Macoun'10

Cocos2D mit Komponenten

Steffen Itterheim
Freelance Police

Ablauf

- Komponenten: wieso, weshalb, warum?
- Umsetzung Komponentensystem
- Besonderheiten
- Vor-/Nachteile, Möglichkeiten

Komponenten:
wieso, weshalb, warum?

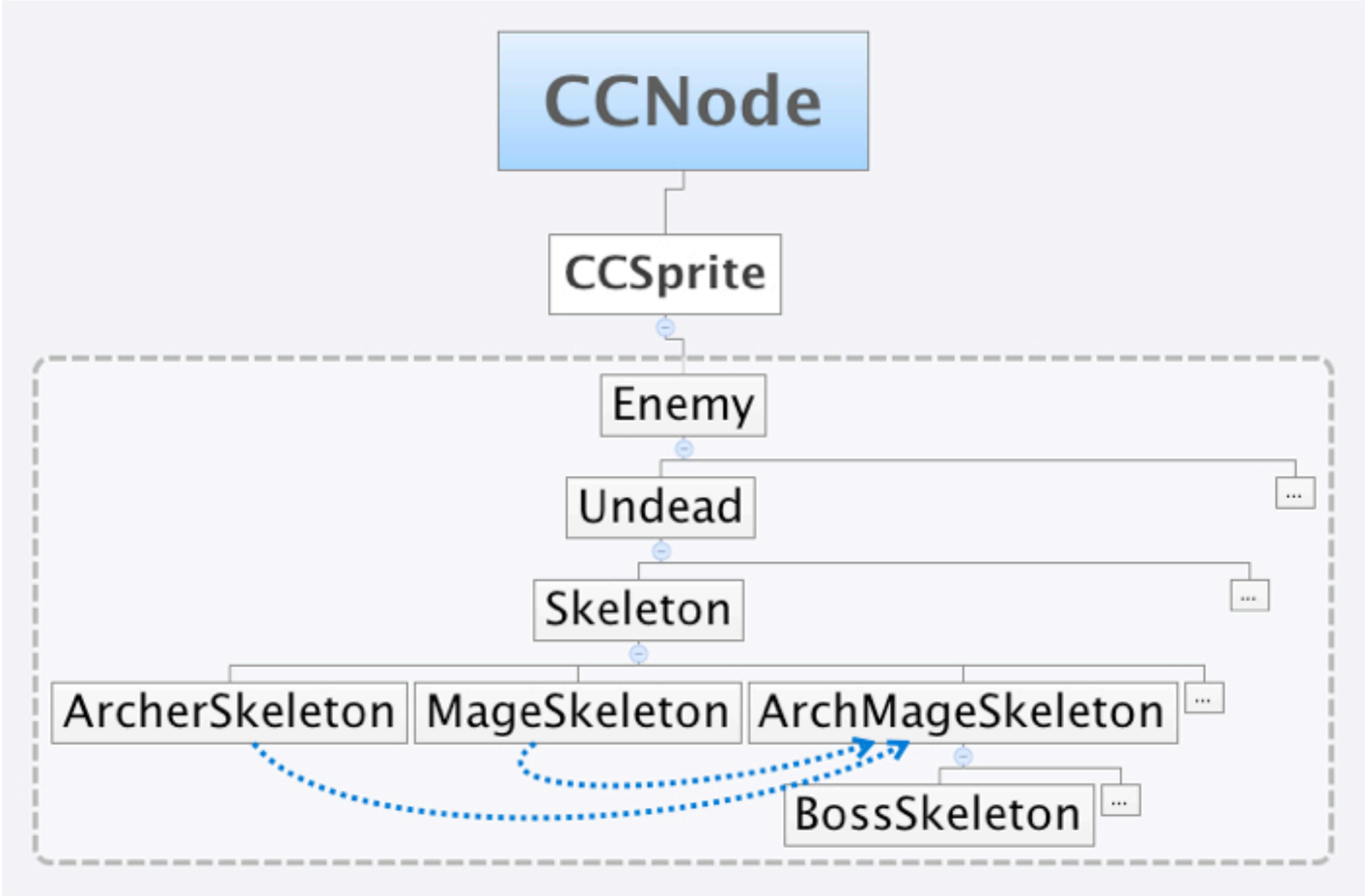
Cocos2D für iPhone

- Open Source 2D Game Engine
- Szenenhierarchie, Basisklasse CCNode

Nachteile

- Spiel-Logik?
- Verleitet zu Klassenhierarchie

Ver-Extrem-Erbung



Vererbung ist OK ...

- in einfachen Spielen (z.b. Sneezies, Paper Toss)
- in klar definierbaren Frameworks (z.b. Qt)

Spieleprogrammierung ist ...

- nicht einfach
- selten vorab klar definierbar

Vererbung: Nachteile

- enge Kopplung
- unflexibel
- API Ballast: vererbte Properties & Methoden

Ballaststoffe

v0.99.5 beta 3	CCNode	CCSprite
Properties	21	36
Methoden	46	79
Instanz-Grösse	220 Bytes	420 Bytes

Die paar Bytes ...

- 420 Bytes entspricht 105 Variablen (32-Bit)
  - für ein Sprite!!

Eins vorweg

- Cocos2D Ballast bleibt ...
- aber: gekapselt.

Die “Großen” haben’s!

- iOS Engines mit ...
  - Unity, iTorque, Shiva
- ... und ohne Komponentensystem:
  - Cocos2D, SIO2, Sparrow, Oolong

Komponentensystem für Cocos2D

Basisklasse: “Entity”

- Container für Komponenten (CCArray)
- Erzeugt, verifiziert & aktualisiert Komponenten
- Instanzgrösse: 16 Bytes

WNEntity

```
@interface WNEntity : NSObject
{
    WNNodeComponent* nodeComponent_; // weak ref

    CCArray* components_;
}
```

Komponenten

- modulare, wiederverwendbare “Plugin” Klassen
- @protocol: `init & update:(ccTime)delta`
- Instanzgröße: ~ 24-64 Bytes

WNComponent

```
@interface WNComponent : NSObject <WNComponentProtocol, NSCopying>
{
    WNEntity* owner_;

    int tag_;

    bool isEnabled_;
    bool isInitialized_;
}
```

isEnabled?

- An-/Ausschalten
- vs.
- Erzeugen, Hinzufügen / Entfernen, Freigeben

Node Komponenten

- Cocos2D Integration
- Proxies für CCNode Klassen
- mehrere pro Entity

Verwaltet CCNodes

```
@interface WNNodeComponent : WNComponent
{
   CCNode* node_;

   CCNode* parentNode_; // weak ref to parent node
   WNNodeComponent* parentNodeComponent_; // weak ref to parent comp
}
```

WNScene

- abgeleitet von CCScene
    - wegen: Nodehierarchie, Szenenwechsel
- Pause & Resume support
- enthält den EntityPool

WNEntityPool

- Container für Entities (CCArray)
- Erzeugt Entities
    - initialisiert Entity mit Komponenten

[self reminder];

- Hier demo zeigen!

Hervorzuheben

- kein alloc / init / addChild
- Entity wie CCNode
- Komponenten flexibel kombinierbar

Rekapitulierung

- WNScene: enthält Entity Pool
- WNEntityPool: verwaltet Entities
- WNEntity: enthält Komponenten
- WNComponent: steuern Spiellogik
- WNNodeComponent: Proxies für CCNode, CCSprite, usw.

Besonderheiten

- Message Forwarding
- Dependency Injection
- Template Entities
- Typenlose Notifications

Message Forwarding

- CCNode Facade:
  - WNEntity reicht Messages weiter
- gleichbedeutend:
  - `entity.rotation = 90;`
  - `entity.node.rotation = 90;`

```
-(NSMethodSignature*) methodSignatureForSelector:(SEL)s
{
  NSMethodSignature* signature = [super methodSignatureForSelector:s];
  if (signature == nil)
  {
    signature = [self.node methodSignatureForSelector:s];
  }
  return signature;
}
```

- Methodensignatur von node holen

```
-(void) forwardInvocation:(NSInvocation*)anInvocation
{
  if ([self.node respondsToSelector:[anInvocation selector]])
  {
    [anInvocation invokeWithTarget:self.node];
  }
  else
  {
    [super forwardInvocation:anInvocation];
  }
}
```

- Message an node weiterleiten

```
@interface WNEntity (CCNodeForwarding)
@property (nonatomic) CGPoint position;
@property (nonatomic) float rotation;
@property (nonatomic) float scale;
@property (nonatomic) int tag;
...
-(CCAction*) runAction: (CCAction*) action;
-(void) stopAllActions;
-(void) stopAction: (CCAction*) action;
-(void) stopActionByTag:(int) tag;
-(CCAction*) getActionByTag:(int) tag;
...
@end
```

- Nachteil: CCNode @interface in WNEntity
- Automatisieren!

Dependency Injection

- “Hintertür”
- Component* iVar Zuweisung
- abwägen:
    - Performance++;
    - Dependency--;

```
-(void) tryInjectComponents:(WNComponent*)component
{
   unsigned int count;
   objc_property_t* properties =
                     class_copyPropertyList([component class], &count);
   for (unsigned int i = 0; i < count; i++)
   {
      ...
   }

   free(properties);
}
```

- Properties jeder Komponente prüfen...

```
for (unsigned int i = 0; i < count; i++)
{
   objc_property_t property = properties[i];

   NSString* attributesString = [NSString
               stringWithCString:property_getAttributes(property)
               encoding:NSASCIIStringEncoding];
   NSString* className = [self
               getClassNameByAttributesString:attributesString];

   Class class = NSClassFromString(className);
   if ([class conformsToProtocol:@protocol(WNComponentProtocol)])
   {
      ...
   }
}
```

- Ist property: id<WNComponentProtocol> ?

```
if ([class conformsToProtocol:@protocol(WNComponentProtocol)])
{
   WNComponent* requestedComponent = [self getComponentByClass:class];

   if (requestedComponent == nil)
   {
      [NSException raise:NSInternalInconsistencyException
                  format:@"component not part of entity!"];
   }

   const char* propertyName = property_getName(property);
   object_setInstanceVariable(component, propertyName,
                                          requestedComponent);
}
```

- iVar zuweisen, da @property readonly

Brutalo-Methode

- Injection ist forciert
  - Gesucht: .NET Attribute
- Mehrere Komponenten?

Entity Templates

- Entity + Komponenten = Template (Setup)
    - NodeKomponenten: verzögerte Initialisierung
- Templates ersetzen Klassenhierarchie

Beispiel: Template

```
WNEntityTemplate* meinMonsterTemplate =
                [WNEntityTemplate entityTemplateWithComponents: sprite,
                 explodeOnTouch, explosionAnim, monsterBehavior,
                 weaponSprite, fartSound, canMoonWalk, cantDance, nil];

[WNEntityTemplatePool addTemplate:meinMonsterTemplate
                                                    name:@"meinMonster"];

WNEntity* meinMonster =
                [WNEntityPool createEntityFromTemplate:@"meinMonster"];
```

- Template: nicht initialisiert, nur Daten

Notifications

- Typenlose Kommunikation
- NSNotificationCenter

```
// senden
[self sendNotification:onTouchedNotificationName];
[self sendNotification:onEnableButton userInfo:userDictionary];

// registrieren
[self registerForNotification:onEnableButton sender:nil
                                        selector:@selector(setButtonEnabled:)];
[[NSNotificationCenter defaultCenter] addObserver:self
  selector:@selector(onButton:) name:@"MyButtonActivated" object:nil];

// empfangen
-(void) setButtonEnabled:(NSNotification*)notification
{
   [self setMenuItemEnabled:YES];
}
```

- flexibel, typenlos, direkt

Probleme, Lösungen, Möglichkeiten

Vorteile

- Vereinheitlichte API
- reduzierte Abhängigkeiten:
    - eher generischer Code
    - schnelles prototyping
- Injection: keine nil pointer Probleme
- CCNode Facade: erleichtert Umgewöhnung

Nachteile

- Zusammenhänge schwerer zu durchschauen
- @interface Generierung für Facade
- Node Komponenten: Pflege

Möglichkeiten

- datengetrieben ==> Editor
- Standard-Komponenten:
    - “Fundgrube” / “Tauschbörse”
- Cocos2D entschlacken?
    - Texturkomponente ...

Fragen?

Vielen Dank!

Macoun'10