# Architektur von Videospielen

*Vincent Langenfeld*, Daniel Dietsch

November 25, 2020

# Organisation

Beginnen Sie jetzt mit dem Programmieren

- z.B.: `/<projektname>.sln`
- *Jenkins* und *Sonar* sind nun online
- Arbeiten Sie an *einem* Projekt, integrieren Sie *kontinuierlich*
- *MS01*: Nächste Woche (Spielobjekt in der Welt bewegbar, bewegliche Ansichten, Level laden/speichern, Soundausgabe)
- Nehmen Sie die wöchentlichen Aufgaben als Chance sich zu verbessern

Ideenpräsentation (Do. Nächste Woche)

- Ab 14 Uhr ct. in Zoom
  - Ab 14 Uhr st. Technik testen
  - Gesamte Gruppe mit Kamera
- max 10min (+ 5min Fragen)
- Bereiten Sie sich vor:
  - Worum geht es?
  - Zentrale Spielmechanik?
  - Spielablauf? (gewinnen, verlieren)
  - Warum macht es Spaß?

# Architektur

# Architektur

*Engine*

- Eingabe
- Menüs und Popups
- Persistente Einstellungen
- Speichern/Laden
- Rendern und Kamera
- Pathfinding
- Kollisionserkennung
- Netzwerk
- Objektverwaltung

*Spielmechanik*

- Spielobjekte und Eigenschaften
- Interaktion
- KI

*Tools*

- Modelle integrieren
- Texturen integrieren
- Sounds integrieren
- Level erstellen
- Debugging output

*Content*

- Sound und Musik
- Modelle
- Sprites
- Levels

# Architektur

*Engine*

- Eingabe
- Menüs und Popups
- Persistente Einstellungen
- Speichern/Laden
- Rendern und Kamera
- Pathfinding
- Kollisionserkennung
- Netzwerk
- Objektverwaltung

*Spielmechanik*

- Spielobjekte und Eigenschaften
- Interaktion
- KI

*Tools*

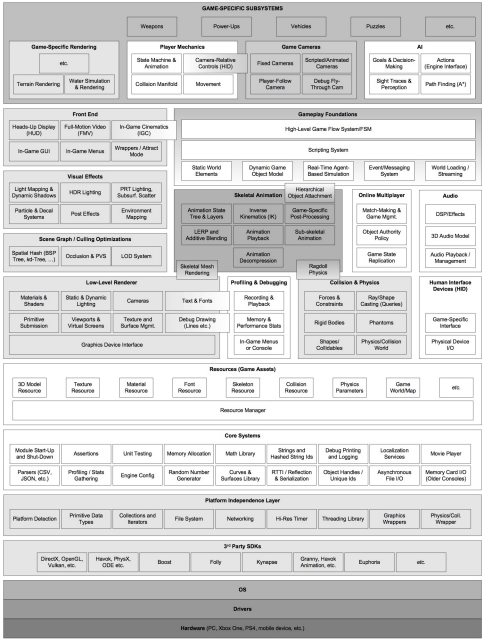
- Modelle integrieren
- Texturen integrieren
- Sounds integrieren
- Level erstellen
- Debugging output

*Content*

- Sound und Musik
- Modelle
- Sprites
- Levels

A Man Ran (WS17/18)

# Architektur

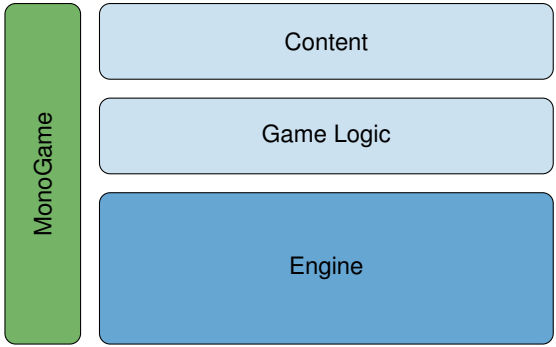



Gregory, *Game Engine Architecture*, 2019

# Architektur

Am Besten *Top-Down* anfangen:

- Zuerst Struktur aufbauen
- Funktionalität aufteilen
- Separation of Concerns



# MonoGame und .NET

# MonoGame und .NET

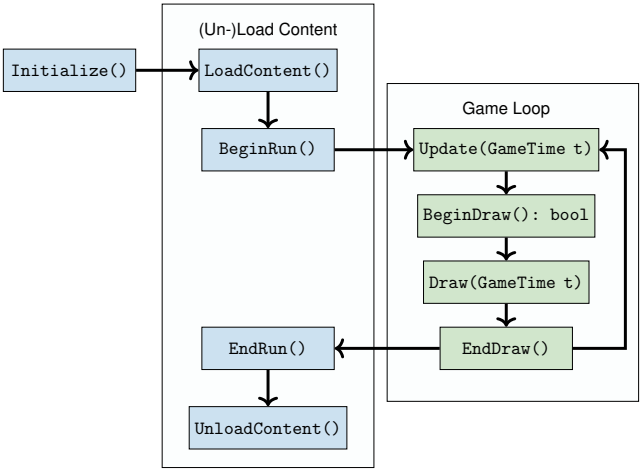
*.net*

- Mathematik
- Zufallszahlen
- (De-)Serialisierung in XML/Binär
- Datenstrukturen (Liste, Menge, ...)
- Debugging
- Profiling

*MonoGame*

- Abstraktion
  - Grafik
  - Sound
  - Input
- Content Pipeline
- Einfache Anzeigemethoden (BasicEffect, SpriteBatch)
- Datentypen
  - Vector2, Vector3
  - Matrix
  - ...
- Game Loop

# MonoGame Lifecycle



# Bausteine

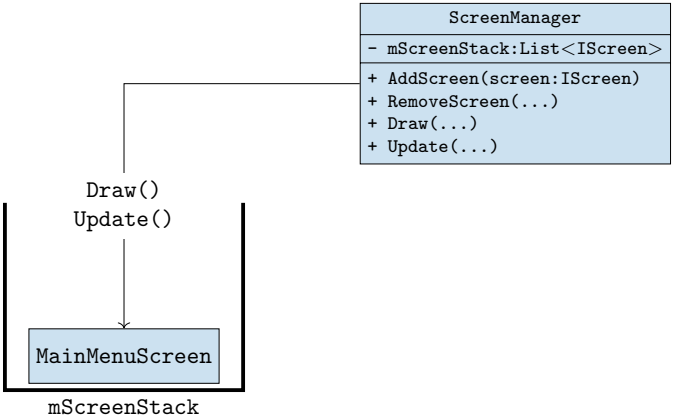
# Screen Management

- Verschiedene Ansichten des Spiels in *Screens* unterteilen
  - Hauptmenü
  - Optionsmenü
  - Spielansicht
  - HUD
  - Ladebildschirm
  - ...

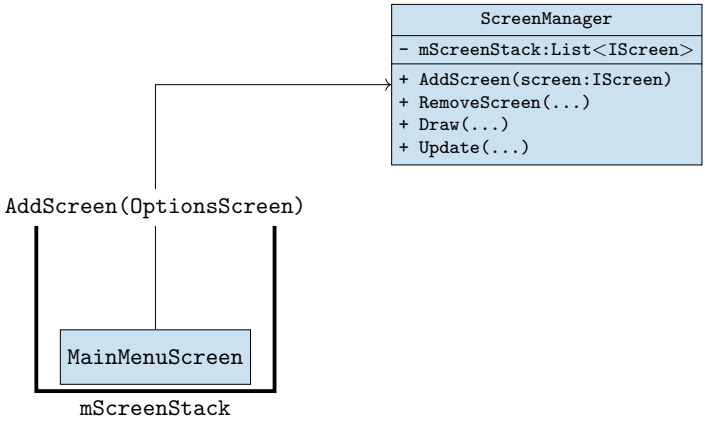
# Screen Management

- Verschiedene Ansichten des Spiels in *Screens* unterteilen
  - Hauptmenü
  - Optionsmenü
  - Spielansicht
  - HUD
  - Ladebildschirm
  - ...
- *ScreenManager* verwaltet alle Screens
- *ScreenManager* verwaltet Übergänge zwischen Screens
- Verwaltung der Screens vom Inhalt des einzelnen Screens trennen

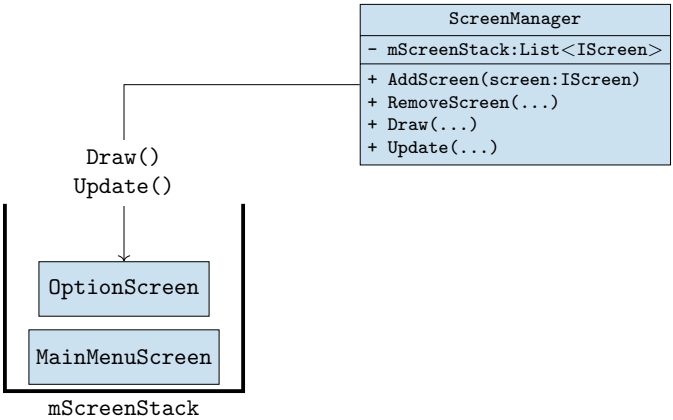
# Screen Management



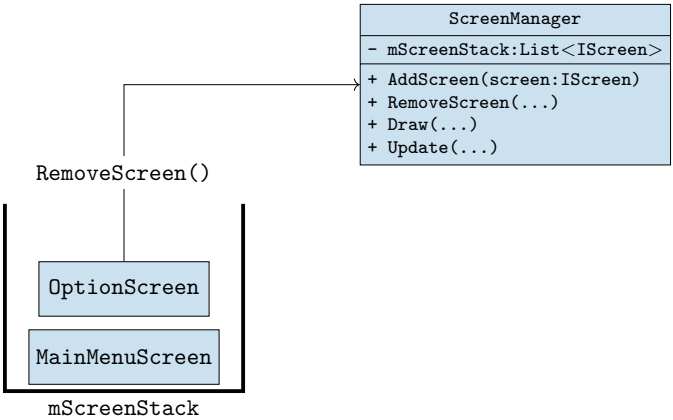
# Screen Management



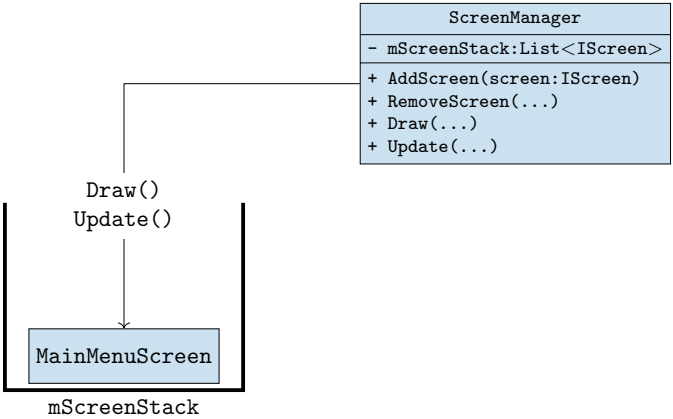
# Screen Management



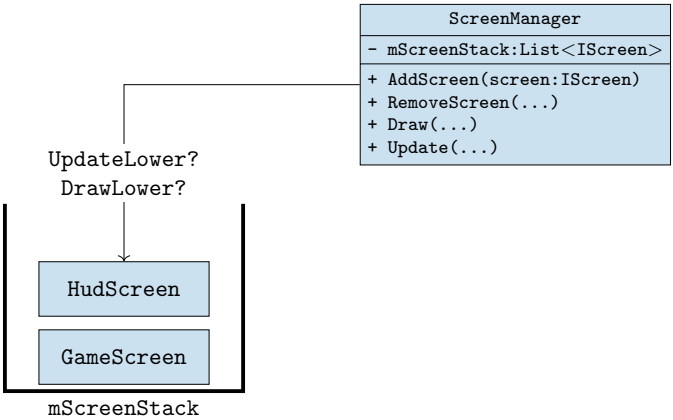
# Screen Management



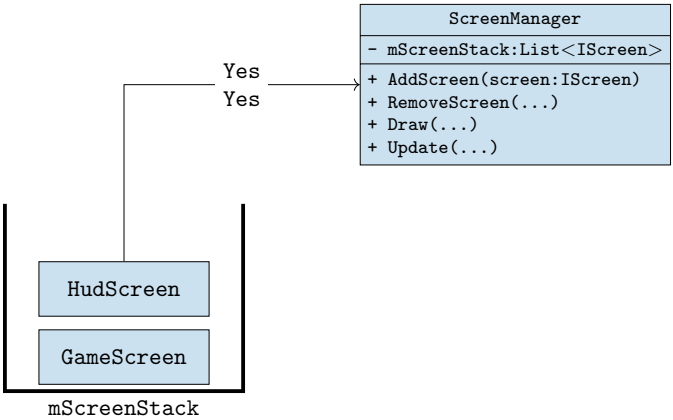
# Screen Management



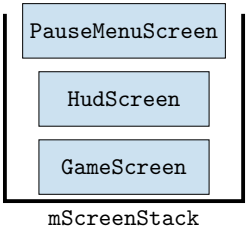
# Screen Management



# Screen Management

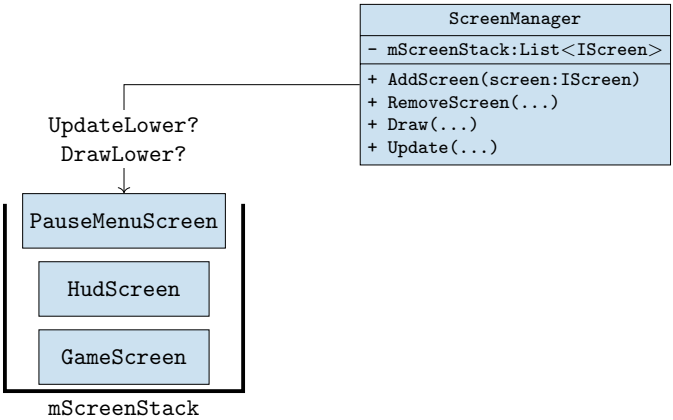


# Screen Management

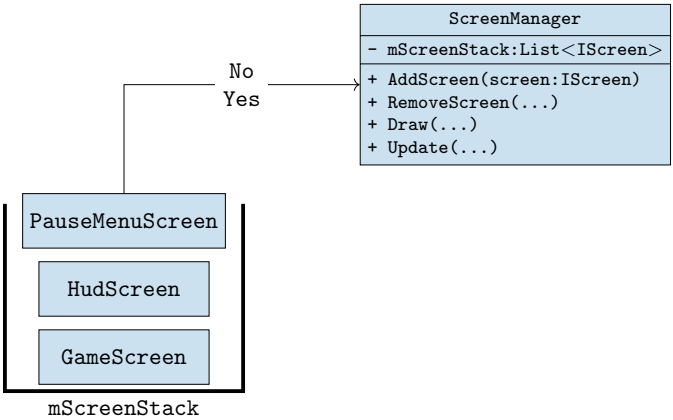



| ScreenManager |
| --- |
| - mScreenStack:List<IScreen> |
| + AddScreen(screen:IScreen)<br>+ RemoveScreen(...)<br>+ Draw(...)<br>+ Update(...) |

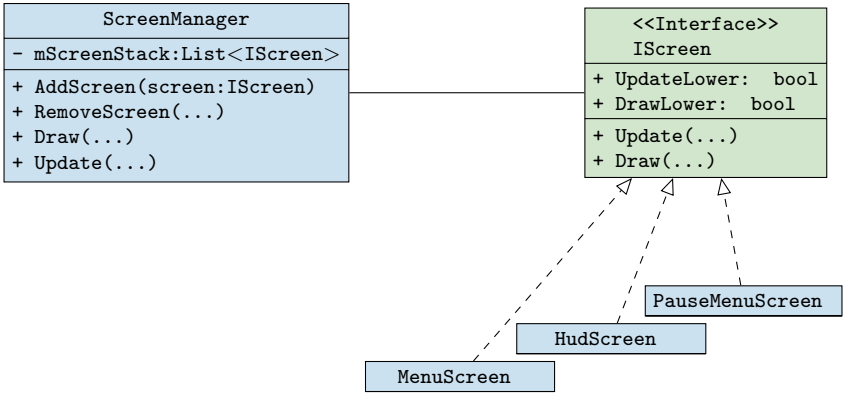
# Screen Management



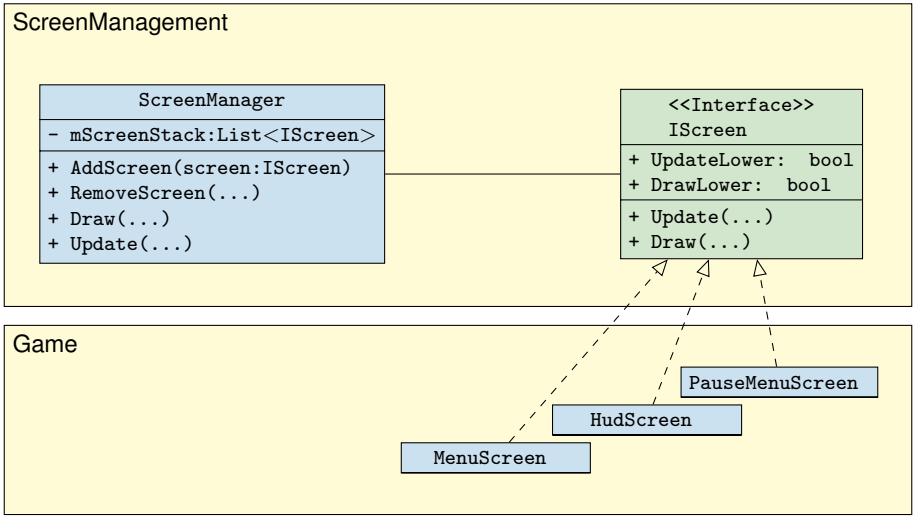
# Screen Management



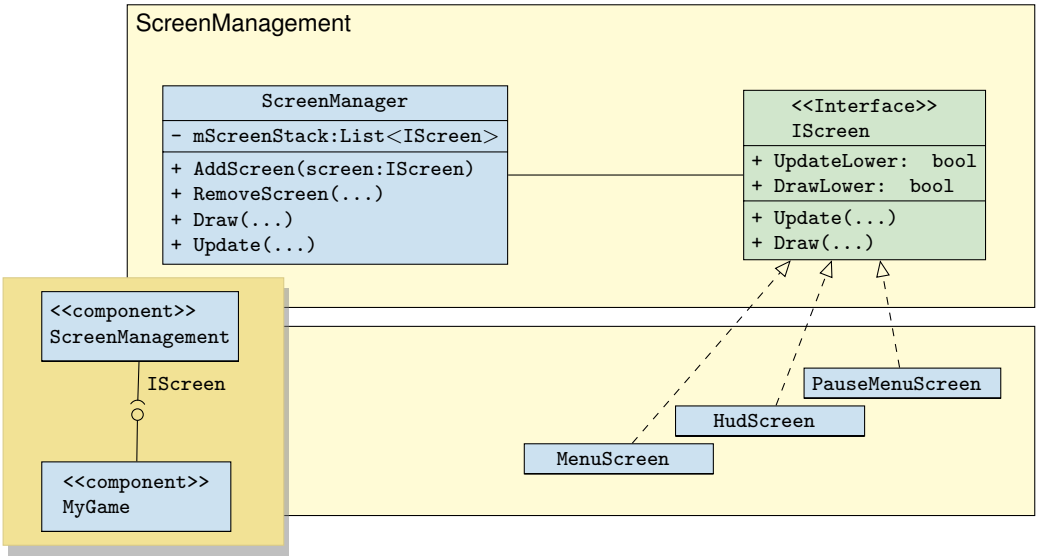
# Screen Management



# Screen Management



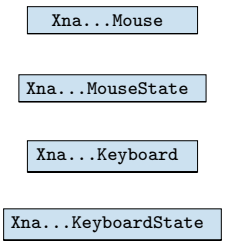
# Screen Management



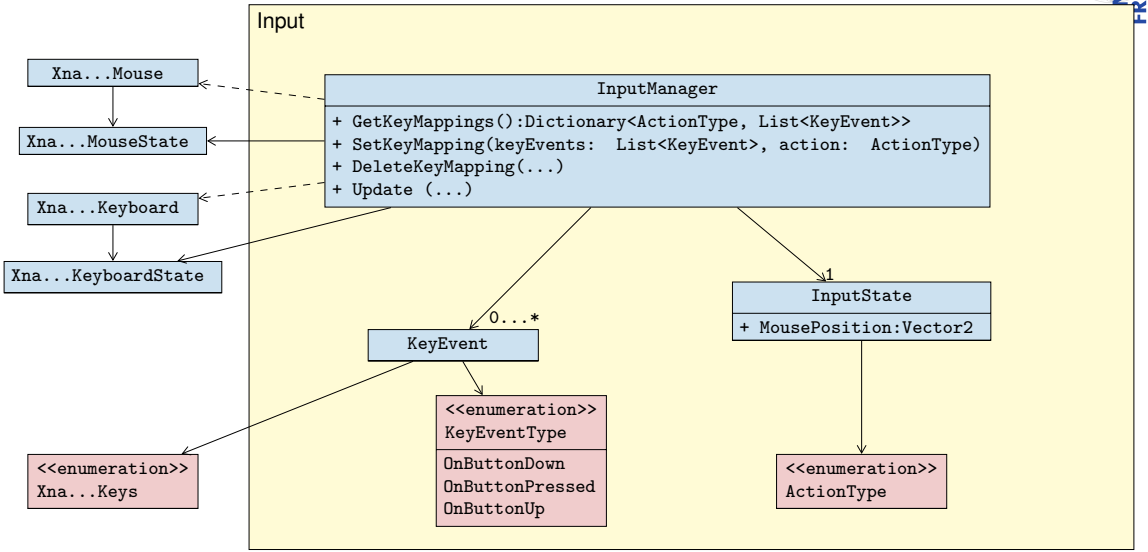
# Input Management

- Tastatur, Maus, Gamepad, ...
- MonoGame Namespace `Microsoft.Xna.Framework.Input`
  - `KeyboardState mKeyboardState = Keyboard.GetState();`
  - Statusinformation per Frame: Zustand der Tasten, Mausposition
  - *keine* Historie
- Wir brauchen
  - Statusinformation in Abhängigkeit von Zeit
    (z.B.: A wurde diese Frame losgelassen)
  - Abstraktion von Tasten zu Aktionen

# Input Management



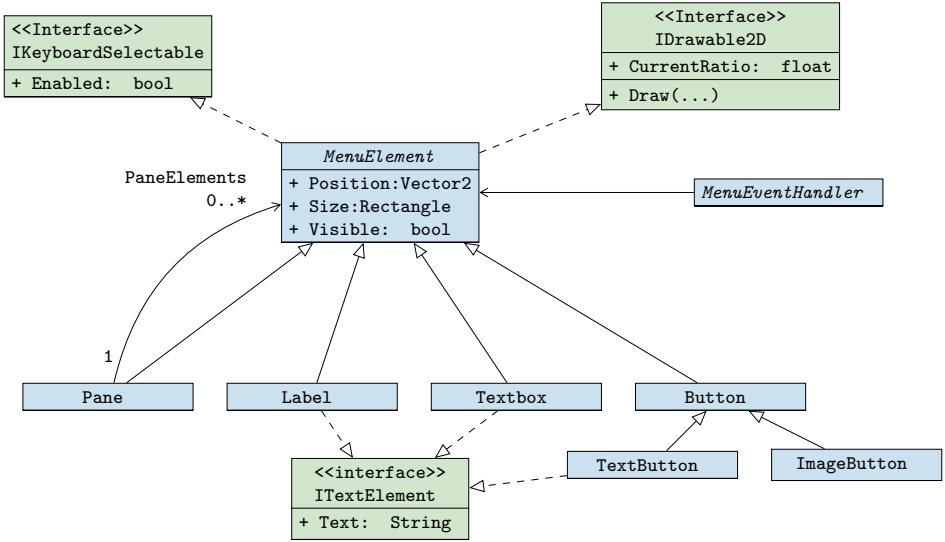
# Input Management



# Menüs

- Viele Elemente die sich sehr ähnlich verhalten
  - Panels
  - Buttons
  - Labels
  - Textboxen
  - Checkboxen
- Auch im Hud verwendbar

# Menüs


<<Interface>>
IKeyboardSelectable
+ Enabled: bool
<<Interface>>
IDrawable2D
+ CurrentRatio: float
+ Draw(...)
MenuElement
+ Position:Vector2
+ Size:Rectangle
+ Visible: bool
MenuEventHandler
PaneElements
0..*
1
Pane
Label
Textbox
Button
<<interface>>
ITextElement
+ Text: String
TextButton
ImageButton

## Spielobjekte

Spielobjekte sind alle Dinge, die eine Repräsentation in der Spielwelt haben.

- Charaktere, Fahrzeuge, Bäume, Raketen, Gras, Steine, Trigger, Lichter, Sounds, ...
- Spielobjekte müssen manchmal
  - gezeichnet werden
  - sich bewegen
  - zerstörbar sein
  - miteinander kollidieren
  - etc.
- Wie verwaltet man so viele (verschiedene) Objekte effizient?

## Spielobjekte

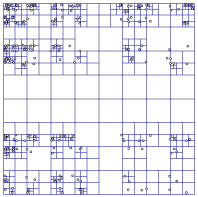
Spielobjekte sind alle Dinge, die eine Repräsentation in der Spielwelt haben.

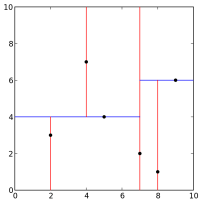
- Charaktere, Fahrzeuge, Bäume, Raketen, Gras, Steine, Trigger, Lichter, Sounds, ...
- Spielobjekte müssen manchmal
  - gezeichnet werden
  - sich bewegen
  - zerstörbar sein
  - miteinander kollidieren
  - etc.
- Wie verwaltet man so viele (verschiedene) Objekte effizient?
  - Zur *Laufzeit*?
  - Im *Softwaredesign*?

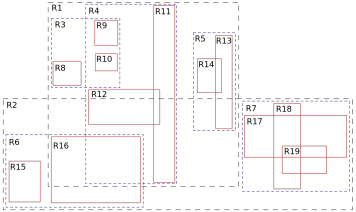
## Szenengraph

Ein Szenengraph ist eine zentrale Datenstruktur, die der logischen und/oder räumlichen Verwaltung der Spielobjekte dient.

- Antwort auf räumliche Fragen
- Update- und Drawaufrufe an Spielobjekte weitergeben
- Beispiele: Liste, Heap, Quad-/ Octree, KD-Tree, R-Tree, ...





# Spielobjekte

*Objektzentriert*

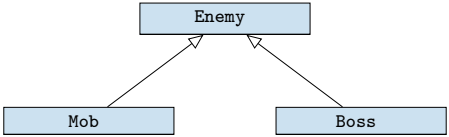
- Spielobjekte sind Klassen mit
  - Eigenschaften des Spielobjekts
  - Verhalten des Spielobjekts
- Spielwelt ist eine Menge von *Instanzen* der Spielobjekte

*Eigenschaftszentriert*

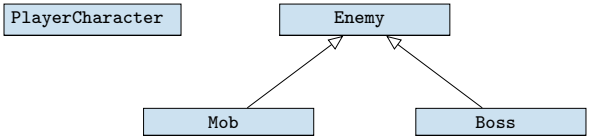
- Jedes Spielobjekt ist nur eine *ID*
- Eigenschaften der Spielobjekte werden in Tabellen gespeichert
- Verhalten von Spielobjekten sind Operationen auf Tabellen
- Ähnlich zu Datenbanken

# Spielobjekte und Vererbung

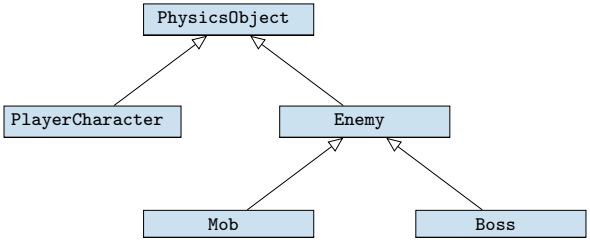
# Spielobjekte und Vererbung



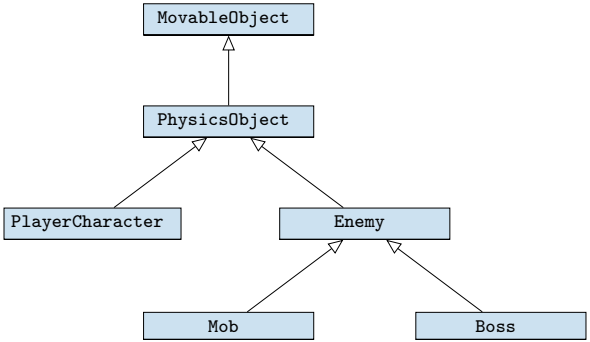
# Spielobjekte und Vererbung



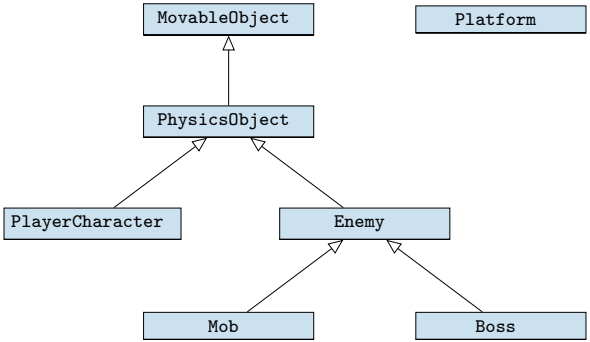
# Spielobjekte und Vererbung



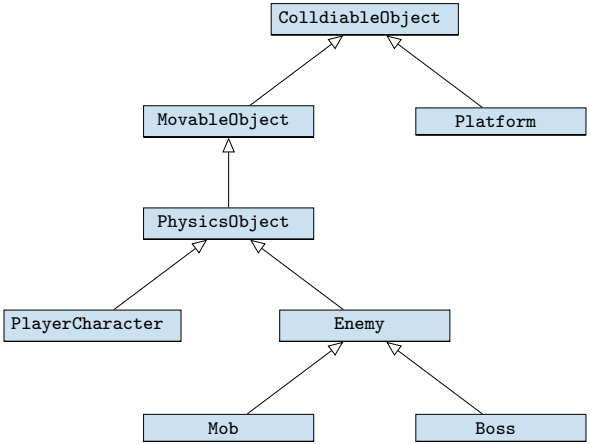
# Spielobjekte und Vererbung



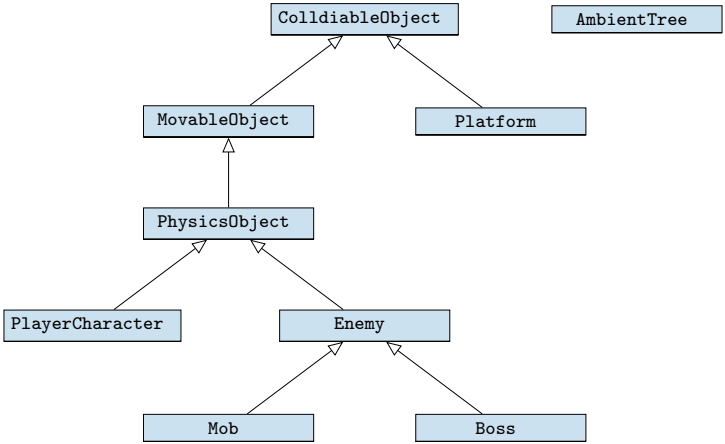
# Spielobjekte und Vererbung



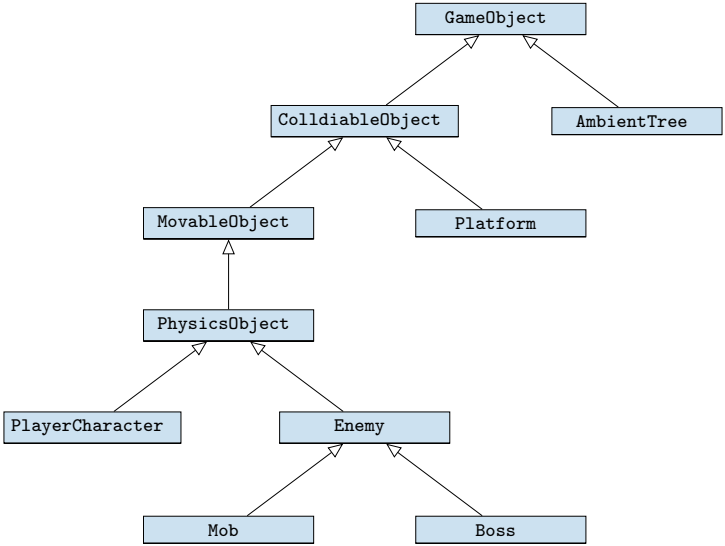
# Spielobjekte und Vererbung



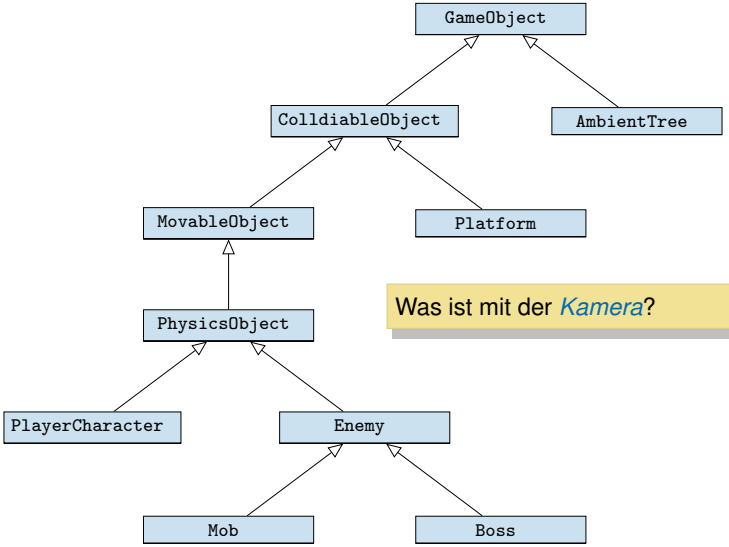
# Spielobjekte und Vererbung



# Spielobjekte und Vererbung



# Spielobjekte und Vererbung

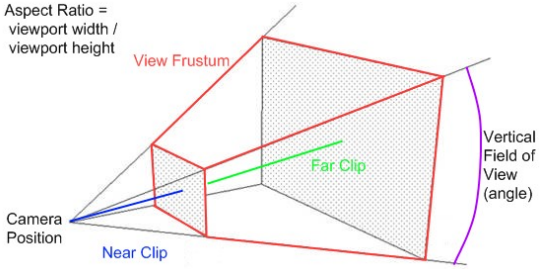



Was ist mit der *Kamera*?

# Spielobjekte

*Eine einfache Kamera*

- Speichert z.B.:
  - View Matrix
  - Projection Matrix
- Grundlage für Picking
- Definiert *View Frustrum*

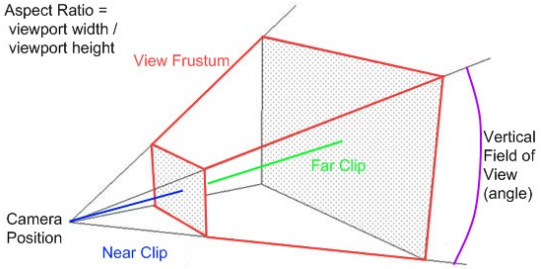


# Spielobjekte

*Eine einfache Kamera*

- Speichert z.B.:
  - View Matrix
  - Projection Matrix
- Grundlage für Picking
- Definiert *View Frustrum*

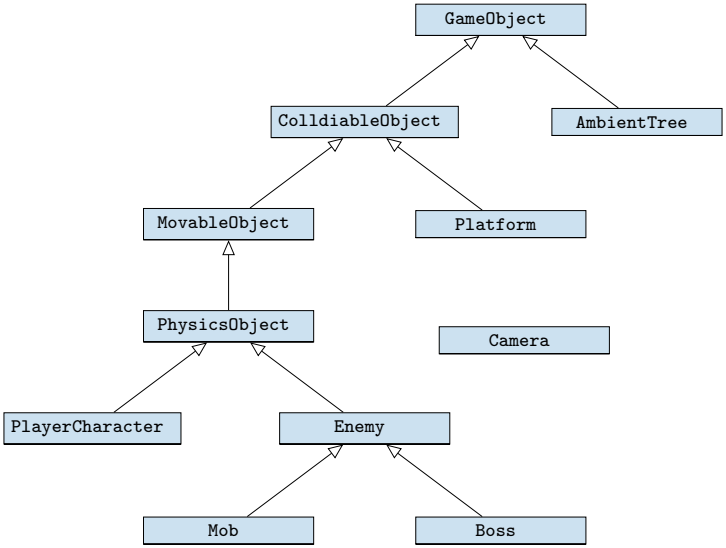
- Teil der Spielwelt
- Nicht *kollidierend*
- Beweglich



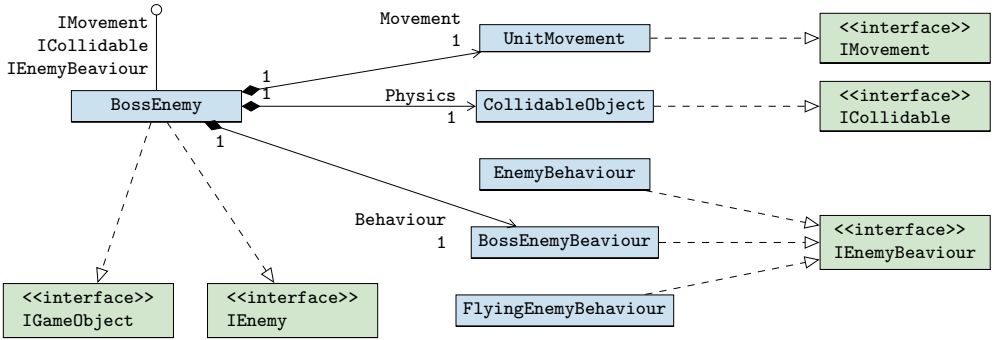
# Spielobjekte

| | Tree | Platform | PlayerCharacter | Mob | BossMob | Camera |
|---|---|---|---|---|---|---|
| GameObject | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Collidable | | ✓ | ✓ | ✓ | ✓ | |
| Movable | | | ✓ | ✓ | ✓ | ✓ |
| Physics | | | ✓ | ✓ | ✓ | |
| EnemyBehaviour | | | | ✓ | ✓ | |

# Spielobjekte und Vererbung



# Spielobjekte und Komposition



# Pathfinding

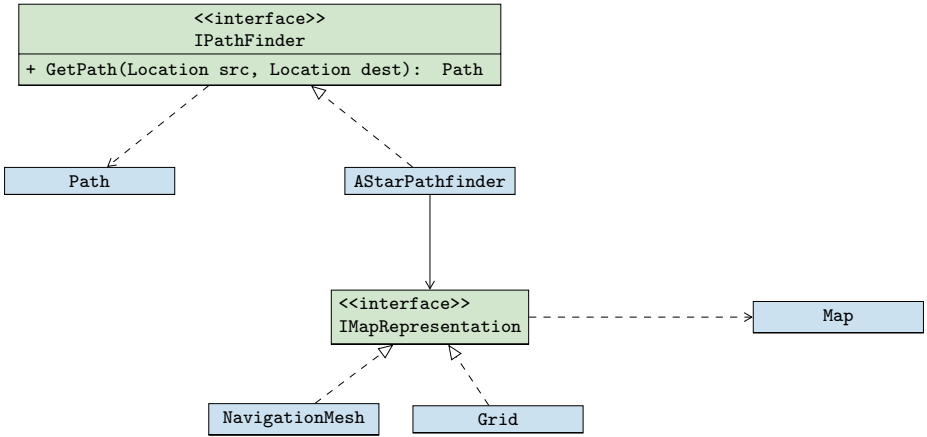
*Offline*

- Berücksichtigt Welt und unbewegliche Objekte
- Meistens *A**
- Viele mögliche Weltrepräsentationen:
  Grid, Hierarchical Grid, Waypoint Graph,
  Navigation Mesh,...

*Online*

- Beweglichen Objekten ausweichen
- Verschiedene Verfahren:
  Steering, Flocking, Flow Fields, ...

# Pathfinding



# Library Falle

# Die Libraryfalle

Die Libraryfalle

- Es existieren viele *Libraries*
- *Pro*:
  - Rad nicht neu erfinden
  - Optimierte Lösungen für bekannte Probleme
- Aber:
  - Problem muss verstanden sein
  - Library muss verstanden werden
  - *Versteckte Kosten* wenn Library fehlerhaft/schlecht/unpassend/...

# Die Libraryfalle

Die Libraryfalle

- Es existieren viele *Libraries*
- *Pro*:
  - Rad nicht neu erfinden
  - Optimierte Lösungen für bekannte Probleme
- Aber:
  - Problem muss verstanden sein
  - Library muss verstanden werden
  - *Versteckte Kosten* wenn Library fehlerhaft/schlecht/unpassend/...
- *Vorher* das Risiko abschätzen:
  - Wie zentral ist die Library?
  - Wie reif ist die Library?
  - Können wir das einfacher selbst machen?

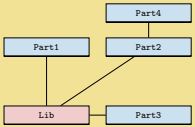
# Die Libraryfalle

Die Libraryfalle

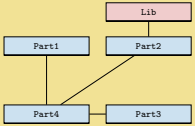
- Es existieren viele *Libraries*
- *Pro*:
  - Rad nicht neu erfinden
  - Optimierte Lösungen für bekannte Probleme
- Aber:
  - Problem muss verstanden sein
  - Library muss verstanden werden
  - *Versteckte Kosten* wenn Library fehlerhaft/schlecht/unpassend/...
- *Vorher* das Risiko abschätzen:
  - Wie zentral ist die Library?
  - Wie reif ist die Library?
  - Können wir das einfacher selbst machen?

*Wie zentral ist die Library?*

-Viel möglicher Schaden:




- Weniger möglicher Schaden:



# Organisation

# Architekturbesprechung

## Ziel

Designprinzipien auf die Architektur anwenden um diese zu verbessern, und eventuelle Probleme mit der Architektur frühzeitig aufdecken.

Ablauf:

- Kurze Zusammenfassung des Spiels
- Übersicht über die Architektur
  - Was machen die einzelnen Komponenten
- Vorstellen der Architektur anhand der Szenarien
- Fragen und zusätzliche Szenarien

```
Quelle: Dozent
Auslöser: Spieler
Umgebung:  Ich drücke im Hauptmenü auf
Spiel Starten, und befinde mich danach
im Spiel.

Es wird eine Karte erzeugt.
```

# Organisation

- Fangen Sie jetzt an zu programmieren
  - In *einem* Projekt
  - Arbeiten Sie gemeinsam
- GDD-Reviews lesen und Probleme beheben
  - Inhaltliche Probleme früh lösen
- Nicht vergessen:
  - Architekturbesprechung
  - Ideenpräsentation

# Organisation

- Fangen Sie jetzt an zu programmieren
  - In *einem* Projekt
  - Arbeiten Sie gemeinsam
- GDD-Reviews lesen und Probleme beheben
  - Inhaltliche Probleme früh lösen
- Nicht vergessen:
  - Architekturbesprechung
  - Ideenpräsentation

Für Fragen gibt es die *Sprechstunde*: Donnerstags 14-18 Uhr, am besten per Mattermost ansprechen