



# Orxonox

## Framework & Coding

Einleitung in das Framework von Orxonox



# Orxonox

## Libraries

- OGRE (Grafikengine)





# Orxonox

## Libraries

- OGRE (Grafikengine)





# Orxonox

## Libraries

- OGRE (Grafikengine)





# Orxonox

## Libraries

- CEGui (GUI-Engine)

The screenshot displays the Particle Editor v0.06 interface. The central view shows a 3D rendering of a vertical flame particle system against a dark background with a planet and moon. The interface is divided into several panels:

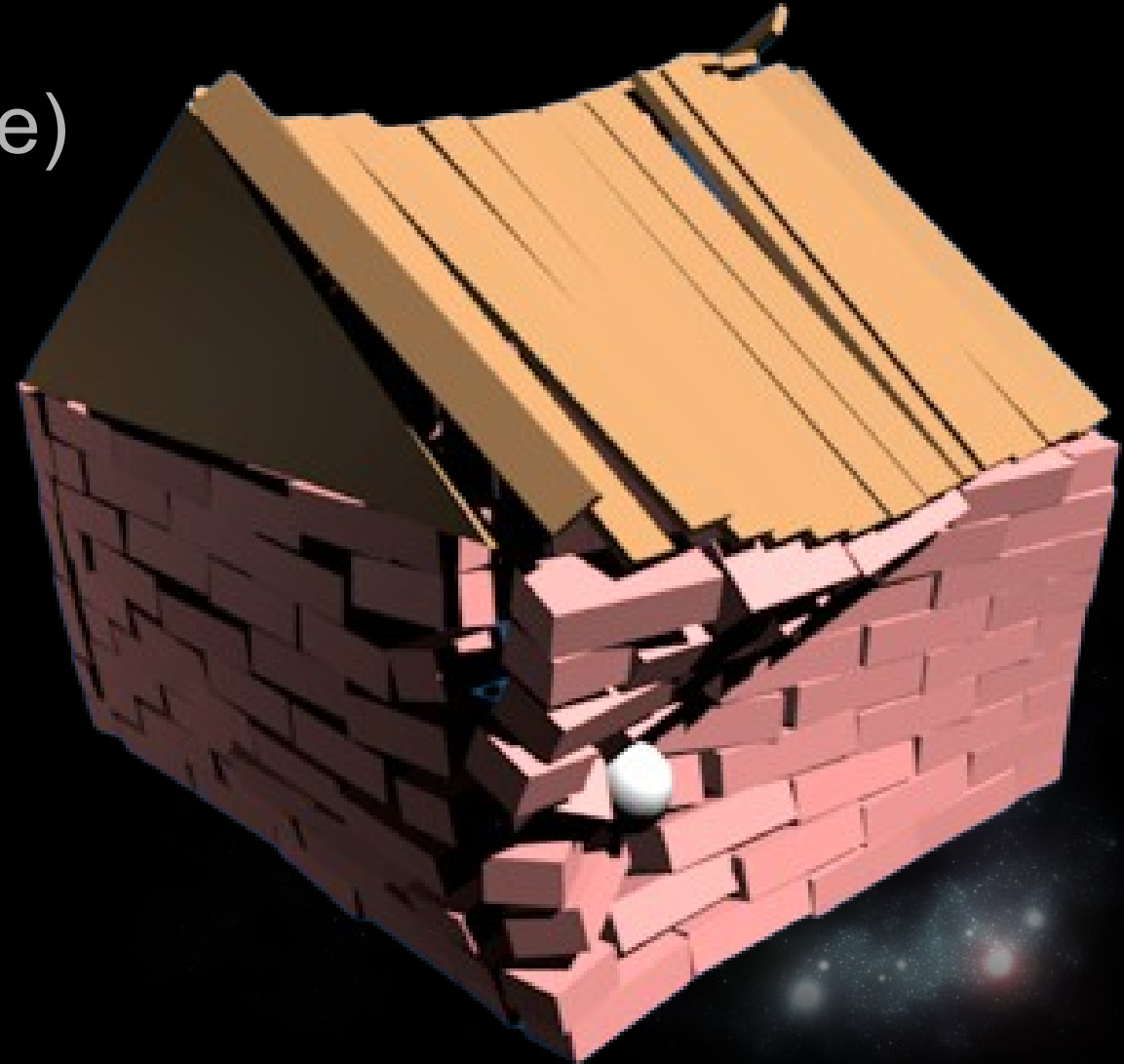
- Template Management:** A panel with tabs for 'Selected', 'Available', and 'New Template'. It lists 'Available Particle System Templates' with columns for 'Script' and 'Template'. The selected template is 'PEExamples/flame'. Buttons for 'Unload All' and 'Load Exclusive' are at the bottom.
- Editor Options:** A panel below the template management, currently empty.
- Particle System Parameters:** A panel on the right with tabs for 'Basic', 'Emitter 1', 'Scaler', and 'ColourFader2'. It contains various parameters such as 'quota' (500), 'material' (PE/lensflare), 'particle\_width' (2), 'particle\_height' (12), 'cull\_each' (false), 'billboard\_type' (point), and 'common\_direction' (3.4021, 3.4021, 3.4021).
- PEExamples/flame - Basic Parameters:** A sub-panel at the bottom right with buttons for 'Randomise', 'Restore', 'Box', 'Add Emitter', and 'ColourFader', 'Add Affector'.
- Performance Metrics:** A panel at the bottom left showing 'Current FPS: 224.652', 'Average FPS: 243.055', 'Worst FPS: 0.356506 2805 ms', 'Best FPS: 578.842 1 ms', and 'Triangle Count: 2346'.
- Status Bar:** At the bottom center, it says 'Exclusively Loaded partide system: PEExamples/flame'.
- OGRE Logo:** The OGRE logo is visible in the bottom right corner.



# Orxonox

## Libraries

- Bullet (Physikengine)





# Orxonox

## Klassenhierarchie - OrxonoxClass

- alle Klassen und Interfaces erben von OrxonoxClass
- notwendig für das Funktionieren des Frameworks -> Servicefunktionalität



# Orxonox

## Klassenhierarchie - Baseobject

- Basisklasse aller „Objects“ in Orxonox
- Objects:
  - Können in ein Level geladen werden.
  - Werden am Ende des Levels wieder gelöscht.





# Orxonox

## Objekteigenschaften - Raum

- jedes Objekt, das im Level einen Ort hat, erbt von der Basisklasse „Worldentity“.
  - jedes Objekt, das man sehen kann
- Worldentity: Punkt und Vektor im Raum
- Worldentities:
  - Statisch: StaticEntity (z.B. eine Spacestation)
  - Beweglich: MovableEntity (z.B. ein Projektil)
  - Kontrolliert: ControllableEntity (z.B. ein Spaceship)



# Orxonox

## Worldentities: Definition

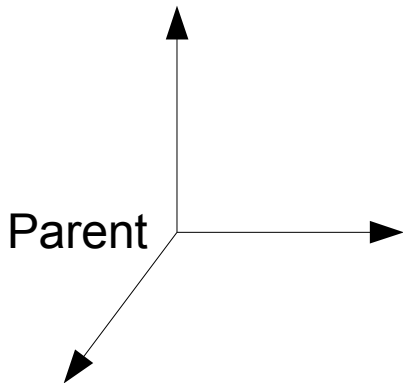
- Worldentities können aneinander „attached“ werden, d.h. man kann sie zusammenhängen. Die Position des angehängten Objekts (Child) ist dann relativ zur Position und Rotation des Basisobjekts (Parent).



# Orxonox

## Worldentities: Attachen

- Absolute Position im Raum (Parent):

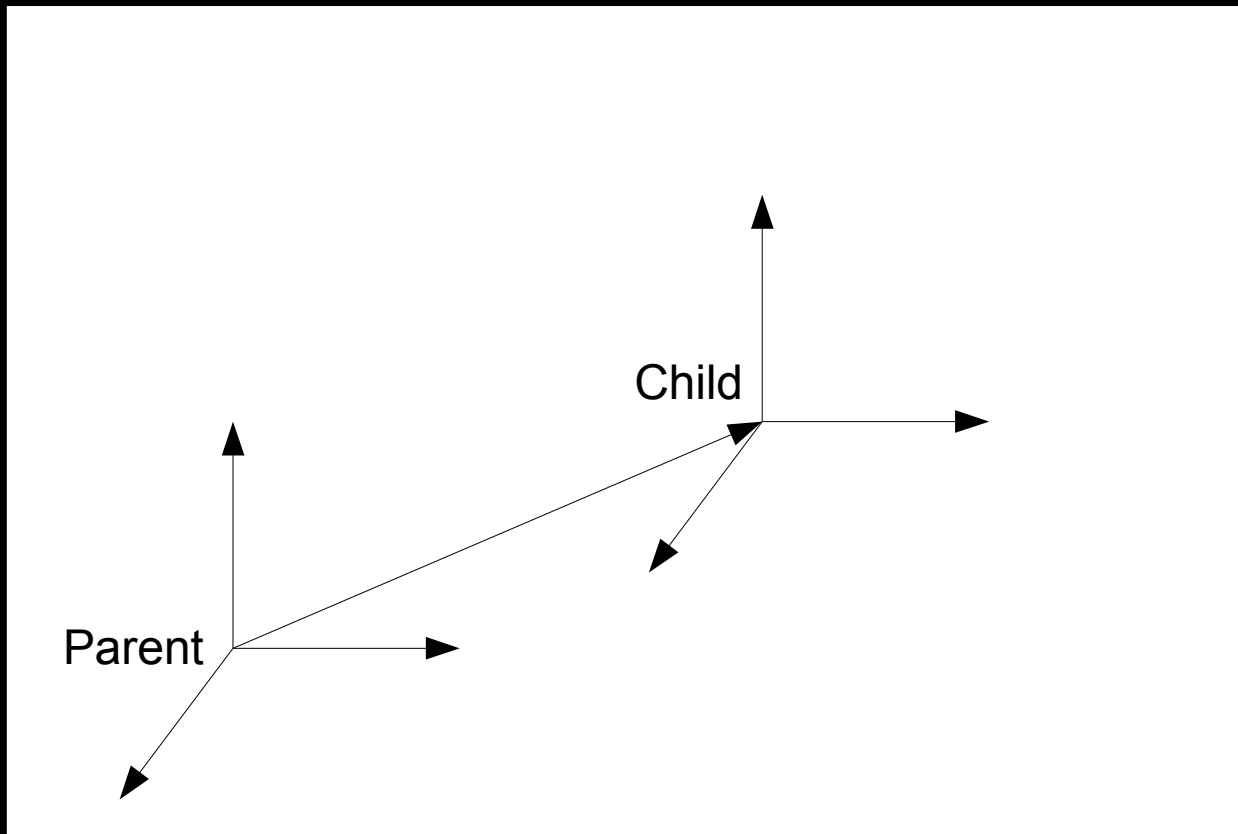




# Orxonox

## Worldentities: Attachen

- Relative Position im Raum (Child):

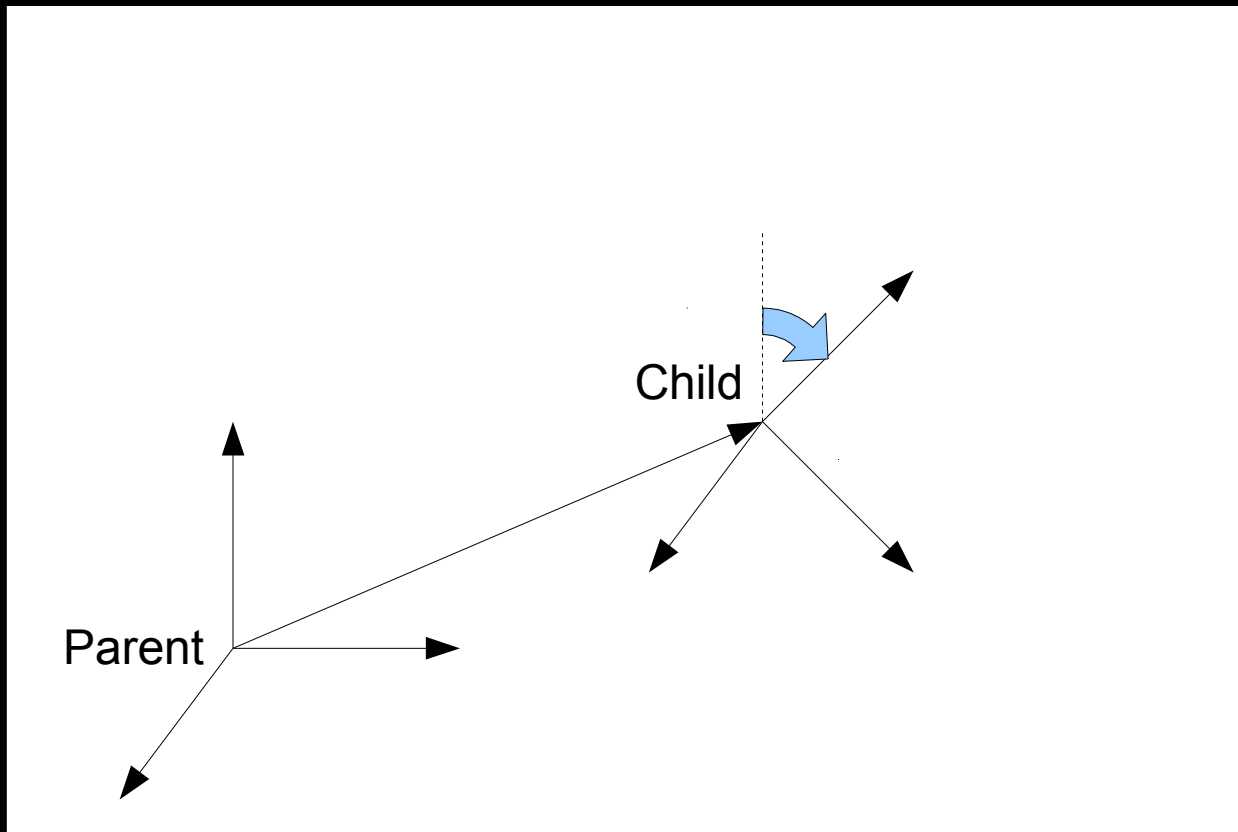




# Orxonox

## Worldentities: Attachen

- Rotation des Child:

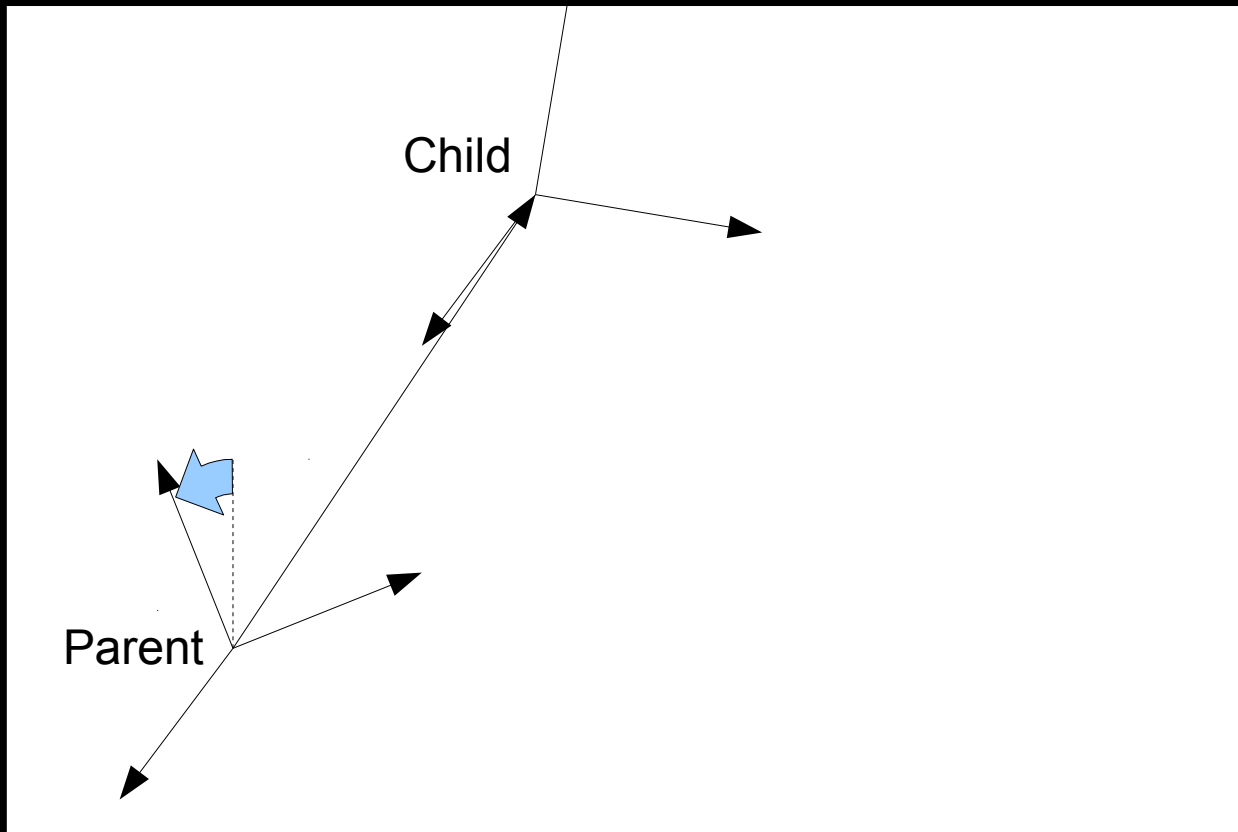




# Orxonox

## Worldentities: Attachen

- Rotation des Parent:





# Orxonox

## Objekteigenschaften - Zeit

- ein Objekt ändert sich in der Zeit -> es muss vom Interface Tickable erben.
- Klassen die von Tickable erben, erben die Funktion tick(float dt).
- ein Frame wird gerendert -> tick(dt) wird aufgerufen
- dt: die Zeit seit dem letzten Aufruf von tick(dt)



# Orxonox

## Objektmodellierung

- is–a–Relation:
  - ein Spaceship ist ein Worldentity
  - Mechanismus: Vererbung
- has–a–Relation:
  - „ein Spaceship hat Waffen, einen Antrieb, ...“
  - Mechanismus: Pointer auf ein anderes Objekt

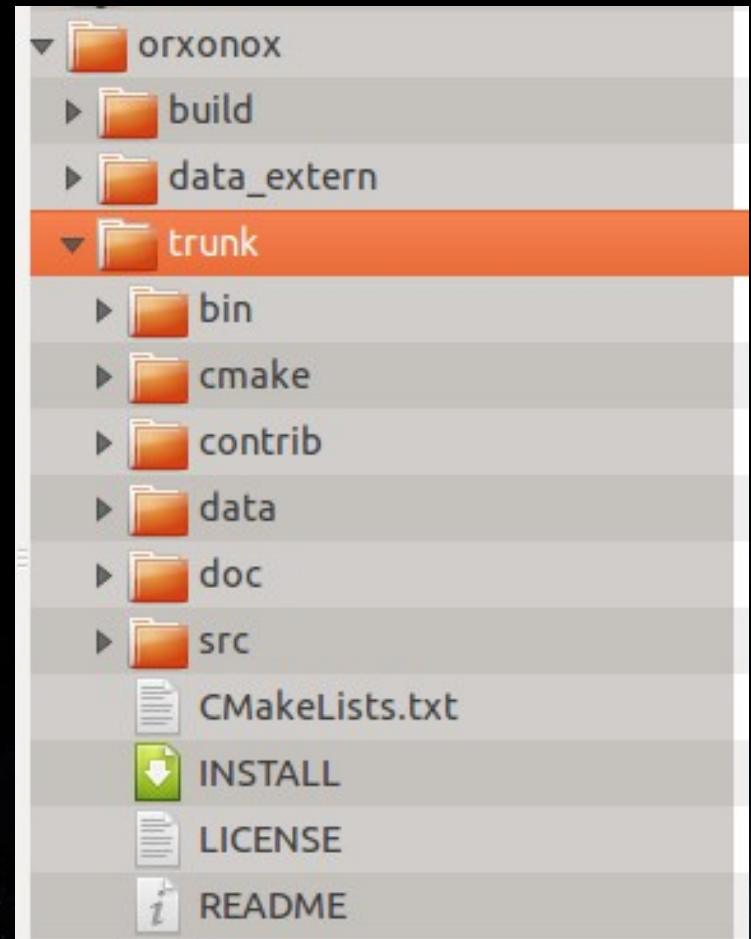




# Orxonox

## Verzeichnisstruktur - Trunk

- cmake: CMake Scripts
- data: XML und Lua Scripts
- src: Quellcode





# Orxonox

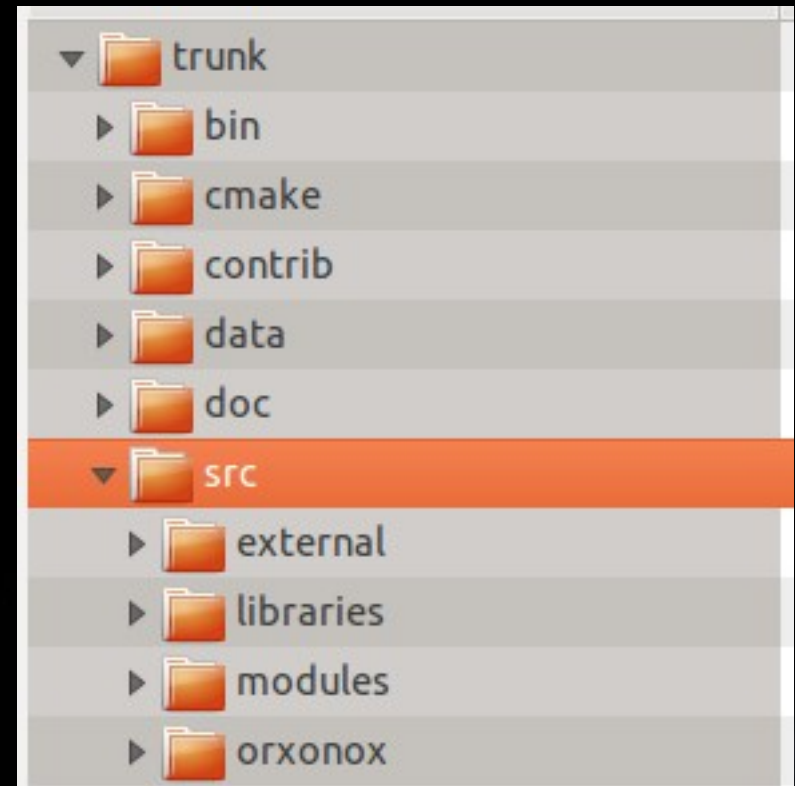
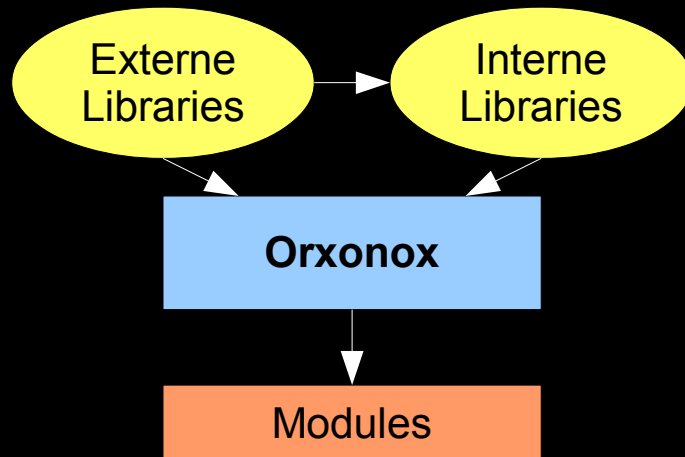
## CMake

- Findet die benötigten Libraries
- Erstellt ein Makefile
- Kann IDE-Projekt-Dateien erstellen



# Orxonox

## Struktur src

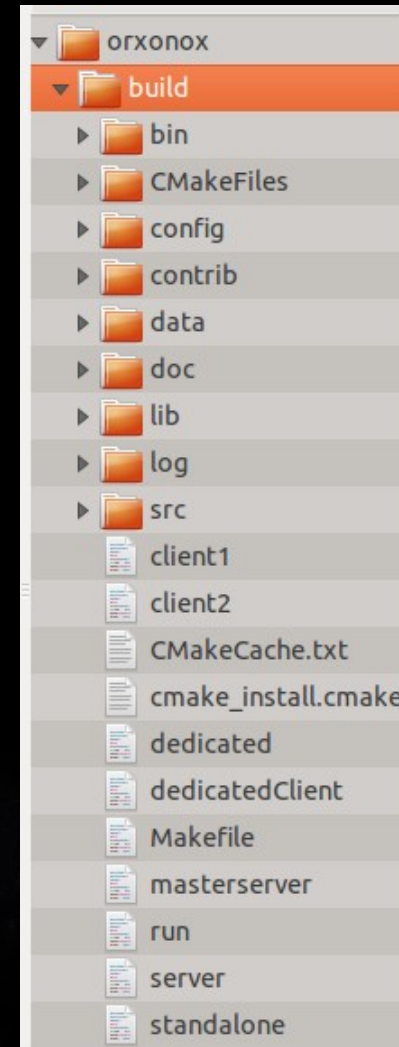




# Orxonox

## Verzeichnisstruktur - Build

- bin: Executables
- config: Config-Files
- log: Output
- run: Startet Orxonox (runscript)





# Orxonox

## Beispielklasse: CMakeLists.txt

- Wir erstellen zwei neue Dateien, MyClass.cc (das Source-File) sowie MyClass.h (das Header-File).
- Im gleichen Ordner in dem wir die Files erstellt haben, suchen wir die Datei „CMakeLists.txt“ und suchen nach einer Liste von anderen Source-Files. Dort Tragen wir MyClass.cc an einer beliebigen Stelle ein.
- Dadurch wird sichergestellt, dass unser neues File kompiliert wird.



# Orxonox

## Beispielklasse: Header

- Im Header-File Deklarieren wir die neue Klasse:

```
class MyClass : public MovableEntity
{
    public:
        MyClass(Context* context);
        virtual ~MyClass();

        virtual void tick(float dt);
};
```

- Unsere Klasse erbt also von MovableEntity (eine bewegliche WorldEntity).
- Da MovableEntity ausserdem vom Interface Tickable erbt, erbt auch unsere Klasse die Tick-Funktion.



# Orxonox

## Beispielklasse: Source

Im Source-File Implementieren wir das Grundgerüst der neuen Klasse:

```
MyClass::MyClass(Context* context)
{
}

MyClass::~MyClass()
{
}

void MyClass::tick(float dt)
{
}
```



# Orxonox

## Beispielklasse: RegisterClass

- Zuerst müssen wir eine Factory erstellen, damit unsere Klasse vom Framework erkannt und auch über XML geladen werden kann:

```
RegisterClass(MyClass) ;  
MyClass::MyClass(Context* context)  
{  
}  
  
MyClass::~MyClass()  
{  
}  
  
void MyClass::tick(float dt)  
{  
}
```





# Orxonox

## Beispielklasse: RegisterObject

- Als nächstes müssen wir direkt zu Beginn des Constructors unser Objekt registrieren:

```
RegisterClass(MyClass);  
MyClass::MyClass(Context* context)  
{  
    RegisterObject(MyClass);  
}  
  
MyClass::~~MyClass()  
{  
}  
  
void MyClass::tick(float dt)  
{  
}
```



# Orxonox

## Beispielklasse: Context

Ausserdem müssen wir den context-Pointer an die Basisklasse weitergeben:

```
RegisterClass(MyClass);  
MyClass::MyClass(Context* context) : MovableEntity(context)  
{  
    RegisterObject(MyClass);  
}  
  
MyClass::~~MyClass()  
{  
}  
  
void MyClass::tick(float dt)  
{  
}
```



# Orxonox

## Beispielklasse: SUPER

Damit auch weiterhin der Tick von MovableEntity aufgerufen wird, müssen wir den Aufruf der Tick-Funktion an die Basisklasse weiterleiten:

```
RegisterClass(MyClass);  
MyClass::MyClass(Context* context) : MovableEntity(context)  
{  
    RegisterObject(MyClass);  
}  
  
MyClass::~~MyClass()  
{  
}  
  
void MyClass::tick(float dt)  
{  
    SUPER(MyClass, tick, dt);  
}
```



# Orxonox

## Beispielklasse: orxout()

Schlussendlich wollen wir noch etwas (sinnlose) Action in die Klasse bringen, daher geben wir in jedem Tick einen Text in die Konsole aus:

```
RegisterClass(MyClass);
MyClass::MyClass(Context* context) : MovableEntity(context)
{
    RegisterObject(MyClass);
}

MyClass::~MyClass()
{
}

void MyClass::tick(float dt)
{
    SUPER(MyClass, tick, dt);

    orxout() << „Hello World“ << endl;
}
```



# Orxonox

## XML

- XML ist eine textbasierte Sprache, um Objekte zu speichern und zu laden.
- XML weist die selbe Form wie HTML auf.
- Wir verwenden XML, um Levels und andere Ansammlungen von Klassen (z.B. HUDs) zu beschreiben.

### • Beispiel:

```
<MyClass myvalue="1" myothervalue="Hello World">  
  <subclasses>  
    <OtherClass somevalue="1.111" />  
    <OtherClass somevalue="2.222" />  
  </subclasses>  
</MyClass>
```



# Orxonox

## XMLPort

- XMLPort ist unser Interface zwischen XML und C++.
- In XMLPort wird definiert, welche Objekte und Attribute in XML beschrieben werden können. Ausserdem werden Funktionen definiert, um diese Attribute lesen und schreiben zu können.
- Für jeden Wert braucht es ein Paar von set- und get-Funktionen. Die set-Funktion setzt den Wert im Objekt, die get-Funktion liest ihn aus.

### • Beispiel:

```
void MyClass::XMLPort(...)
{
    SUPER(MyClass, XMLPort, ...);

    XMLPortParam(MyClass, "myvalue", setValue, getValue, xmlelement, mode);
    XMLPortParam(MyClass, "myothervaluevalue", setOtherValue, get...

    XMLPortObject(MyClass, OtherClass, "subclasses", addSubclass, getSubclass, xmlelement, mode);
}
```



# Orxonox

