

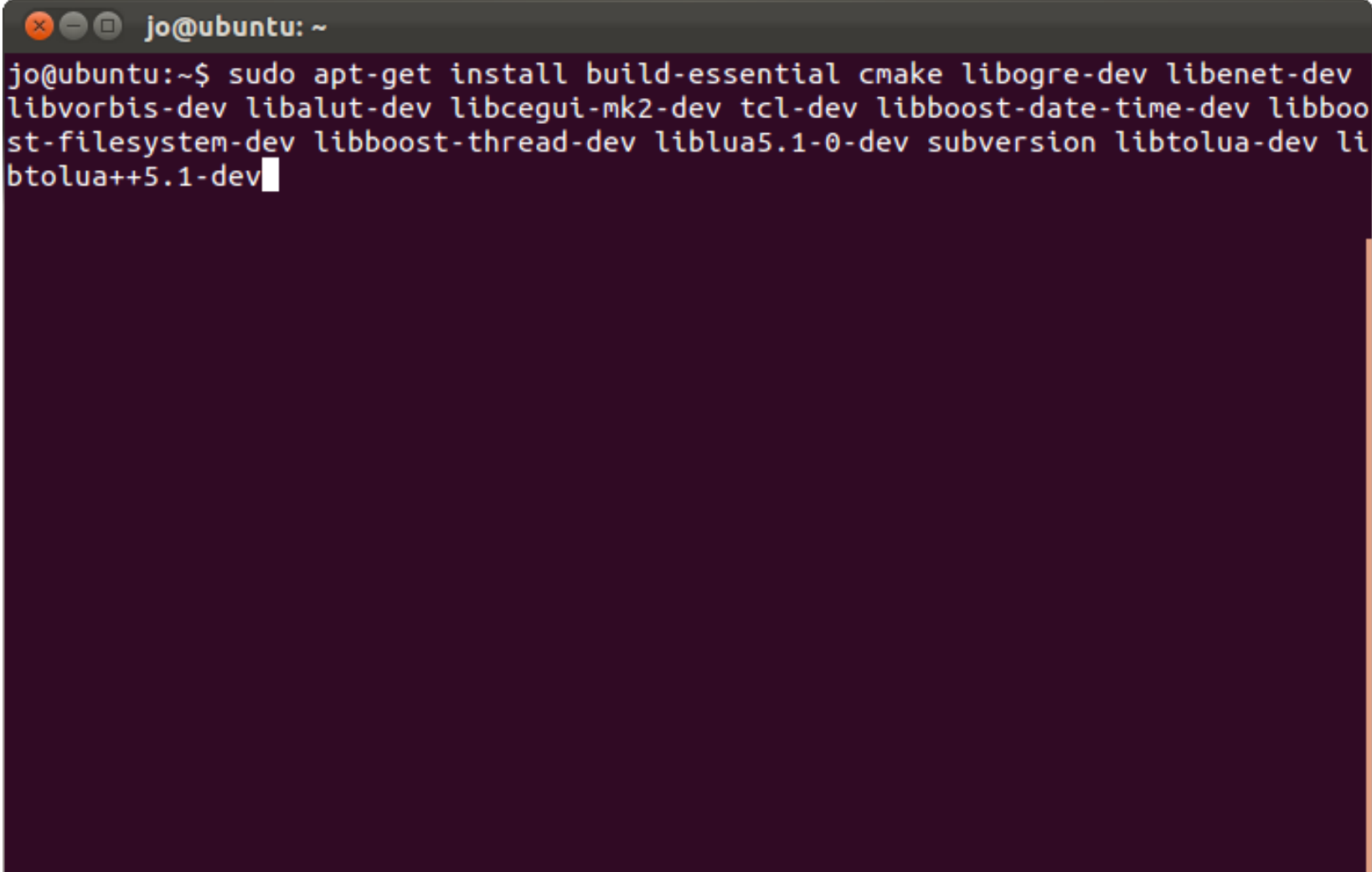
# Framework & Coding

Einleitung in das Framework von Orxonox

# Interaktion

- Bitte geht auf die PPS-Seite und dann auf das „Coding Tutorial“

# Installation – Libraries



```
jo@ubuntu: ~  
jo@ubuntu:~$ sudo apt-get install build-essential cmake libogre-dev libenet-dev  
libvorbis-dev libalut-dev libcegui-mk2-dev tcl-dev libboost-date-time-dev libboo  
st-filesystem-dev libboost-thread-dev liblua5.1-0-dev subversion libtolua-dev li  
btolua++5.1-dev
```

# Libraries

Ogre (Grafikengine)



# Libraries

Ogre (Grafikengine)





# Libraries

Ogre (Grafikengine)



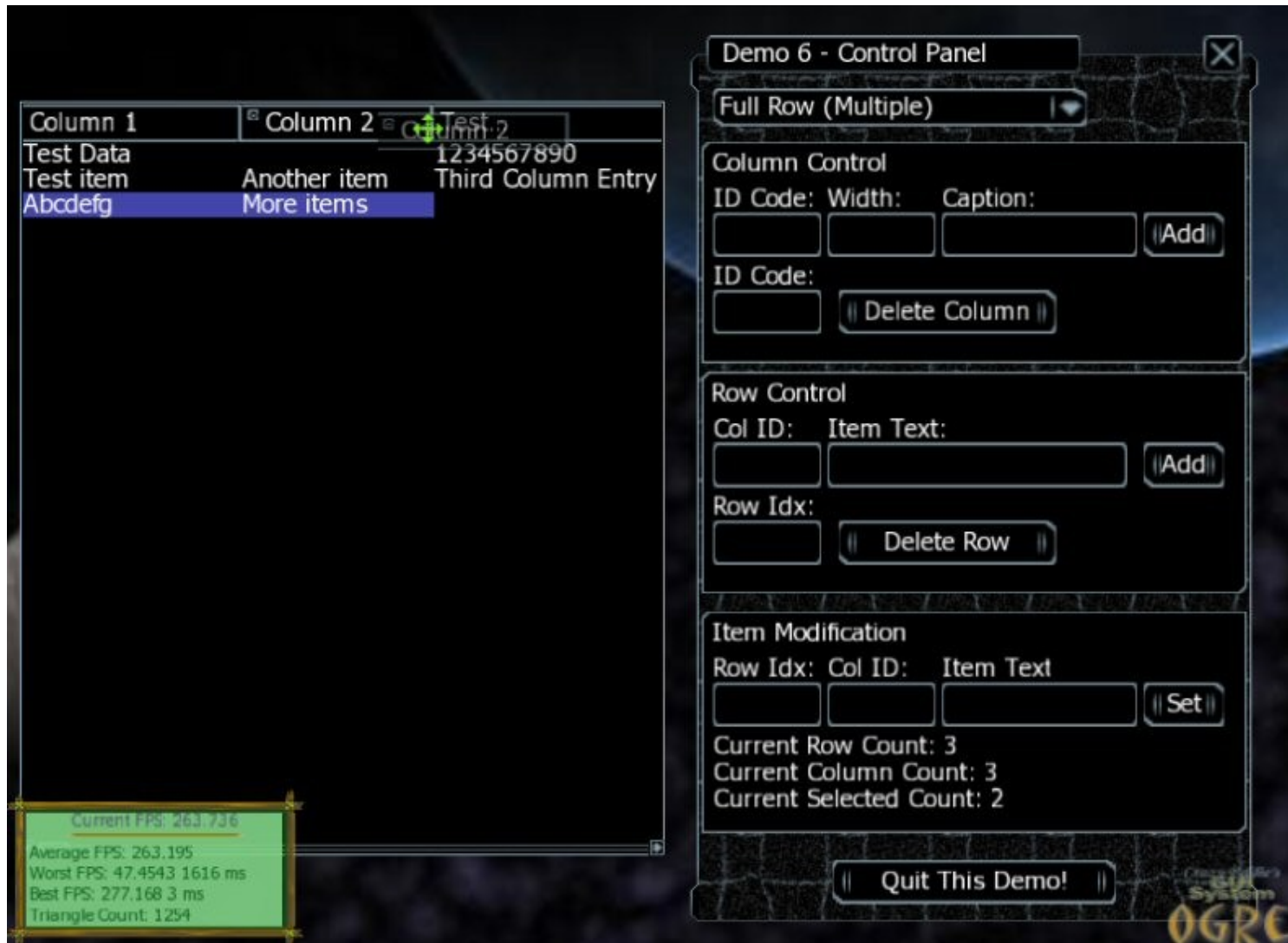
# Libraries

Ogre (Grafikengine)



# Libraries

## CEGui (GUI-Engine)





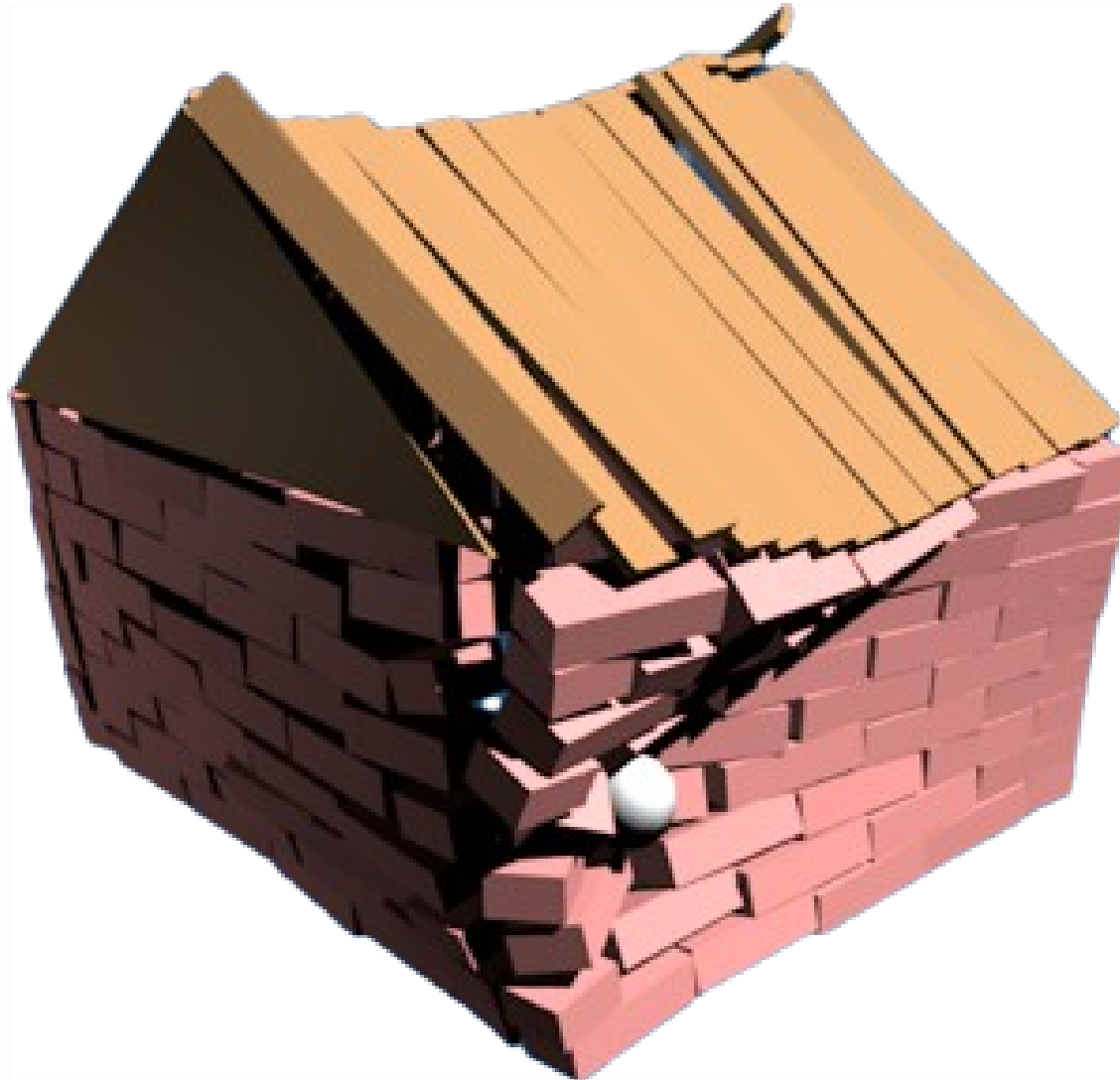
# Libraries

## CEGui (GUI-Engine)

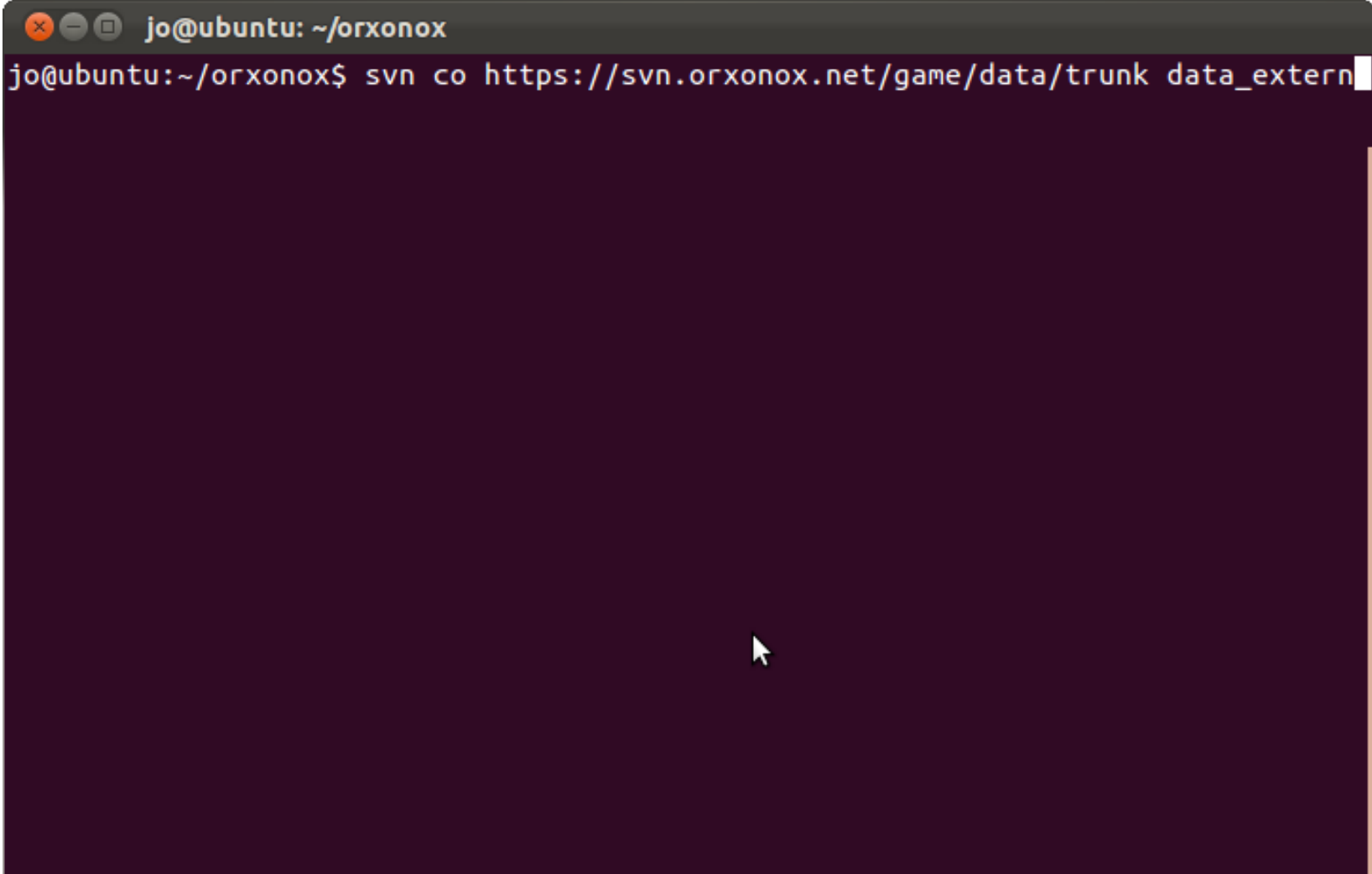


# Libraries

Bullet (Physikengine)



# Download - Content

A terminal window with a dark purple background and a grey title bar. The title bar contains three window control icons (close, minimize, maximize) and the text 'jo@ubuntu: ~/orxonox'. The terminal shows a command prompt 'jo@ubuntu:~/orxonox\$' followed by the command 'svn co https://svn.orxonox.net/game/data/trunk data\_extern'. A white mouse cursor is positioned at the end of the command line.

```
jo@ubuntu: ~/orxonox
jo@ubuntu:~/orxonox$ svn co https://svn.orxonox.net/game/data/trunk data_extern
```

# Download - Code

A terminal window with a dark purple background and a grey title bar. The title bar contains three window control icons (close, minimize, maximize) and the text 'jo@ubuntu: ~/orxonox'. The terminal shows a command prompt 'jo@ubuntu:~/orxonox\$' followed by the command 'svn co http://svn.orxonox.net/game/code/trunk' with a white cursor at the end of the line.

```
jo@ubuntu: ~/orxonox
jo@ubuntu:~/orxonox$ svn co http://svn.orxonox.net/game/code/trunk
```



# Objekteigenschaften - Raum

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

# Framework

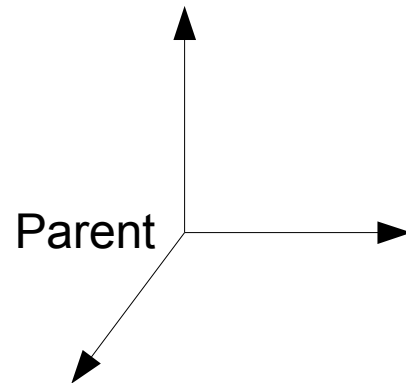
## Worldentities: Definition

- 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).

# Framework

## Worldentities: Attachen

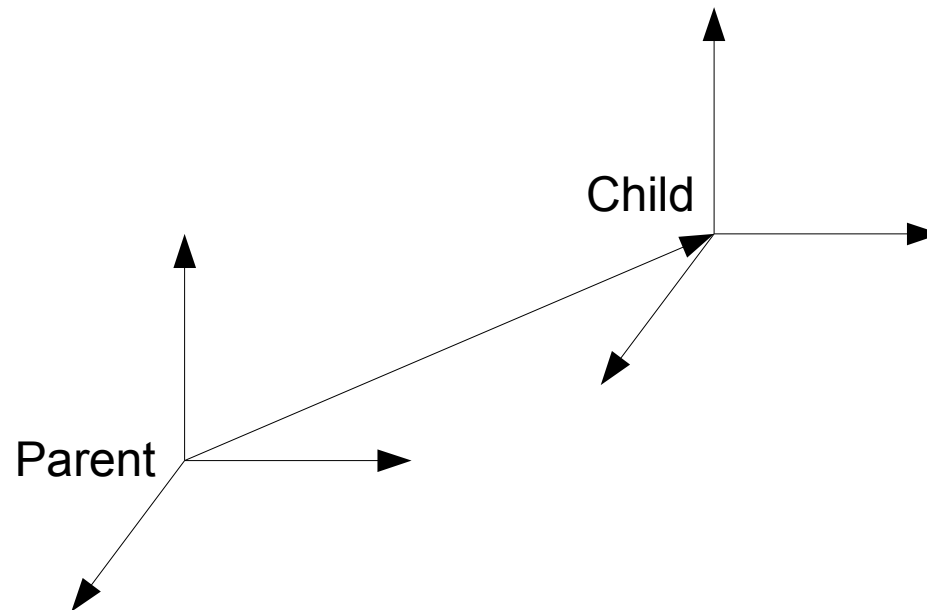
- Absolute Position im Raum (Parent):



# Framework

## Worldentities: Attachen

- Relative Position im Raum (Child):

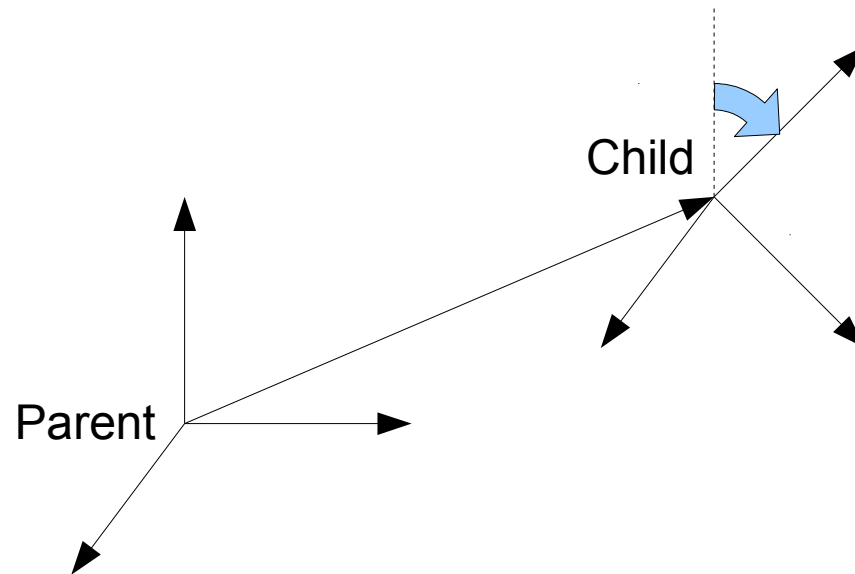




# Framework

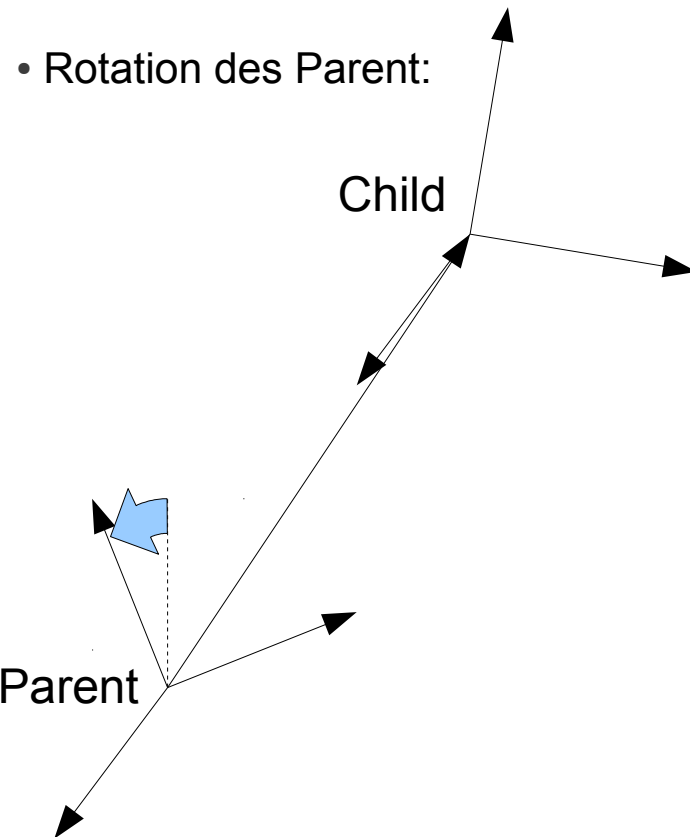
## Worldentities: Attachen

- Rotation des Child:



# Framework

## Worldentities: Attachen



# Objekteigenschaften - Zeit

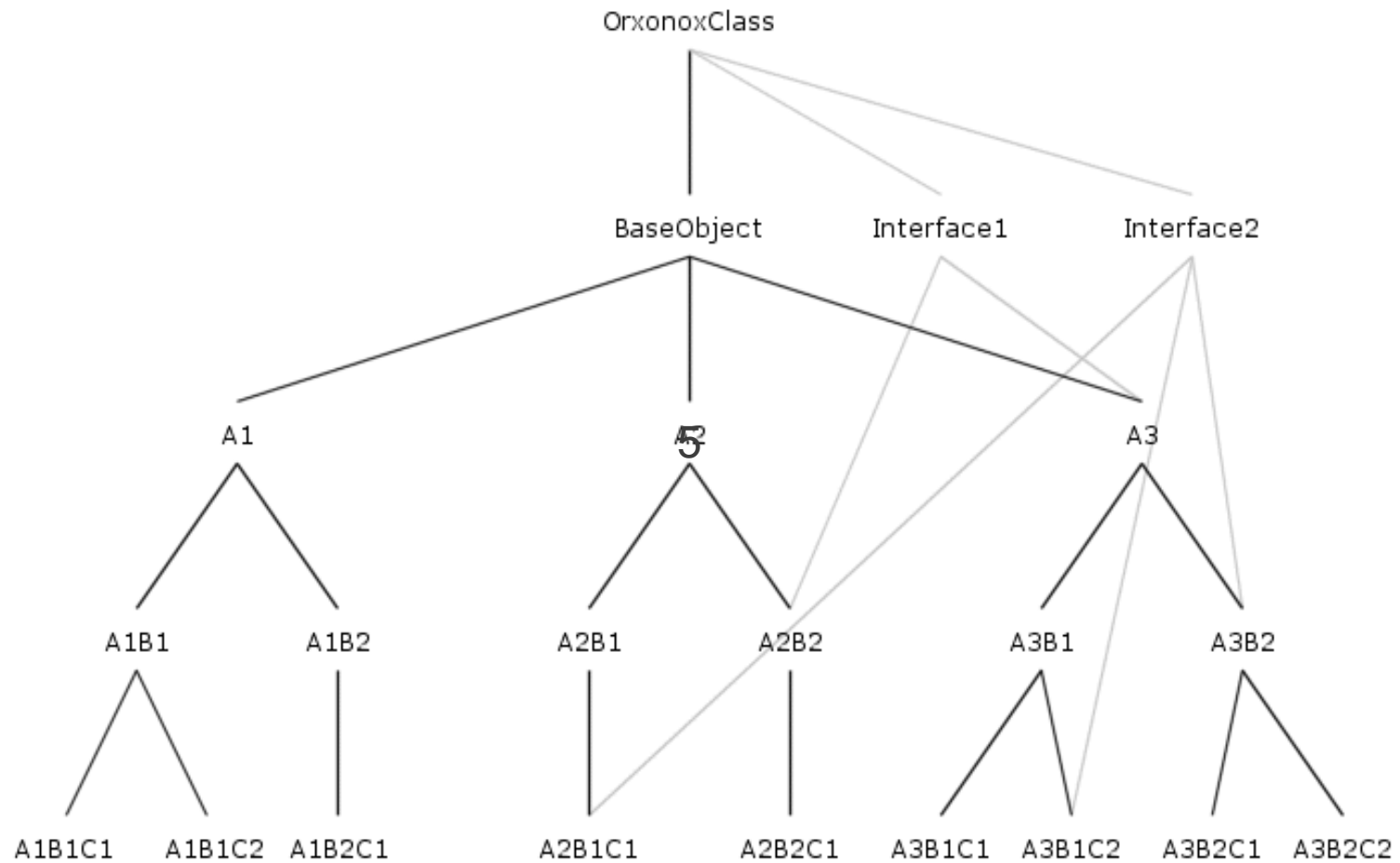
- ein Objekt ändert sich in der Zeit -> es muss vom Interface Tickable erben.
- Klassen die von Tickable erben, erben die Funktion tick(float dt).
- ein Frame wird gerendert -> tick(dt) wird aufgerufen
- dt: die Zeit seit dem letzten Aufruf von tick(dt)

# 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



# Klassenhierarchie



# Klassenhierarchie - OrxonoxClass

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

# Klassenhierarchie - Baseobject

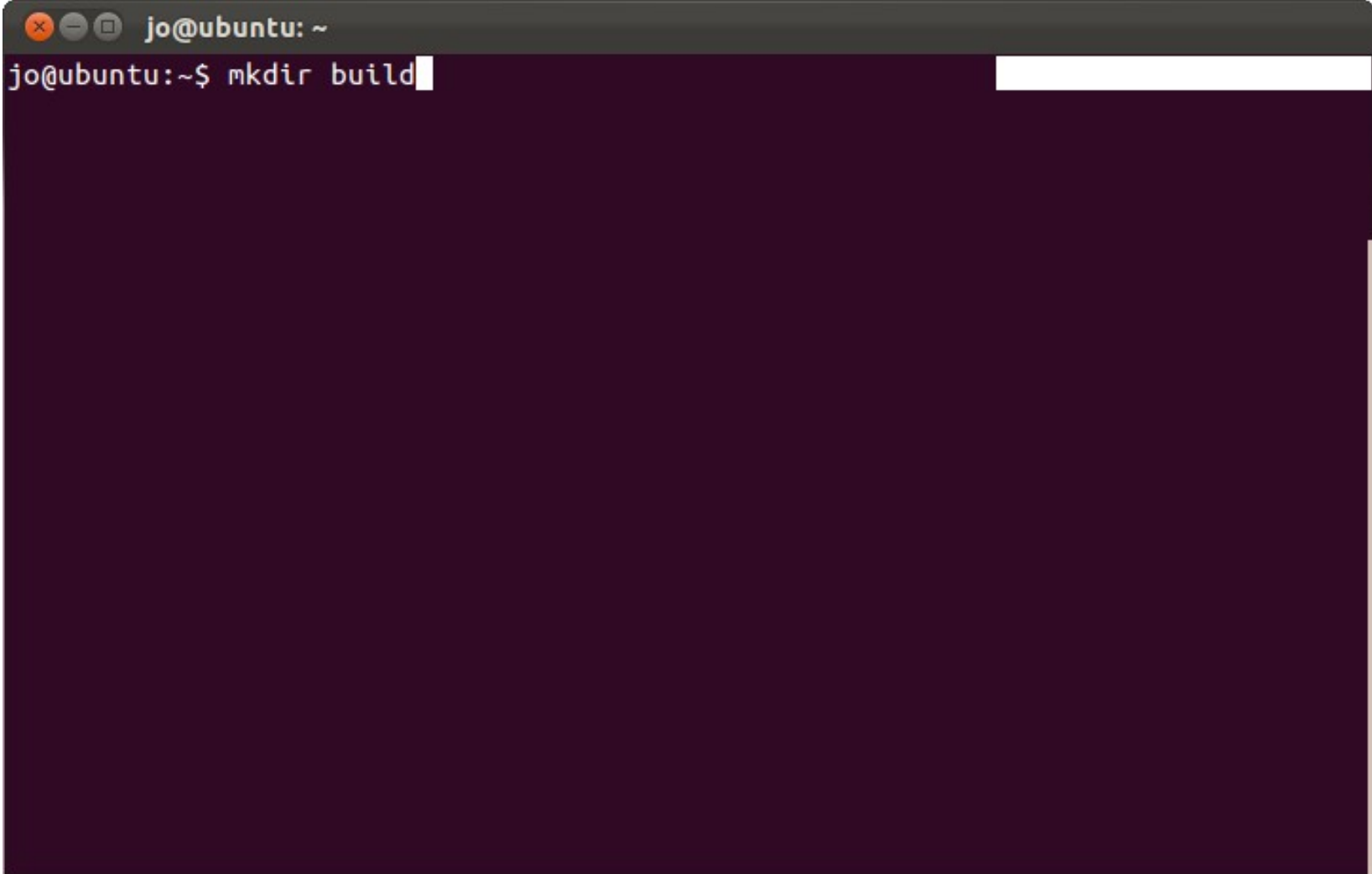
- Basisklasse aller „Objects“ in Orxonox
- Objects:
  - Können in ein Level geladen werden.
  - Werden am Ende des Levels wieder gelöscht.

# Klassenhierarchie - Baseobject

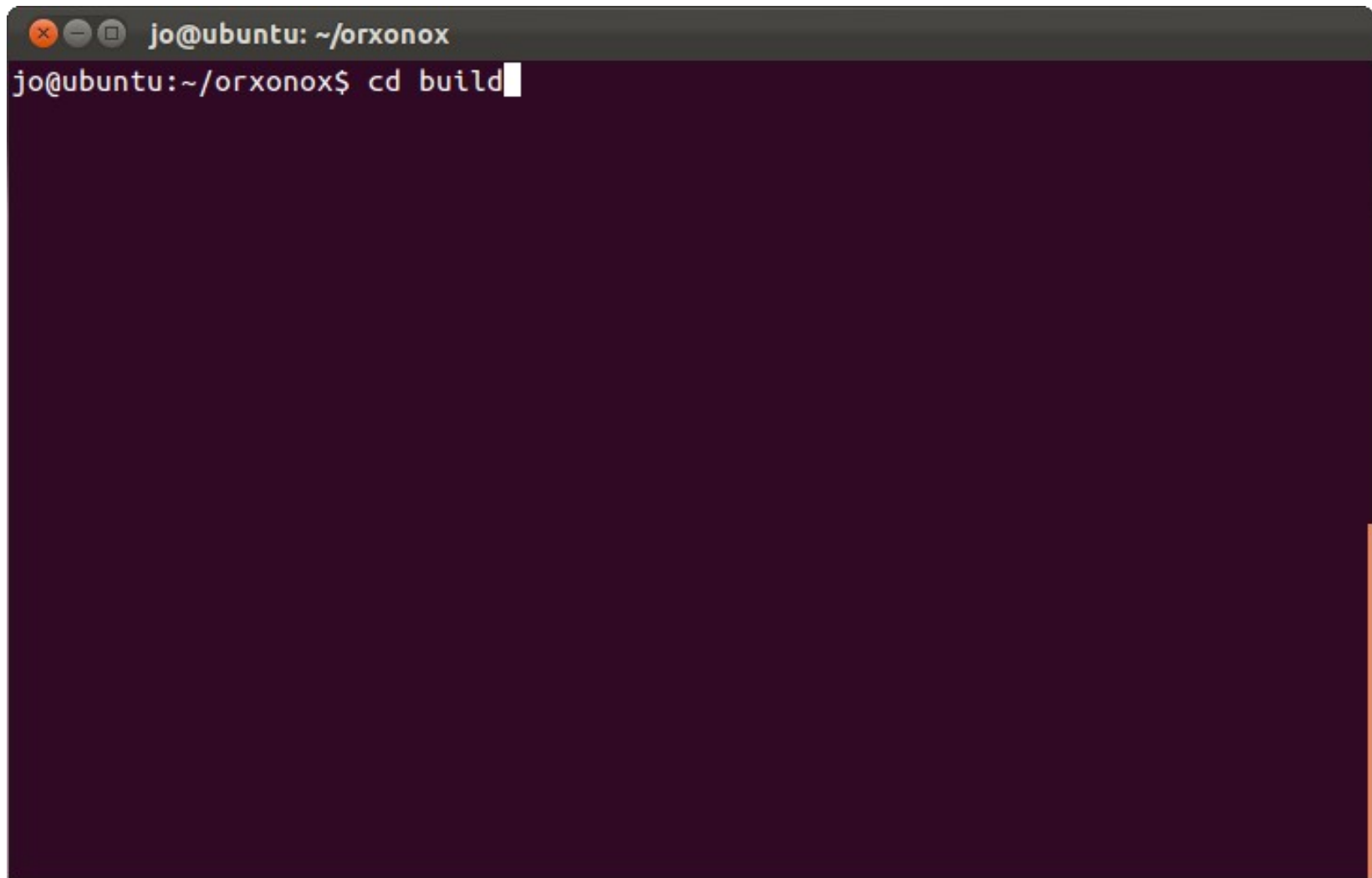
- Basisklasse aller „Objects“ in Orxonox
- Objects:
  - Können in ein Level geladen werden.
  - Werden am Ende des Levels wieder gelöscht.



# Installation – Build Ordner

A terminal window with a dark purple background and a grey title bar. The title bar contains three window control icons (close, minimize, maximize) and the text 'jo@ubuntu: ~'. The terminal shows the command 'jo@ubuntu:~\$ mkdir build' with a white cursor at the end of the line.

```
jo@ubuntu: ~  
jo@ubuntu:~$ mkdir build
```

A terminal window with a dark gray title bar and a dark purple body. The title bar contains three window control icons (close, minimize, maximize) and the text 'jo@ubuntu: ~/orxonox'. The terminal body shows a command prompt 'jo@ubuntu:~/orxonox\$' followed by the command 'cd build' and a white cursor at the end of the line.

```
jo@ubuntu: ~/orxonox
jo@ubuntu:~/orxonox$ cd build
```

# Installation - CMake

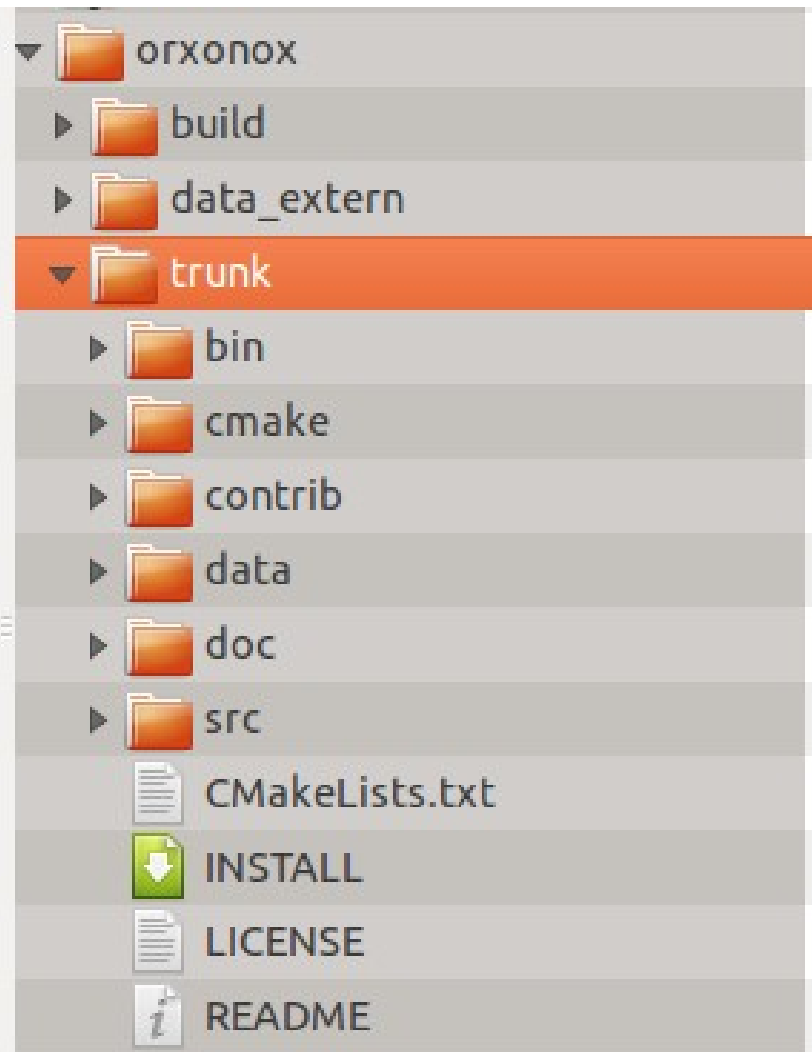
A terminal window with a dark purple background. The title bar at the top shows a red close button, a grey minimize button, and a grey maximize button, followed by the text 'jo@ubuntu: ~/orxonox/build'. The terminal content shows a command prompt 'jo@ubuntu:~/orxonox/build\$' followed by the command 'cmake -G"Eclipse CDT4 - Unix Makefiles" ../trunk' with a white cursor at the end.

```
jo@ubuntu: ~/orxonox/build
jo@ubuntu:~/orxonox/build$ cmake -G"Eclipse CDT4 - Unix Makefiles" ../trunk
```

# CMake

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

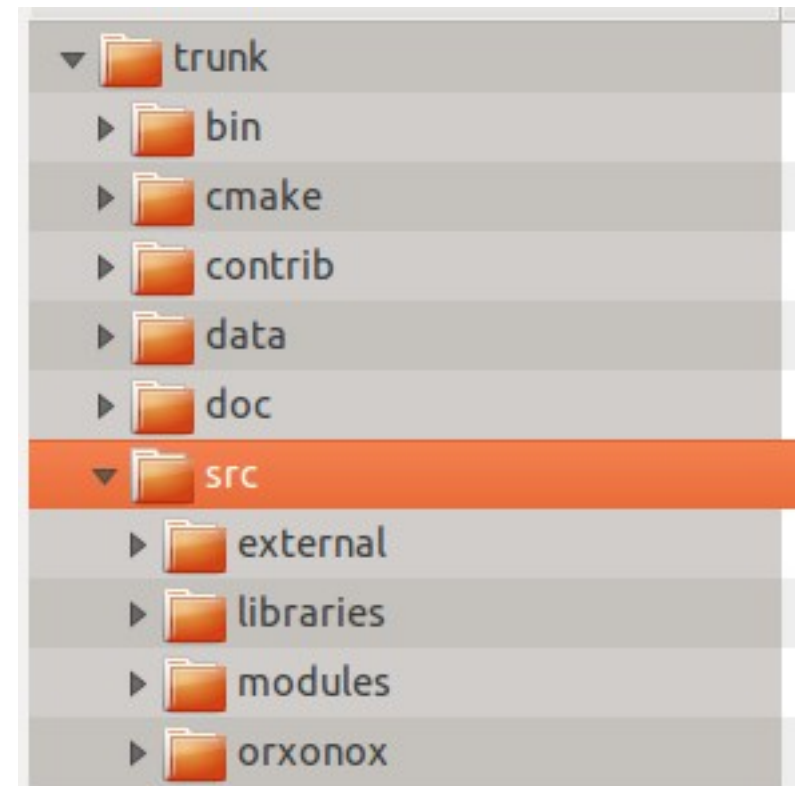
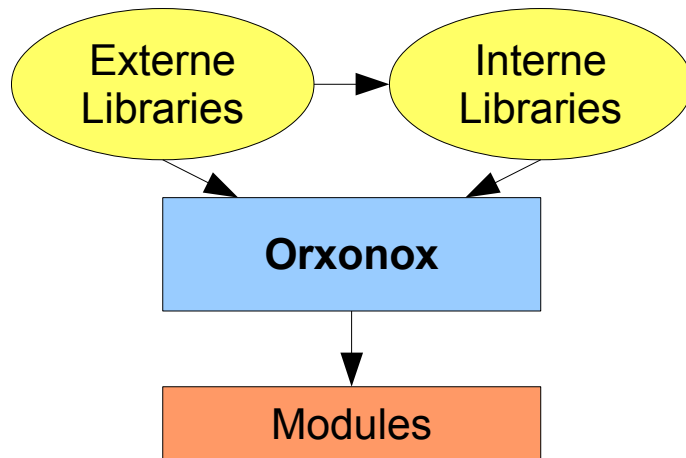
# Verzeichnisstruktur - Trunk




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

# Framework

## Struktur

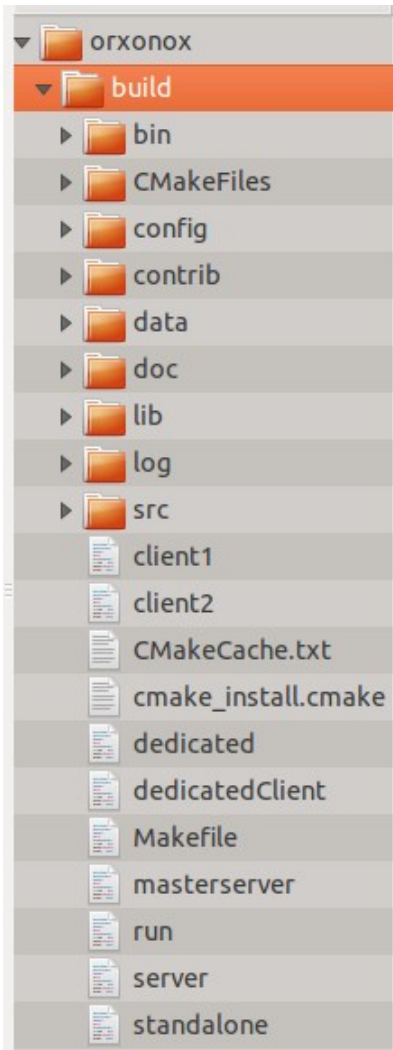


# Installation - Make

A terminal window with a dark purple background. The title bar at the top shows a red close button, a grey minimize button, and a grey maximize button, followed by the text 'jo@ubuntu: ~/orxonox/build'. The main area of the terminal is dark purple. The prompt 'jo@ubuntu:~/orxonox/build\$' is visible, followed by the command 'make -j4' and a white cursor at the end.

```
jo@ubuntu: ~/orxonox/build
jo@ubuntu:~/orxonox/build$ make -j4
```

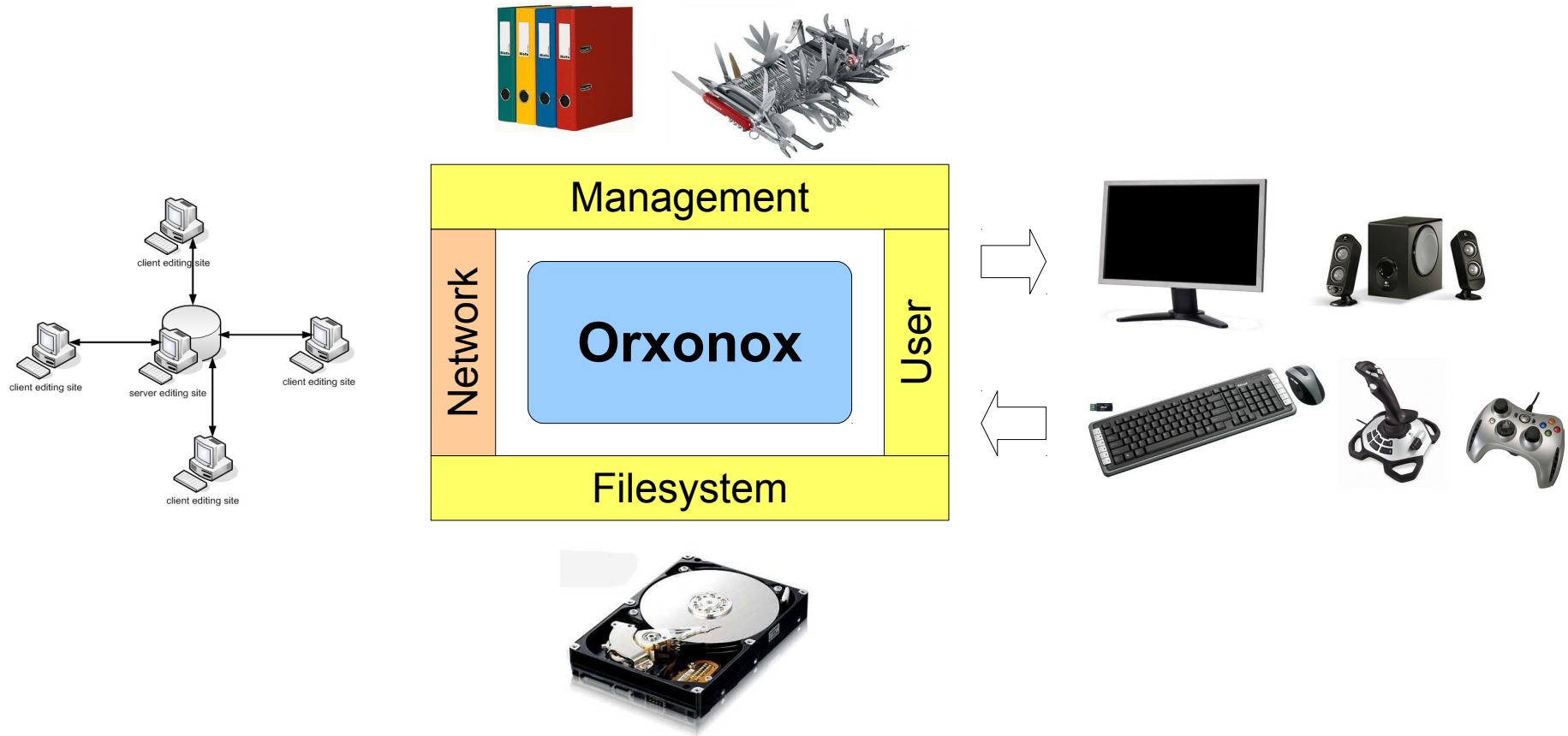
# Verzeichnisstruktur - Build



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



# Framework



# Framework

## 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.

# Framework

## 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 (ein bewegliches WorldEntity).
- Da MovableEntity ausserdem vom Interface Tickable erbt, erbt auch unsere Klasse die Tick-Funktion.

# Framework

## Beispielklasse: Source

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

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

MyClass::~~MyClass()
{
}

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

# Framework

## 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)
{
}
```

# Framework

## 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)
{
}
```

# Framework

## 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)
{
}
```

# Framework

## Beispielklasse: SUPER

Damit nicht nur die Tick-Funktion von MyClass aufgerufen wird, sondern auch weiterhin der Tick von MovableEntity, 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);
}
```



# Framework

## 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;
}
```

# Grafik & XML Demo

- Models
- Billboard
- Particle Effects

# Framework

## 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>
```

# Framework

## 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);
}
```

