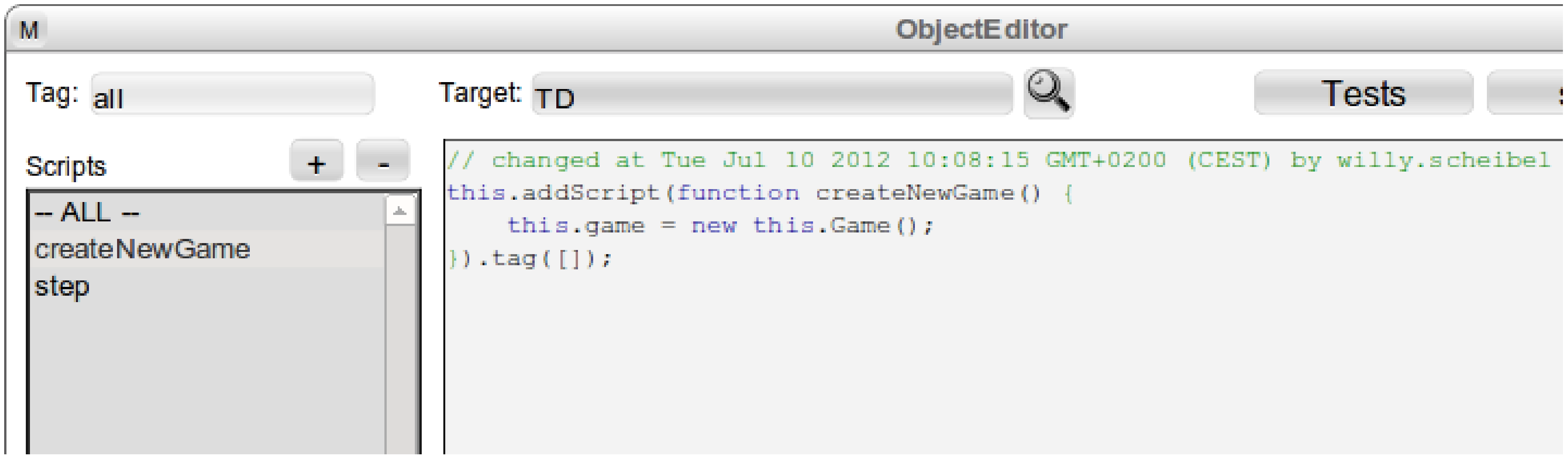
- Using private classes within a Morph it is attached to

- Using private classes within a private class of the same Morph

- Using private classes across Morphs, but within a Part

Using Private Classes within a Morph

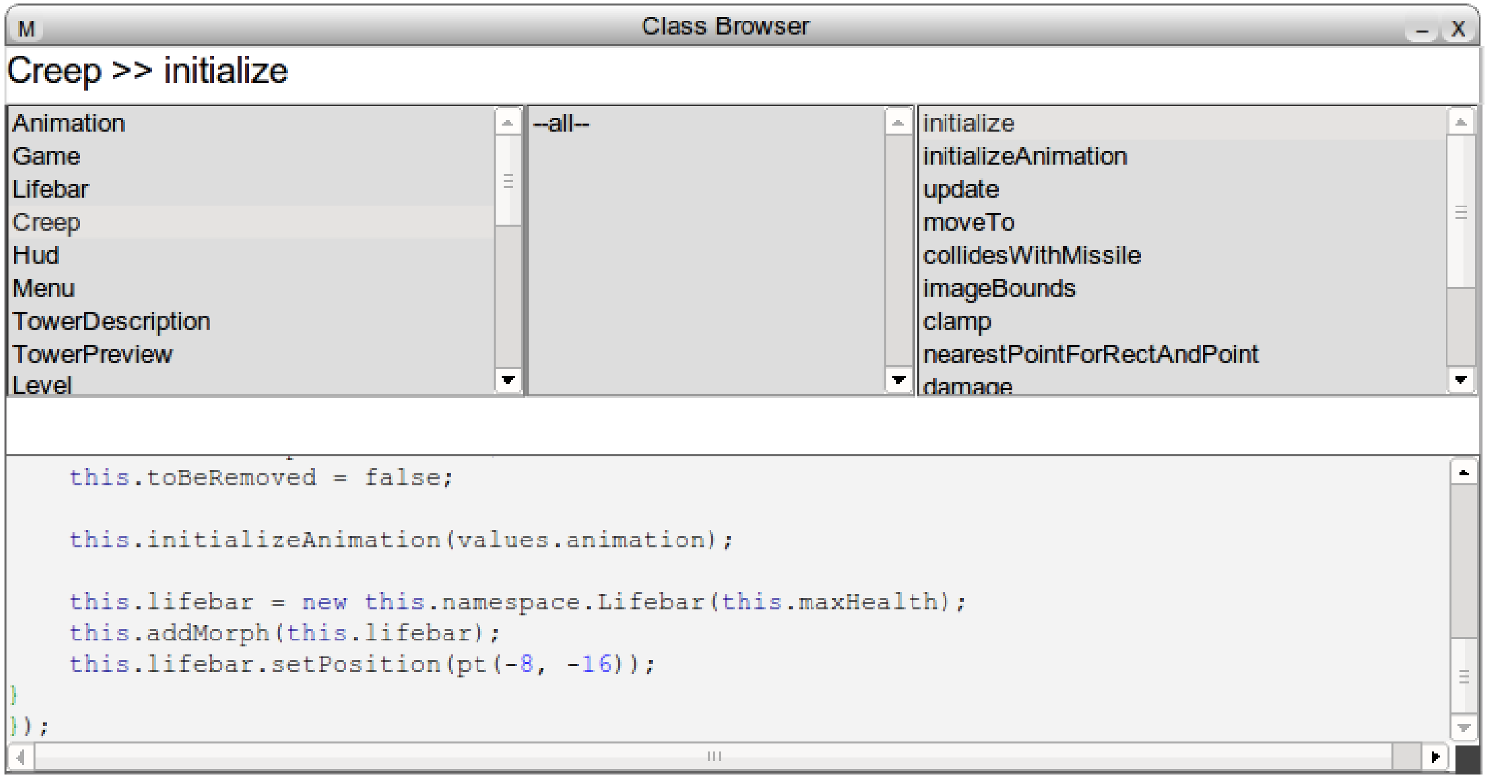
- Private class is stored in a slot of the Part/Morph so that
  - this.ClassName is the class
  - new this.ClassName() creates a new instance of the private class



```
// changed at Tue Jul 10 2012 10:08:15 GMT+0200 (CEST) by willy.scheibel
this.addScript(function createNewGame() {
    this.game = new this.Game();
}).tag([]);
```

Using Private Classes within a Private Class

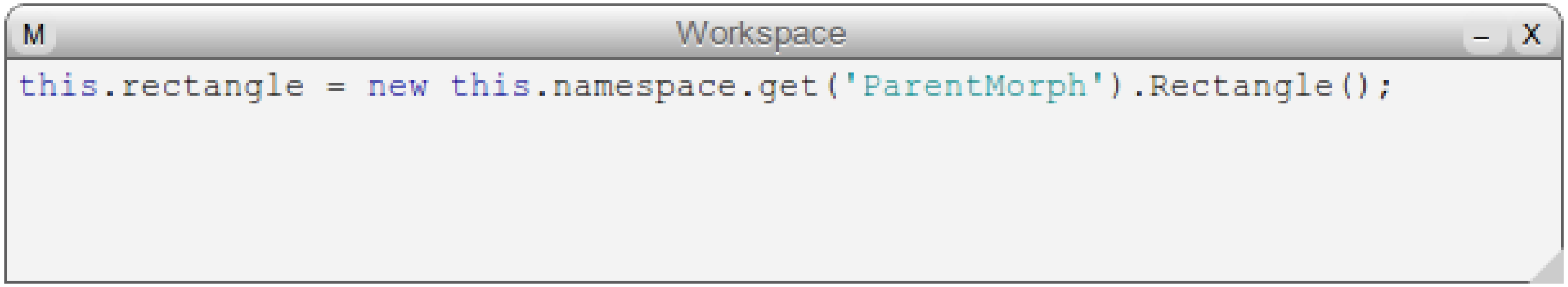
- Each instance of a private class has a slot holding the Morph with the other classes
  - Accessible through this.namespace so that
  - new this.namespace.OtherClass() returns an instance of OtherClass



```
    this.toBeRemoved = false;

    this.initializeAnimation(values.animation);

    this.lifebar = new this.namespace.Lifebar(this.maxHealth);
    this.addMorph(this.lifebar);
    this.lifebar.setPosition(pt(-8, -16));
}
});
```

Using Private Classes across Morphs

- Other Morphs can be found using object traversal
  - this.namespace is the original Morph
    - this.namespace.owner
    - this.namespace.get(“name”)



```
this.rectangle = new this.namespace.get('ParentMorph').Rectangle();
```

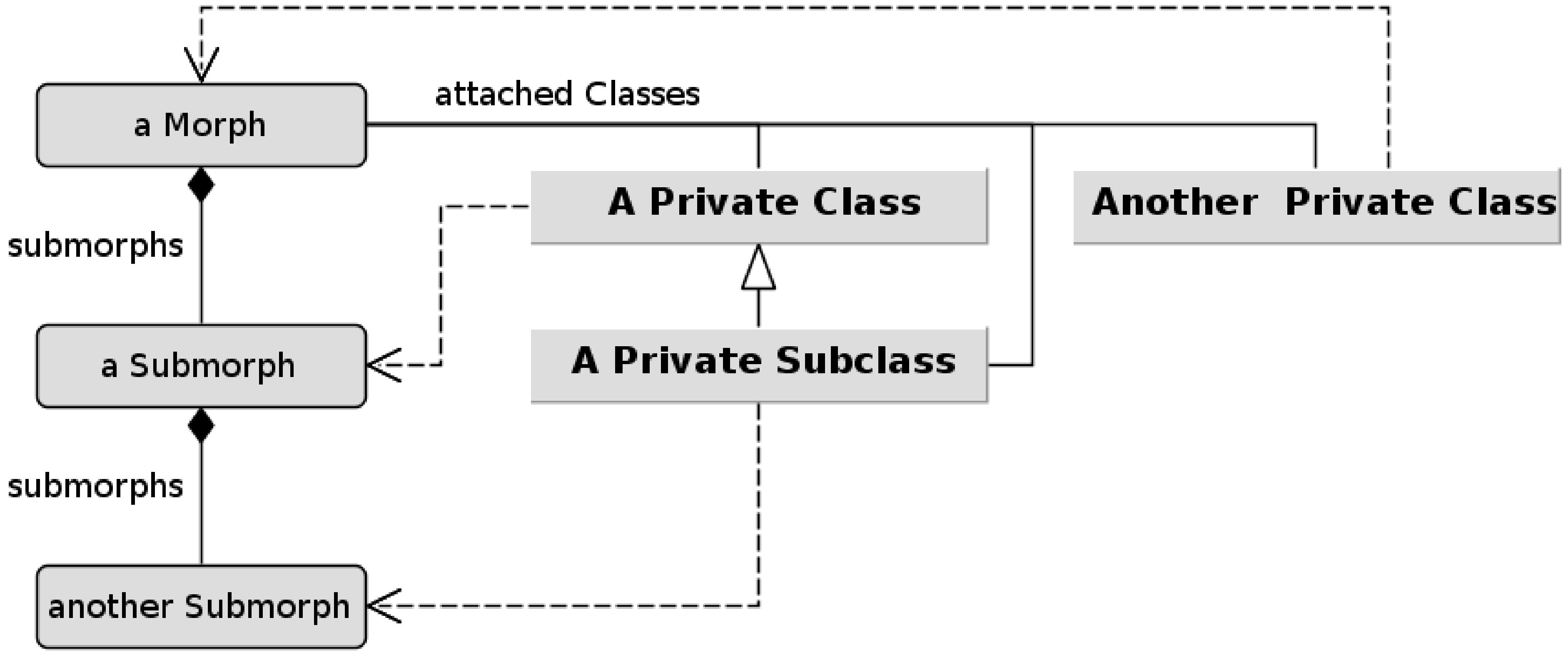
Future Work

- Extend Object Editor
- Serialize not only prototype but also the class variables and methods
- Handle class extensions
- Improve interface to access private classes
  - $namespace instead of this.namespace

Conclusion

- Private Classes can be
  - attached to Morphs
  - saved in Parts and the World by using a Serializer plugin
  - accessed from anywhere in the Part
  - subclassed by other private classes

- Tooling support in form of a class browser to
  - view the private classes of a Morph
  - enable editing the private classes
    - Adding classes and methods
    - Changing superclass
    - Change methods


a Morph
attached Classes
A Private Class
Another Private Class
submorphs
a Submorph
A Private Subclass
submorphs
another Submorph