Architektur von Videospielen

ORGANISATORISCHES & DISCLAIMER

Organisatorisches

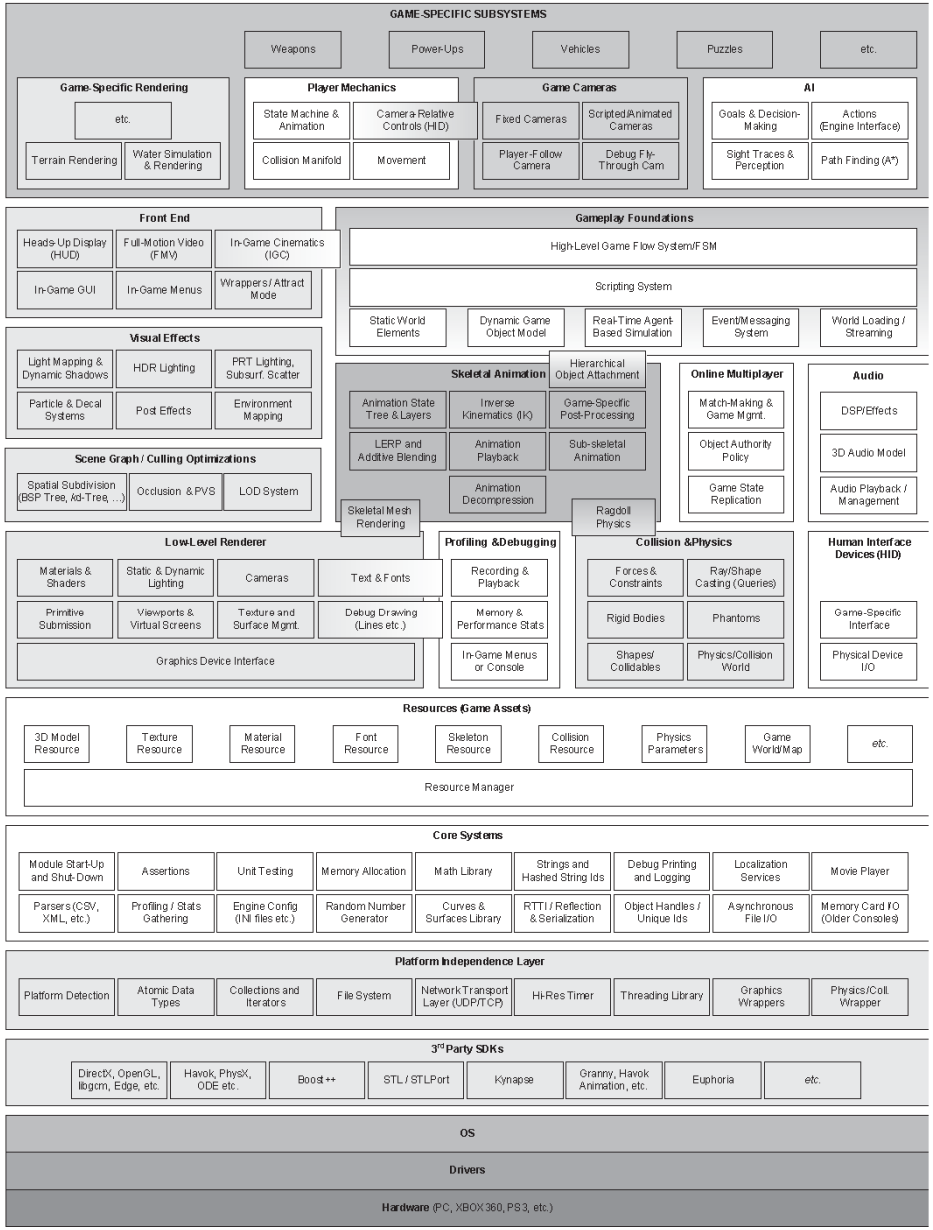
- Beginnen Sie mit dem Programmieren im Team.
  - Diese Woche:
    - GDD Beta bis Samstag 23:59 Uhr.
    - Projekt im SVN anlegen.
    - Wiederkehrende Aufgaben: Product Owner und Qualitätssicherung.
  - Übernächste Woche:
    - MS01 (Spielobjekt in der Welt bewegbar, interaktive Kamera, Karte laden/speichern, Soundausgabe).
- Ebenfalls Übernächste Woche:
  - Abgabe Architektur Beta.
  - Zwischenevaluation 1.
  - Präsentation 1 (Worum geht es im Spiel? Warum macht das Spiel Spaß? Was ist das Alleinstellungsmerkmal? Gewinnen/Verlieren?).

Disclaimer

- Wir bieten Puzzleteile, um am Anfang schnell sinnvoll Gliedern und Aufgaben verteilen zu können.
- Architektur ist sehr von der konkreten Spielidee abhängig.
- Es ist möglich, dass bei Ihrem Spiel die Architektur ganz anders aussieht als wir es hier vorstellen.
- Wir geben nur Tipps und Hinweise. Es gibt sehr viele andere Lösungsmöglichkeiten.
  - Tipp Nr. 1: Haben Sie keine Angst davor, Entscheidungen nachträglich rückgängig zu machen oder zu ändern.

Viele Fragen

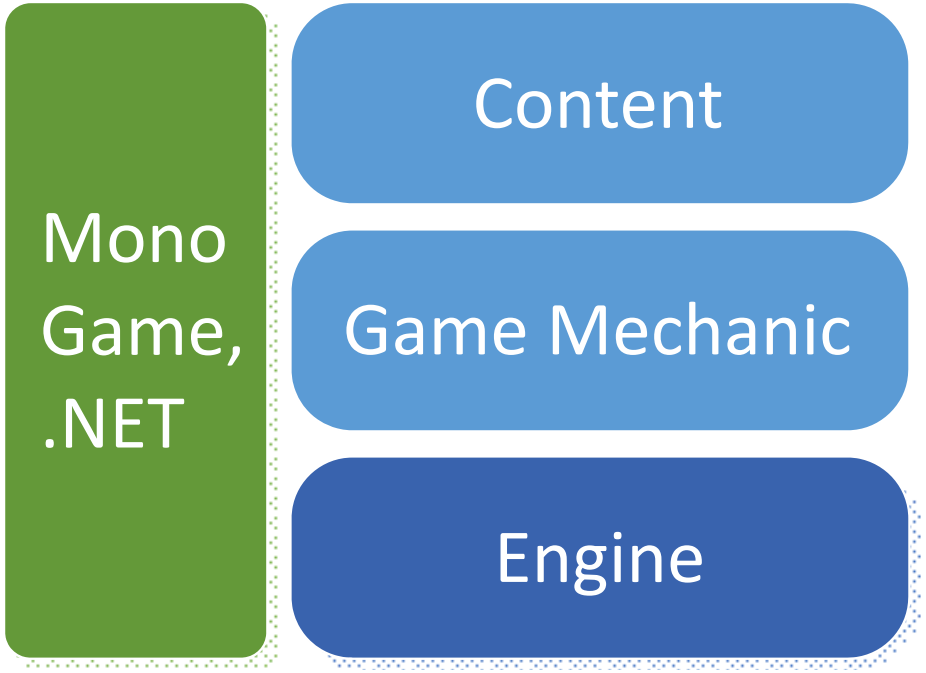
- Um was muss ich mich alles kümmern?
  - Während das Spiel läuft (Runtime):
    Eingabe/Ausgabe, Menüs, Einstellungen, Speichern/Laden, Rendern, Kamera, Eigenschaften von Spielobjekten, Interaktion zwischen Spielobjekten, Sound, Musik, KI, Netzwerk, Pathfinding, Kollisionserkennung, Debug-Helfer, Effekte,…
  - Während der Entwicklung (Tools):
    - Wie bekomme ich Assets (Modelle, Texturen, Sound, etc.) ins Spiel?
    - Wie erstelle ich Assets (z.B. Level)?
    - Debug-Helfer.
- Was wird mir bereits zur Verfügung gestellt?
- Wie verbinde ich alles?



[Gregory, 2009, 29]

Wo fangen wir an?

- Top-Down: Zuerst Strukturen, die Dinge aufteilen.
- Ganz grobe Unterteilung:



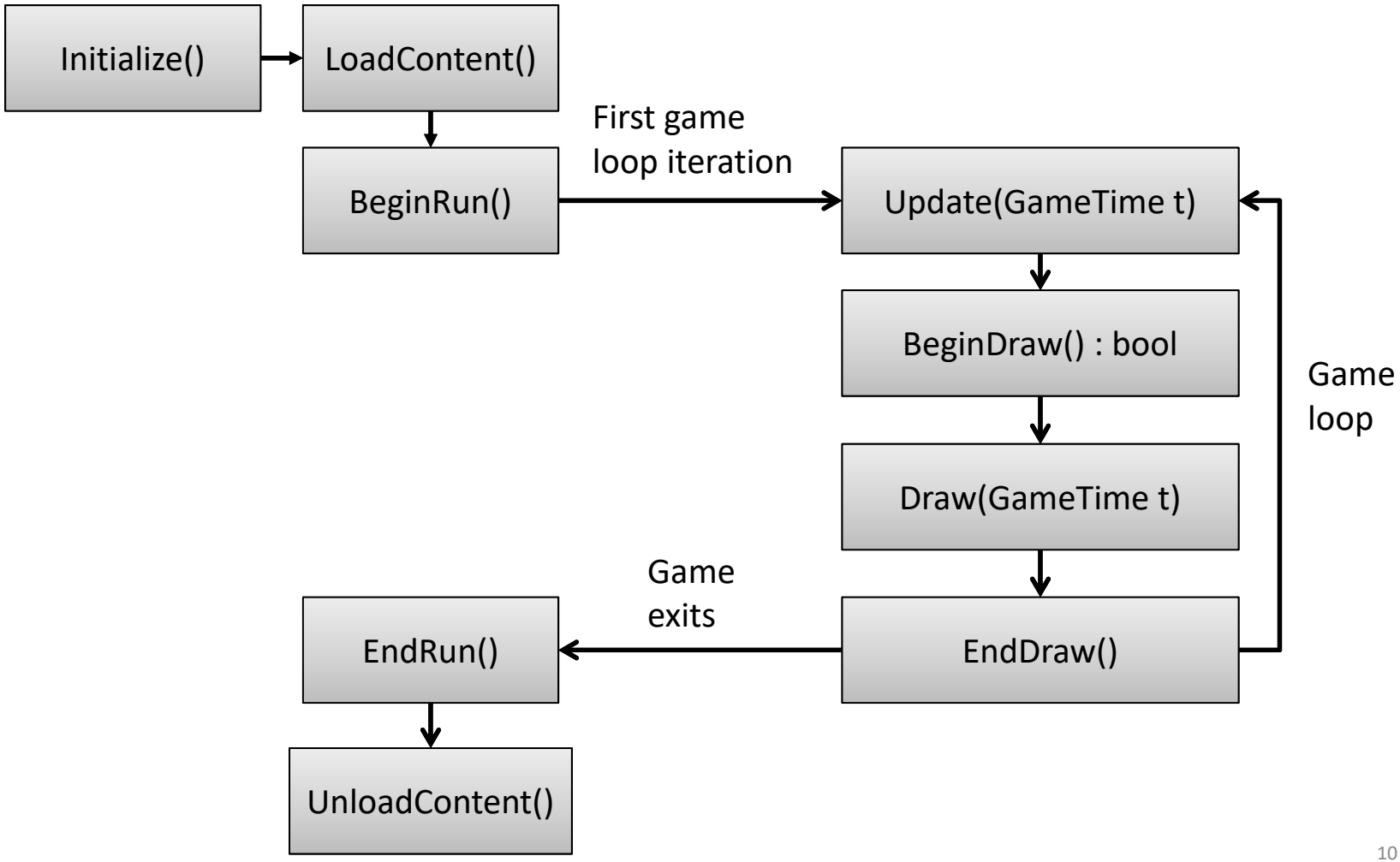
MONOGAME UND .NET

MonoGame und .NET

- MonoGame übernimmt bereits viele Aufgaben:
  - Abstraktion von Grafik, Sound und Input.
  - Aufbereiten von Assets (Content Pipeline).
  - Einfache Anzeigemethoden (BasicEffect, SpriteBatch).
  - Datentypen (Vector2D, Vector3D, Matrix, Rectangle, …).

- .NET bietet z.B.:
  - Mathematik und Zufallszahlen.
  - Serialisierung / Deserialisierung von XML und binären Daten.
  - Diverse Datenstrukturen (Listen, Mengen, …).
  - Debugging (vor allem im Zusammenspiel mit VS2015).

MonoGame Game Lifecycle

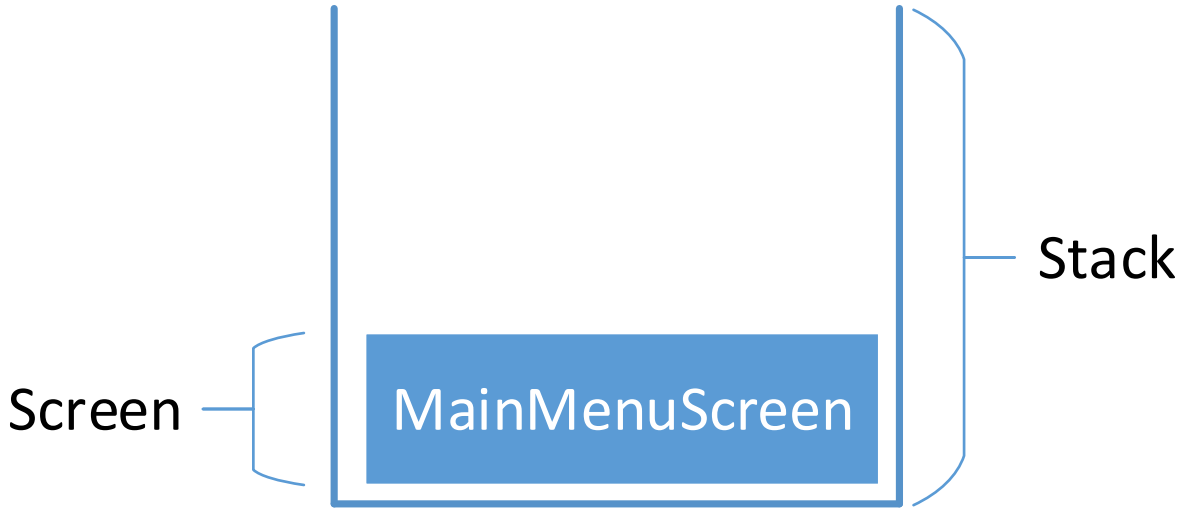


PUZZLETEIL I: SCREENS UND INPUTS

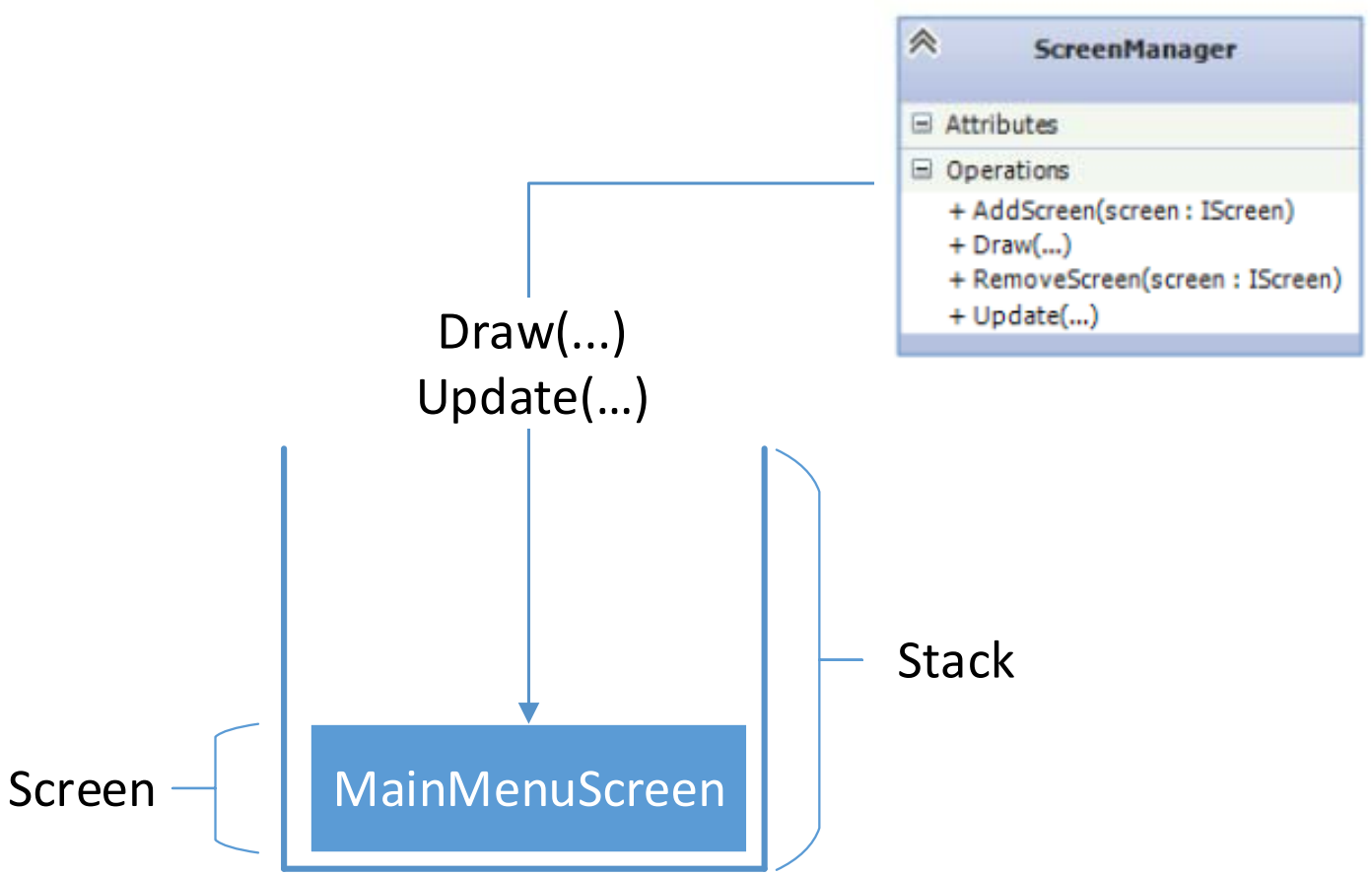
Screen Management

- Verschiedene Ansichten des Spiels in Screens unterteilen
  - Hauptmenü
  - Optionsmenü
  - Spielansicht
  - HUD
  - Ladescreen
  - …
- Alle aktiven Screens werden durch einen ScreenManager verwaltet.
- Übergänge zwischen Screens durch Methoden des ScreenManagers.

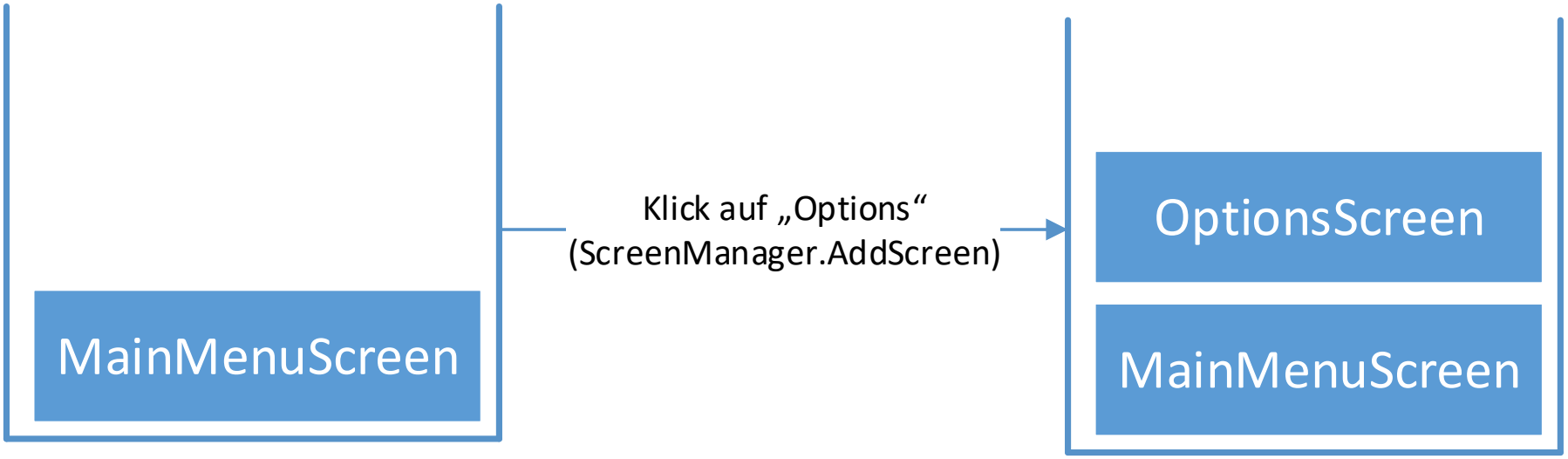
ScreenManager



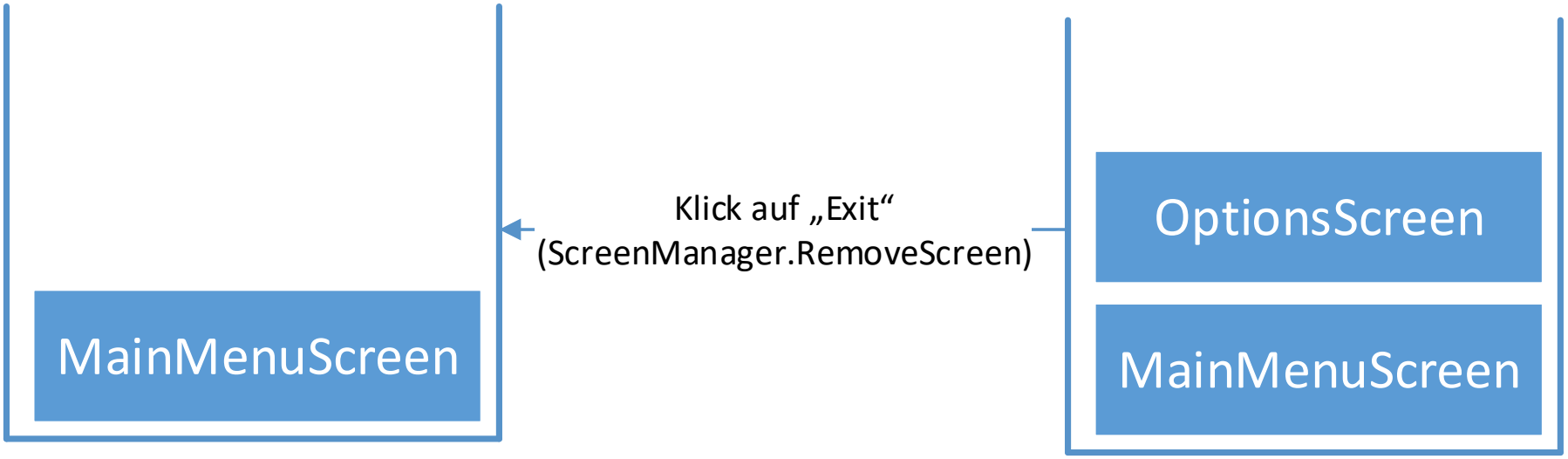
ScreenManager



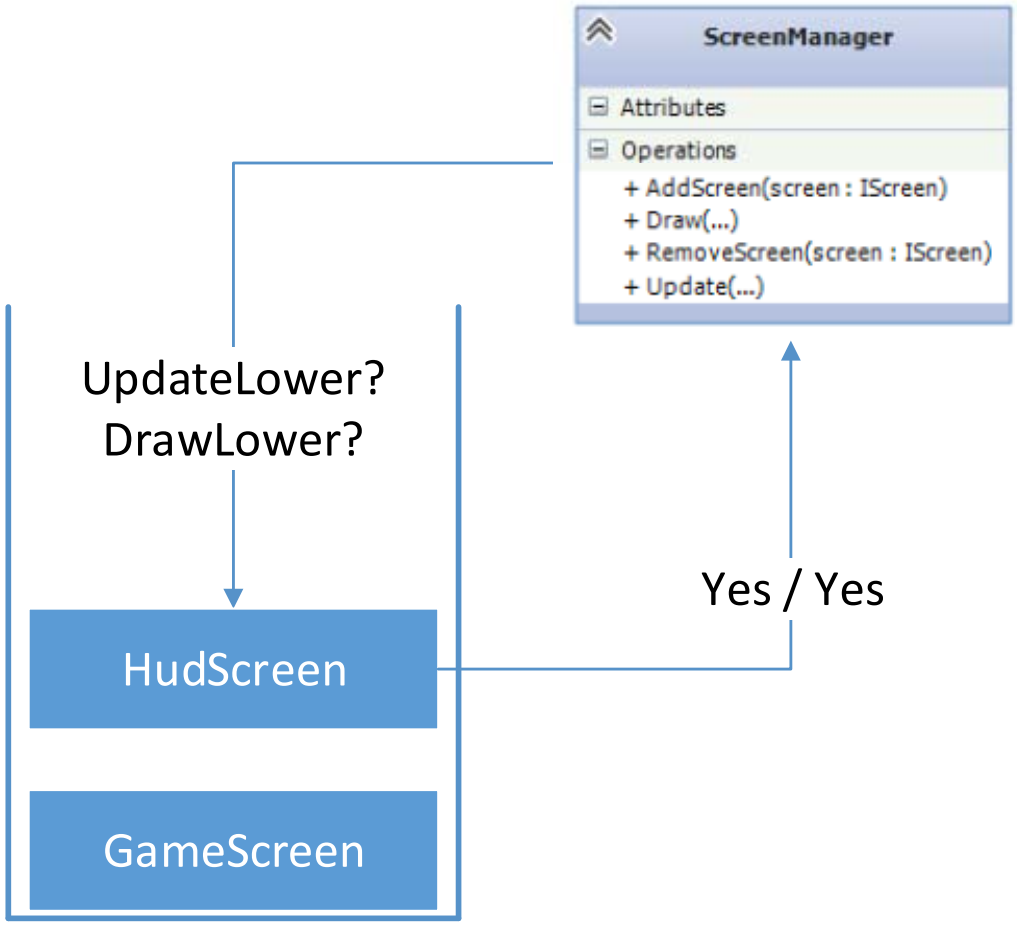
ScreenManager



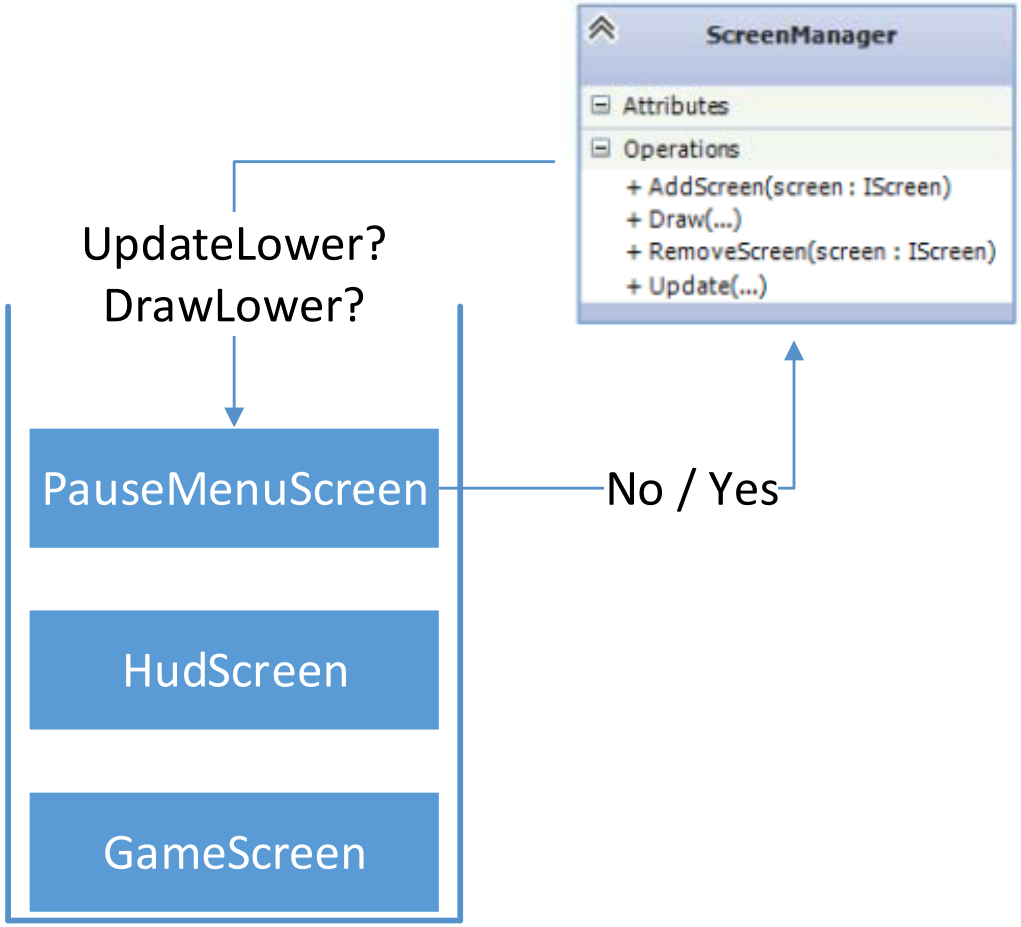
ScreenManager



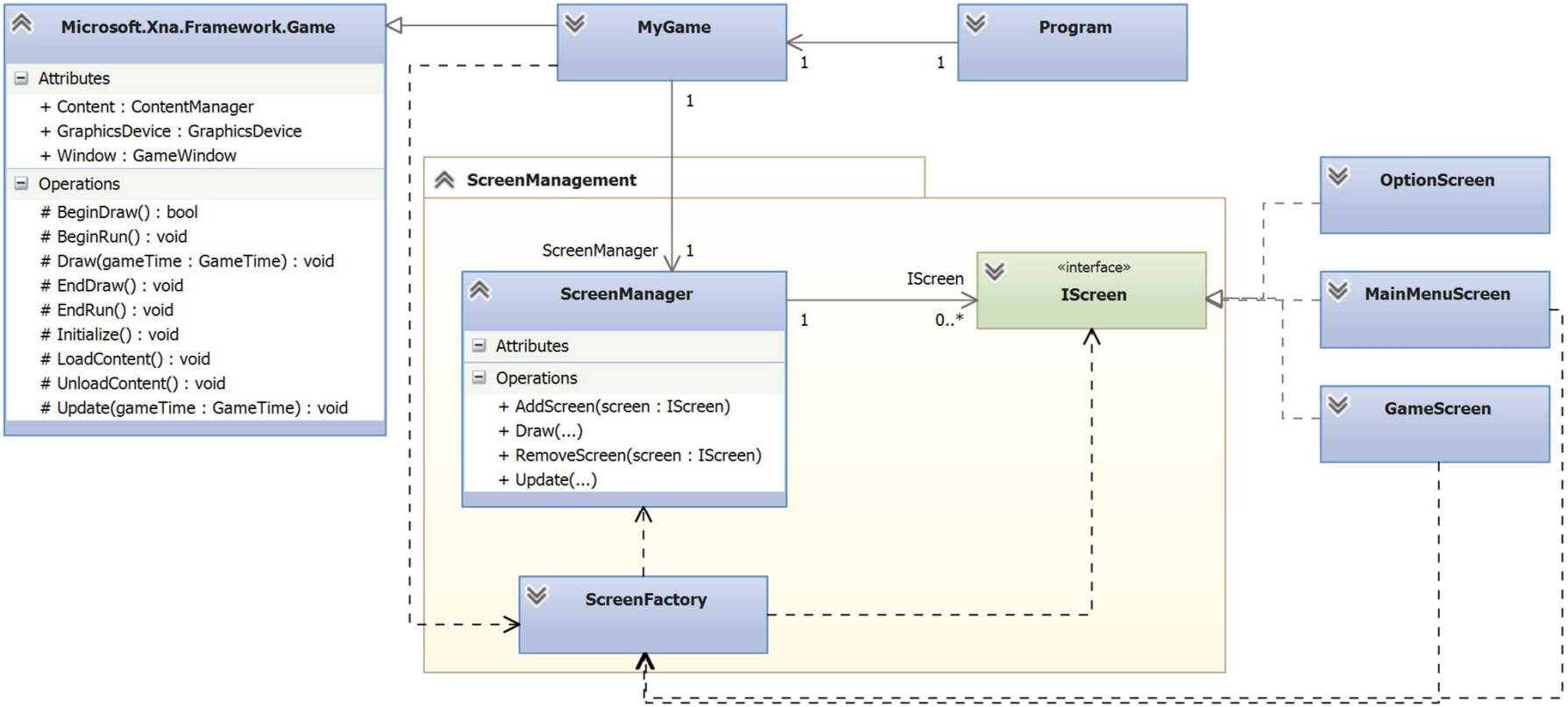
ScreenManager



ScreenManager



ScreenManagement



Inputs

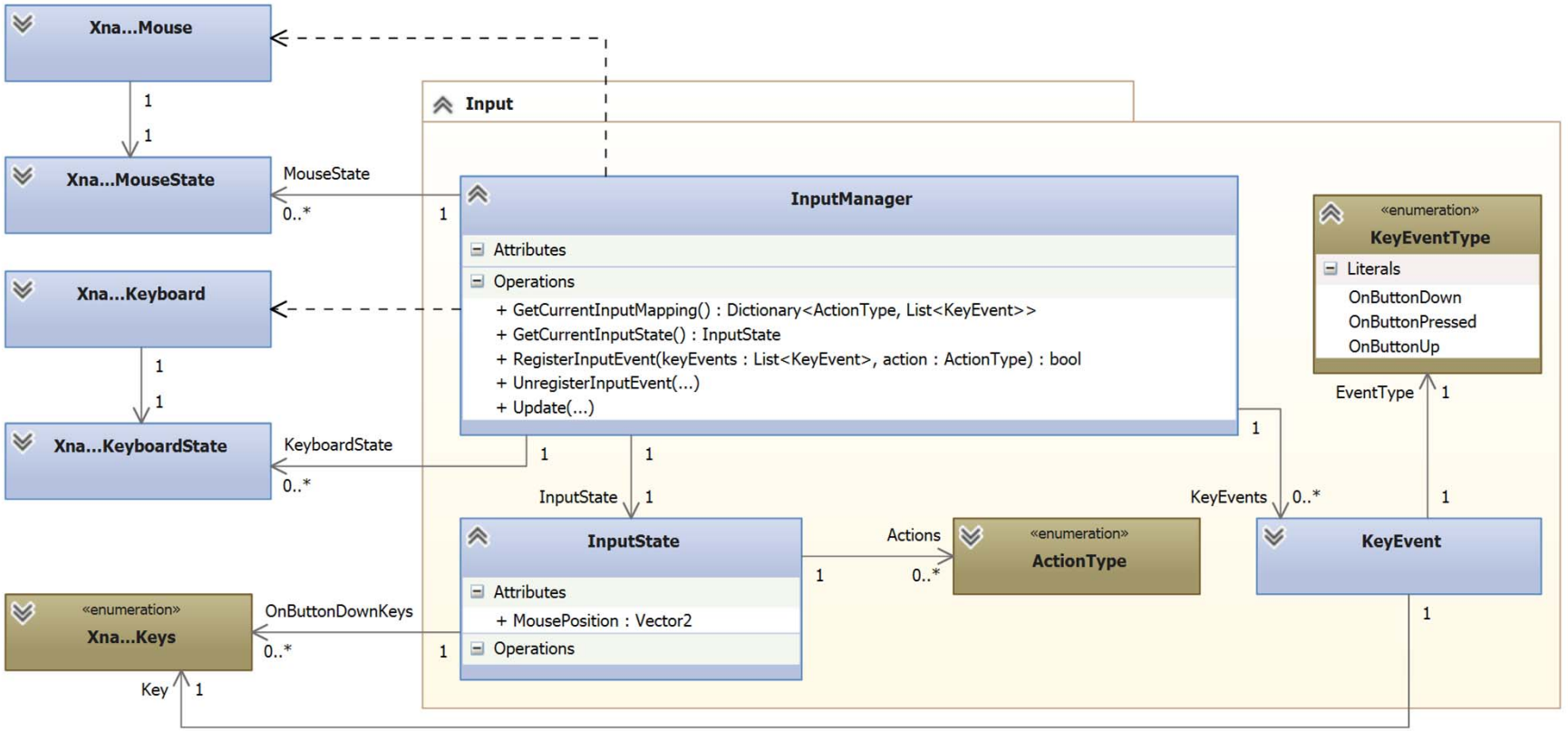
- Keyboard, Mouse, Gamepad, etc. sind Input Devices.
- Ereignisse von Input Devices heißen Inputs.
- Inputs können in MonoGame durch den Microsoft.Xna.Framework.Input Namespace abgefragt werden, z.B.:

```
KeyboardState k = Keyboard.GetState();
```

- State enthält Informationen wie Mausposition, Zustand einer Taste (gedrückt, nicht gedrückt), etc., aber keine Historie.

- Wir wollen:
  - Unterscheiden zwischen Taste „gerade eben“ gedrückt, Taste losgelassen, etc.
  - Keine Inputs verpassen.
  - Aus konkreten Inputs („A“ gedrückt) abstrakte Inputs machen (z.B. AttackAction).

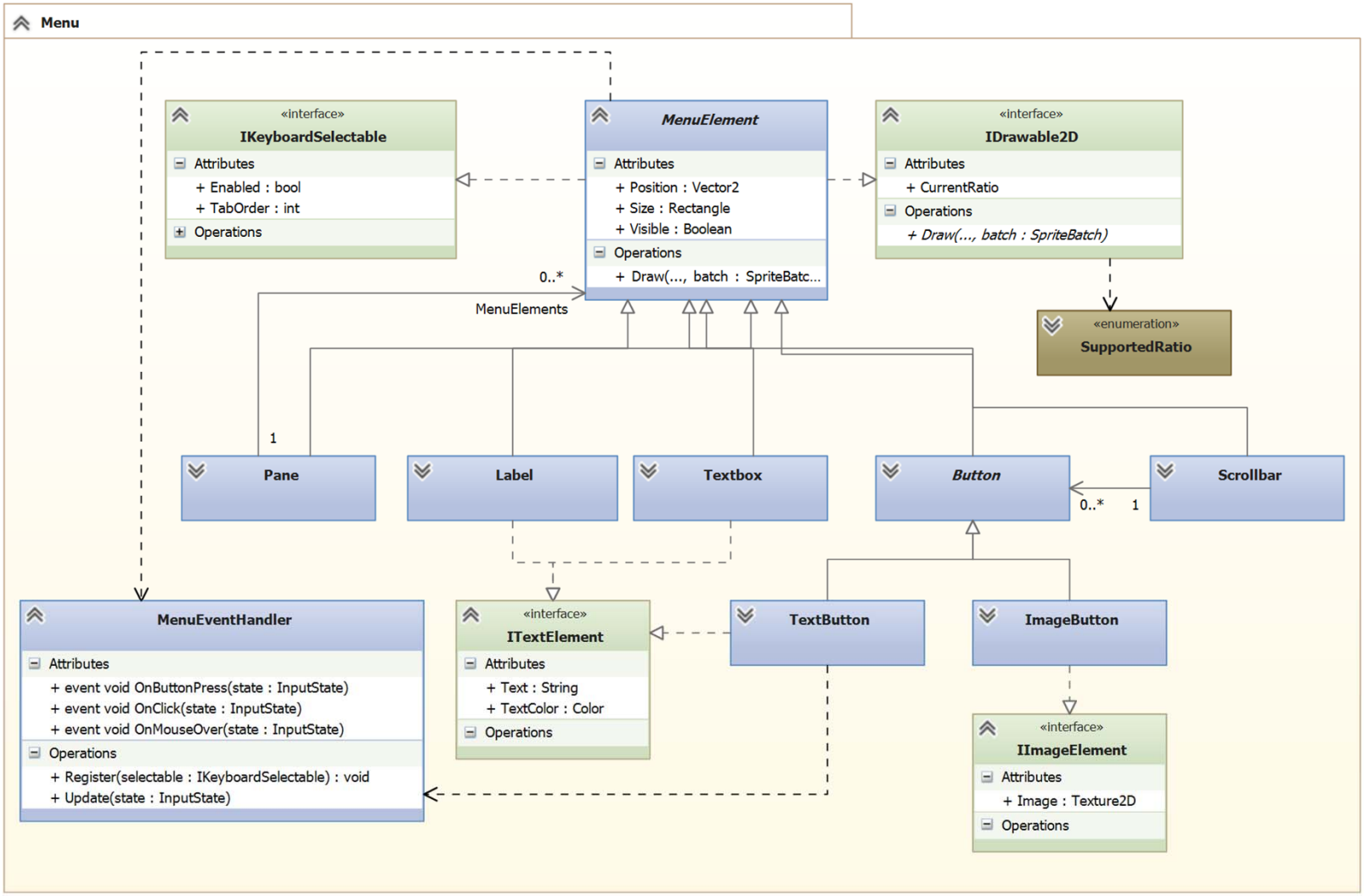
Inputs


Xna...Mouse
Xna...MouseState
Xna...Keyboard
Xna...KeyboardState
Xna...Keys
Input
InputManager
Attributes
Operations
+ GetCurrentInputMapping() : Dictionary<ActionType, List<KeyEvent>>
+ GetCurrentInputState() : InputState
+ RegisterInputEvent(keyEvents : List<KeyEvent>, action : ActionType) : bool
+ UnregisterInputEvent(...)
+ Update(...)
InputState
Attributes
+ MousePosition : Vector2
Operations
«enumeration»
KeyEventType
Literals
OnButtonDown
OnButtonPressed
OnButtonUp
«enumeration»
ActionType
KeyEvent
MouseState
KeyboardState
InputState
Actions
KeyEvents
EventType
OnButtonDownKeys
Key
1
0..*

Menü

- Ein Menü besteht aus vielen Elementen:
  Button, Label, Scrollbar, Textbox, Checkbox, …
- Menüelemente werden nicht nur im Hauptmenü verwendet, sondern auch im HUD.
- Probleme:
  - Positionierung
  - Input Handling
  - Verschiedene Auflösungen und Seitenverhältnisse

Simples Menü



Das eigentliche Spiel

- Besteht aus mehreren Screens:
  - GameScreen
  - HUD
  - Minimap
- Was muss im GameScreen alles getan werden?
  - Dinge zeichnen (Spielobjekte, Karte).
  - Aktionen des Spielers verarbeiten.
  - Spielzustand (Positionen, Ressourcen, Spielobjektzustände, etc.) verwalten.
  - …

- Offene Probleme
  - Screens dispatchen nur Update und Draw. Reicht das?
  - Wie teilen mehrere Screens Informationen?
  - …

PUZZLETEIL II: SPIELOBJEKTE

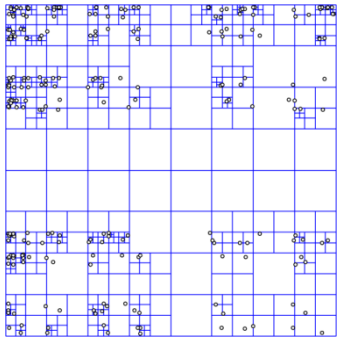
Spielobjekte

- Spielobjekte sind all die Dinge, die eine Repräsentation in der Spielwelt haben.
  - Charaktere, Fahrzeuge, Bäume, Raketen, Gras, Steine, Trigger, Lichtquellen, …
- Spielobjekte müssen manchmal…
  - gezeichnet werden.
  - sich bewegen.
  - zerstörbar sein.
  - miteinander kollidieren.
  - etc.
- Wie kann ich diese vielen Objektarten und ihre Operationen effizient verwalten?

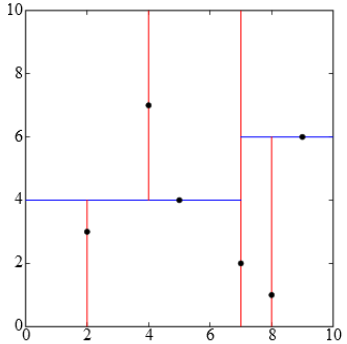
Szenengraph

- Ein Szenengraph ist eine zentrale Datenstruktur, die logische und/oder räumliche Repräsentation einer Szene verwaltet.
- Wird z.B. verwendet um
  - Antworten auf räumliche Fragen zu beschleunigen.
  - Update- und Draw-Aufrufe an alle Spielobjekte weiterzureichen.
- Beispiele

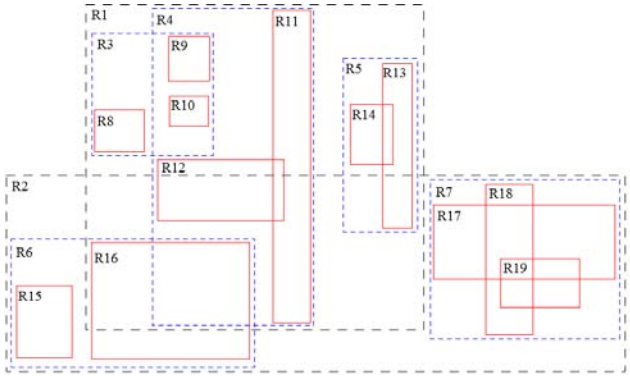
  Liste, Heap, Quad- / Octree, KD-Tree, R-Tree, …



Quadtree



KD-Tree



R-Tree

Spielobjekt-Architekturen

- Objekt-zentriert
  - Spielobjekte werden als Klassen implementiert und als Instanzen repräsentiert.
  - Eigenschaften und Verhalten wird durch die Klasse(n) gekapselt.
  - Spielwelt ist eine Ansammlung von Spielobjekt-Instanzen.

- Eigenschaften-zentriert
  - Eigenschaften der Spielobjekte werden als Tabellen gespeichert, eine pro Eigenschaft.
  - Spielobjekte sind nur eine ID.
  - Operationen sind nach wie vor Klassen/Instanzen, die diese Tabellen lesen oder modifizieren.
  - Ähnlichkeit zu relationalen Datenbanken.

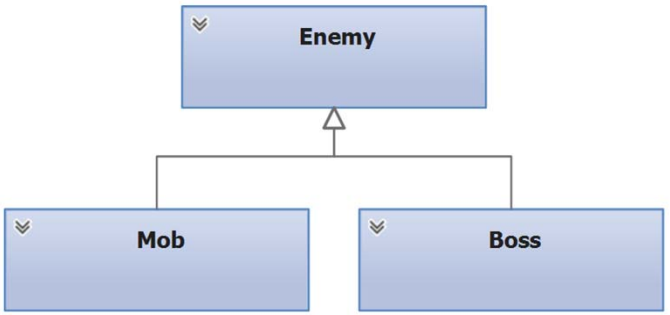
- …

Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.

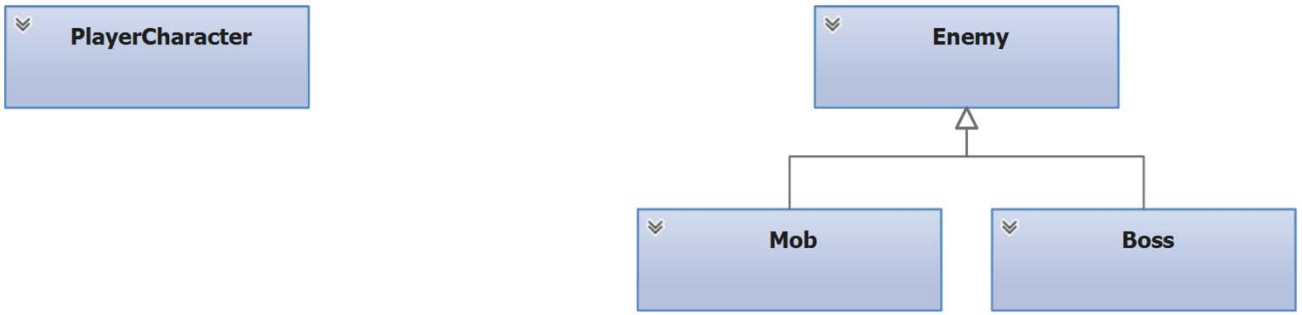
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



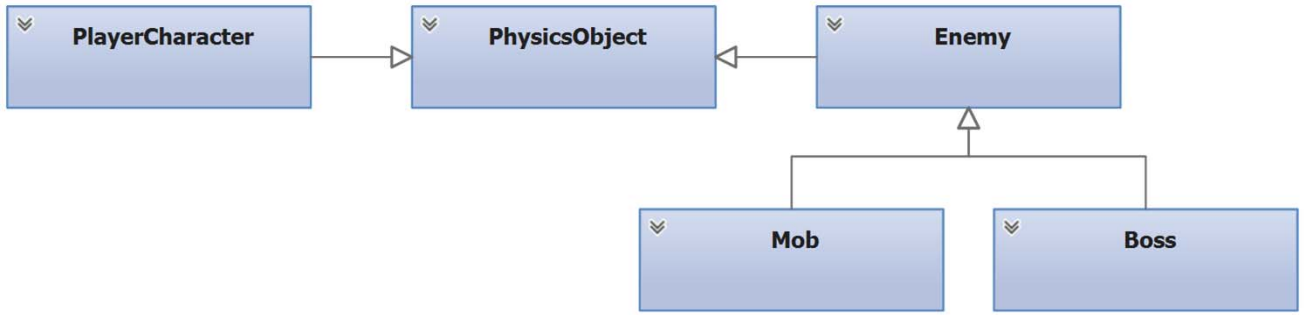
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



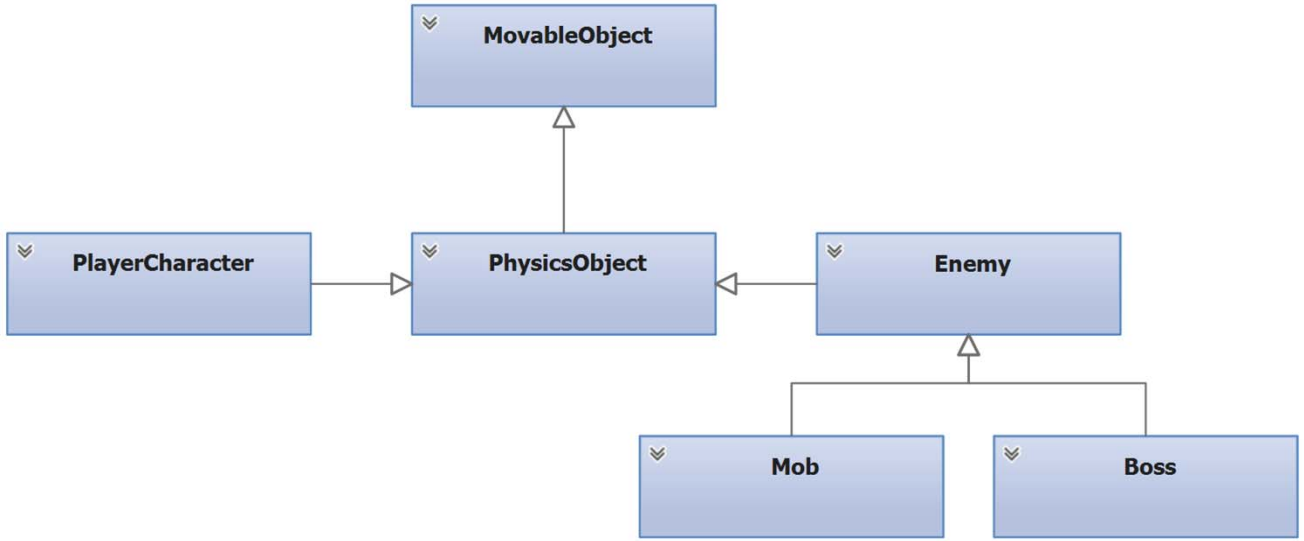
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



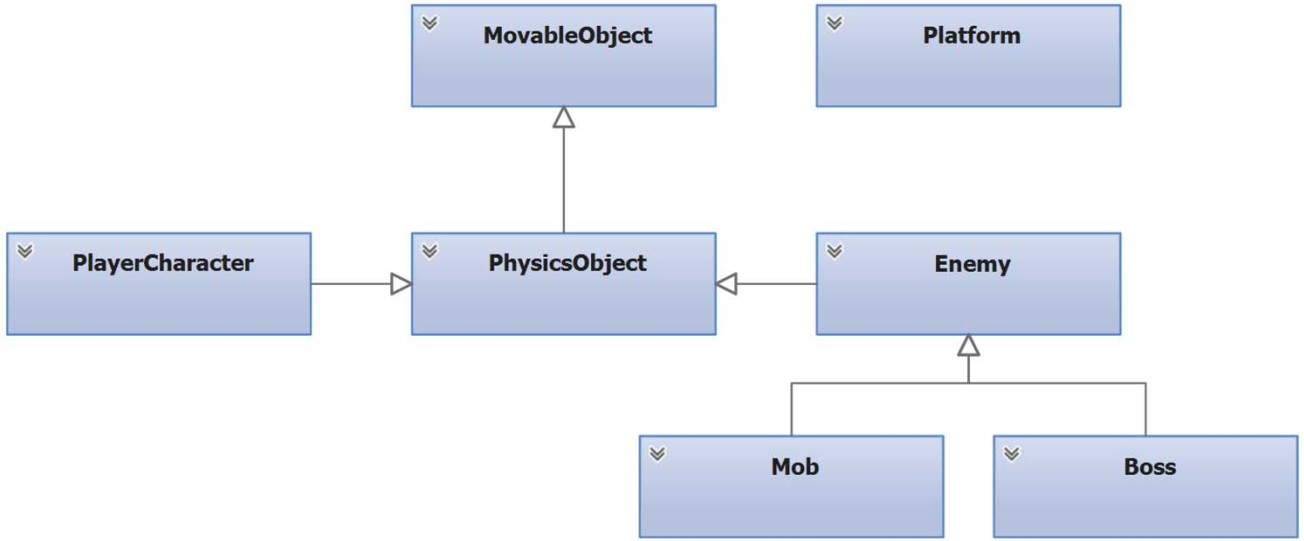
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



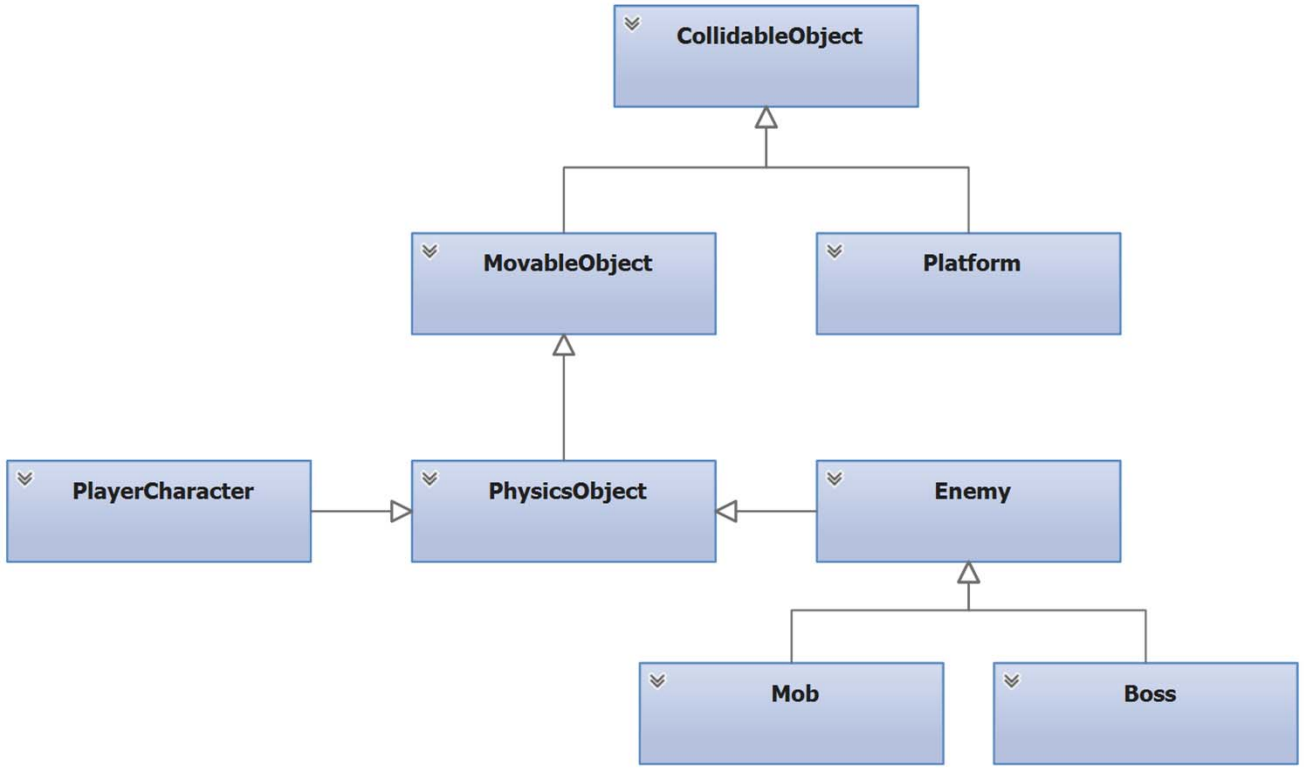
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



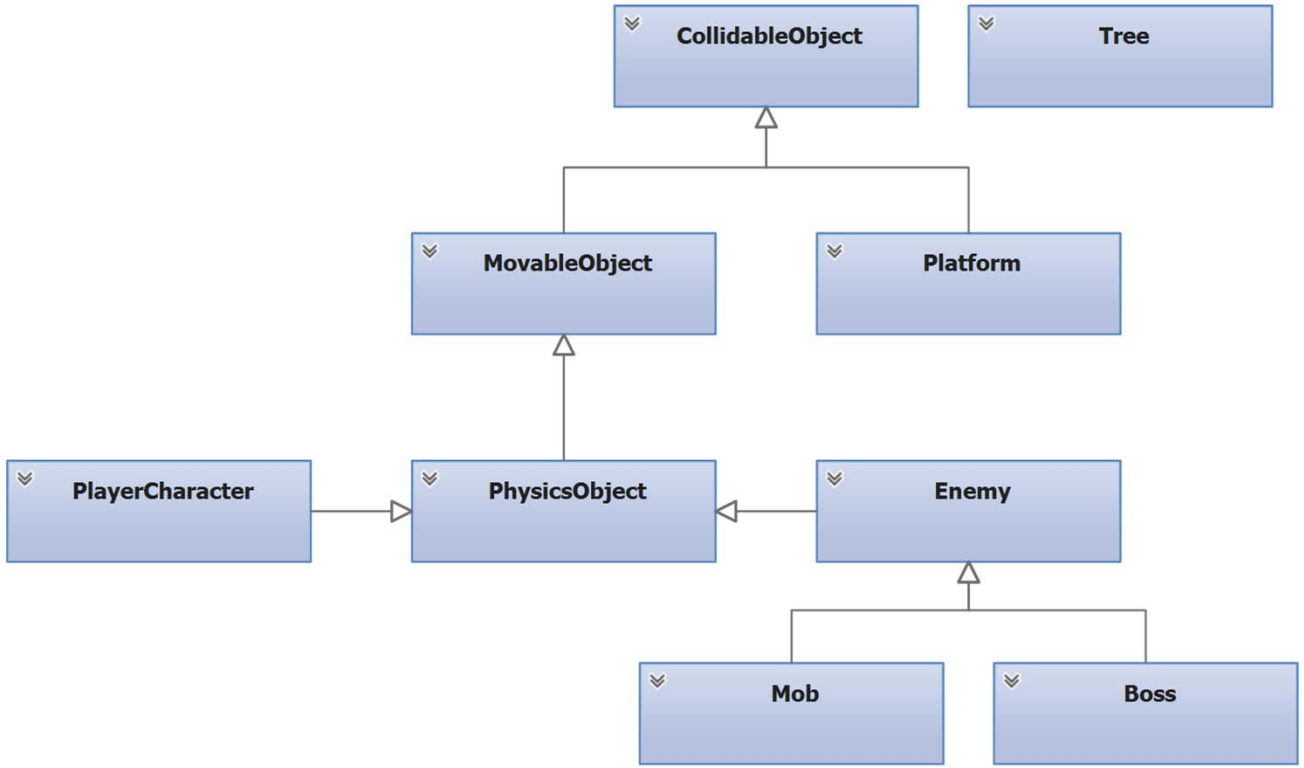
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



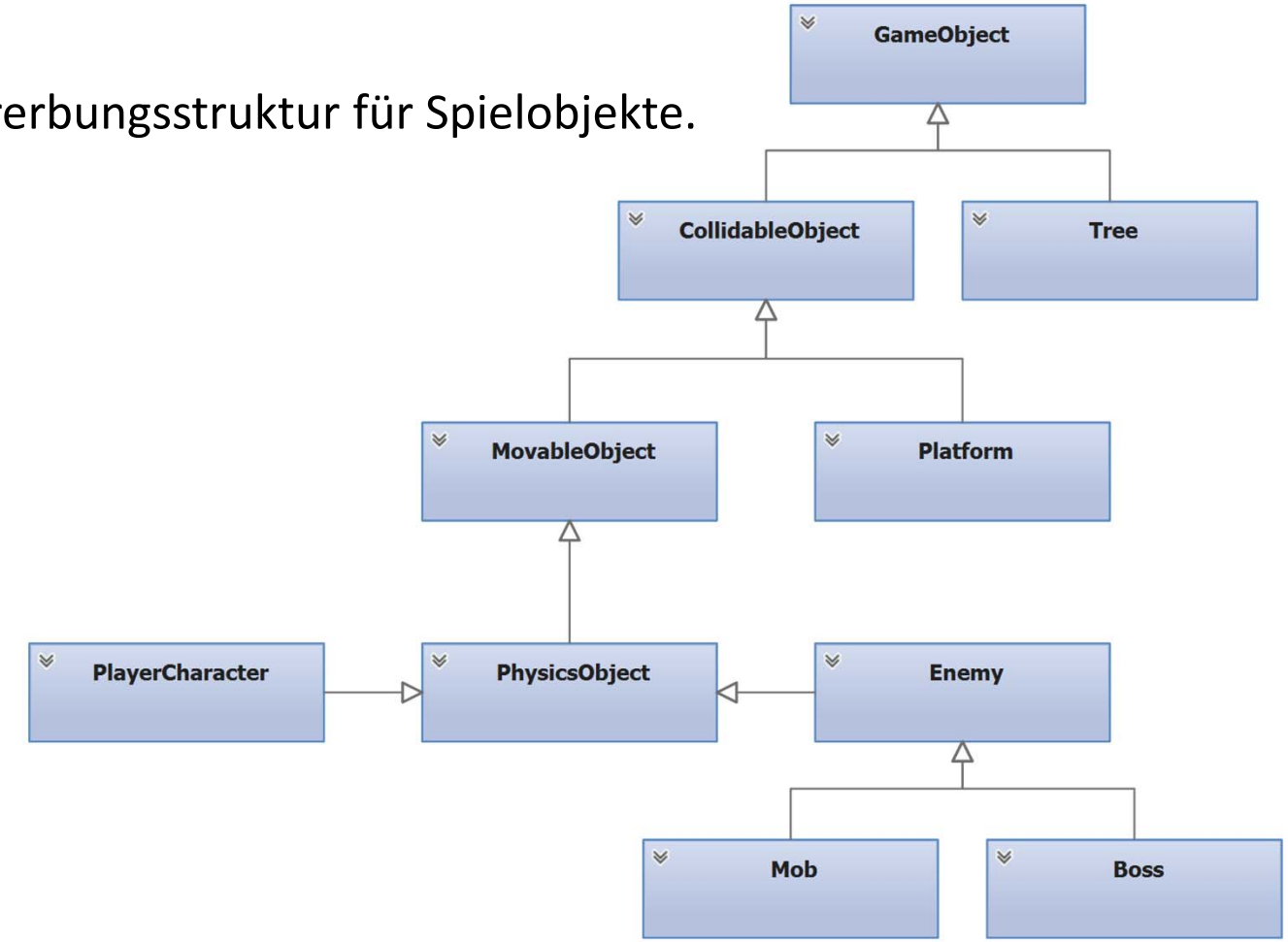
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



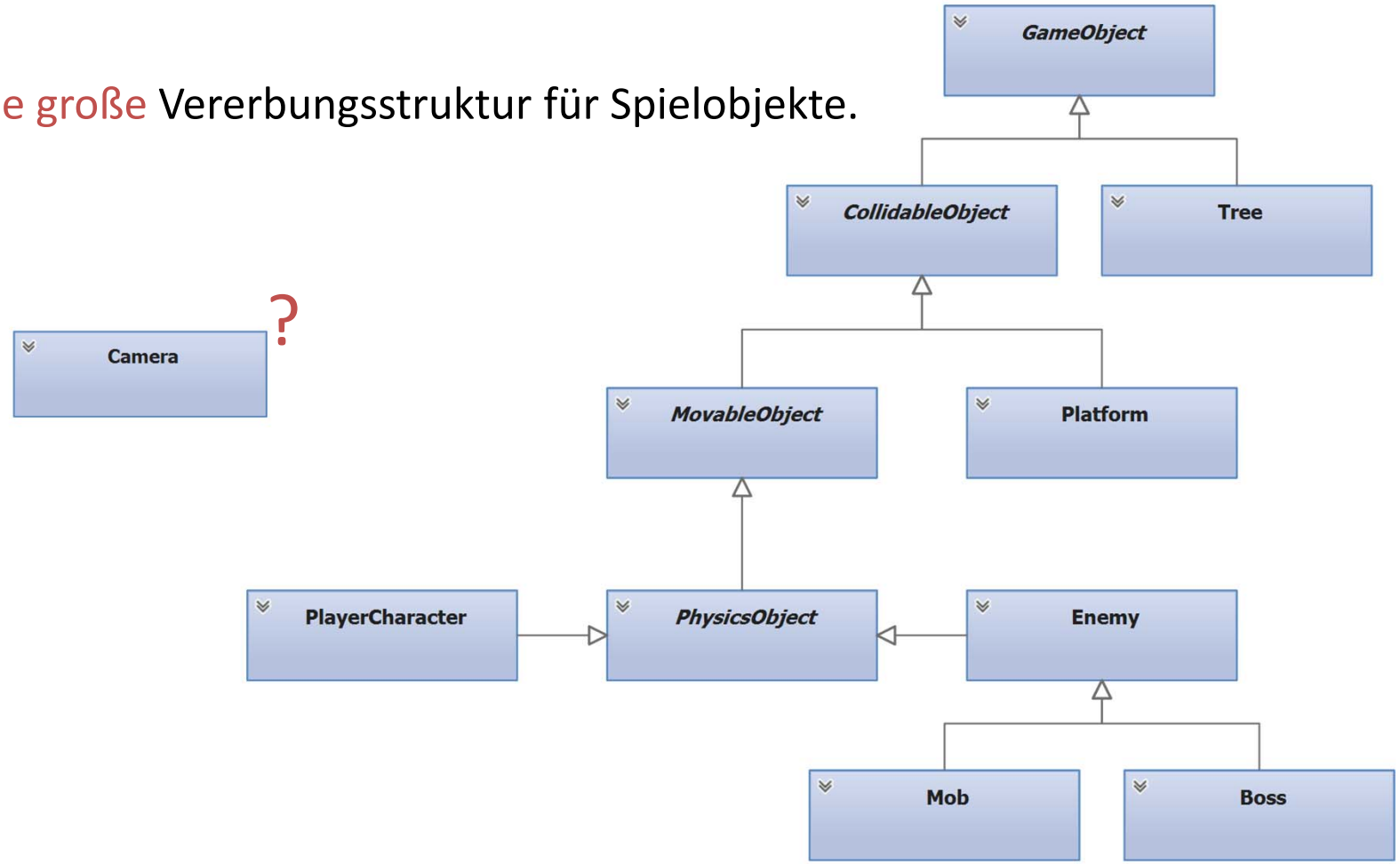
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



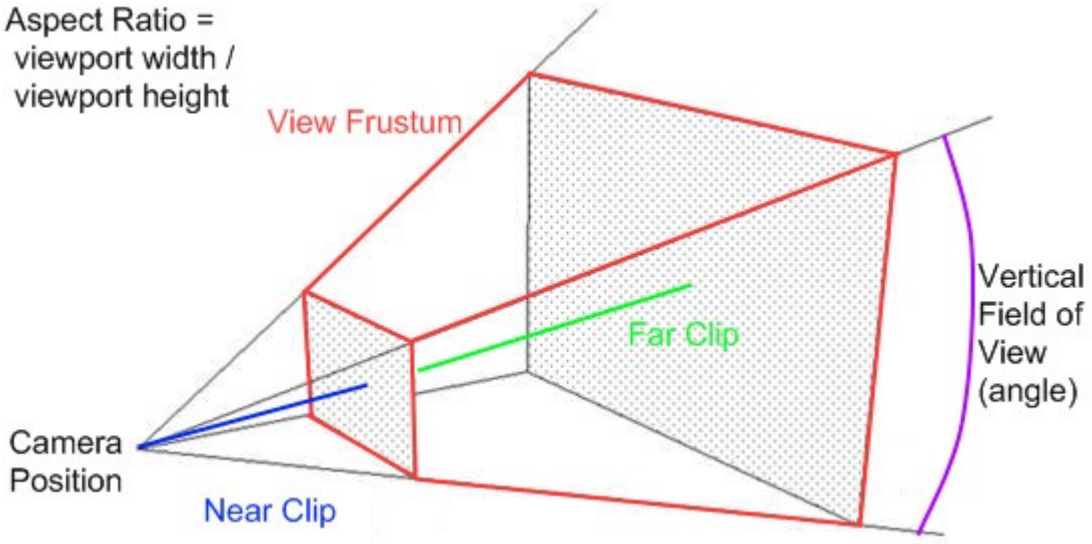
Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.



Exkurs: Simple Kamera

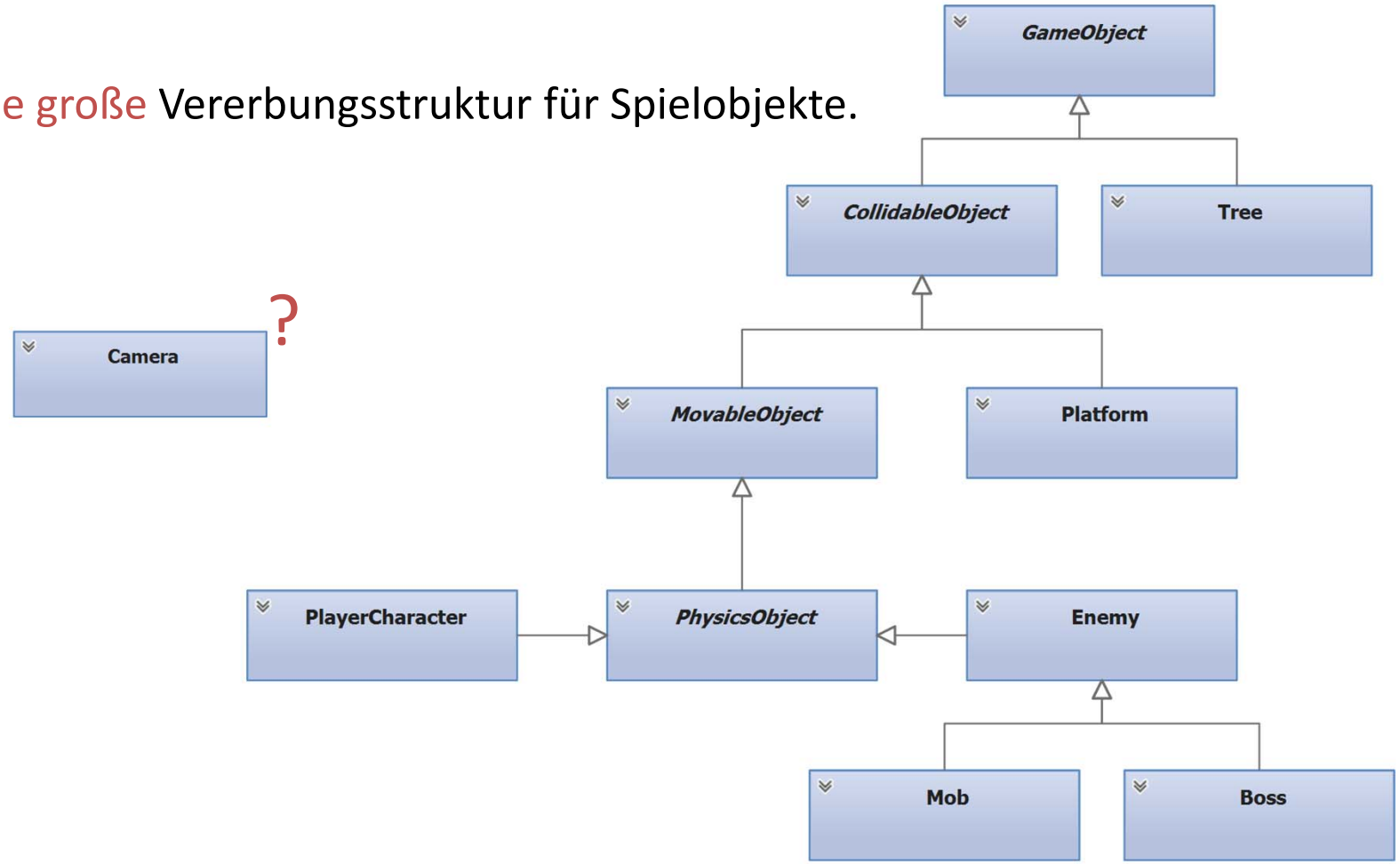
- Speichert Position, ViewMatrix, ProjectionMatrix.
- Grundlage für Picking.
- Definiert View Frustrum:



[http://ksgamedev.wordpress.com/tag/maths/]

Deep Hierarchy

- Eine große Vererbungsstruktur für Spielobjekte.

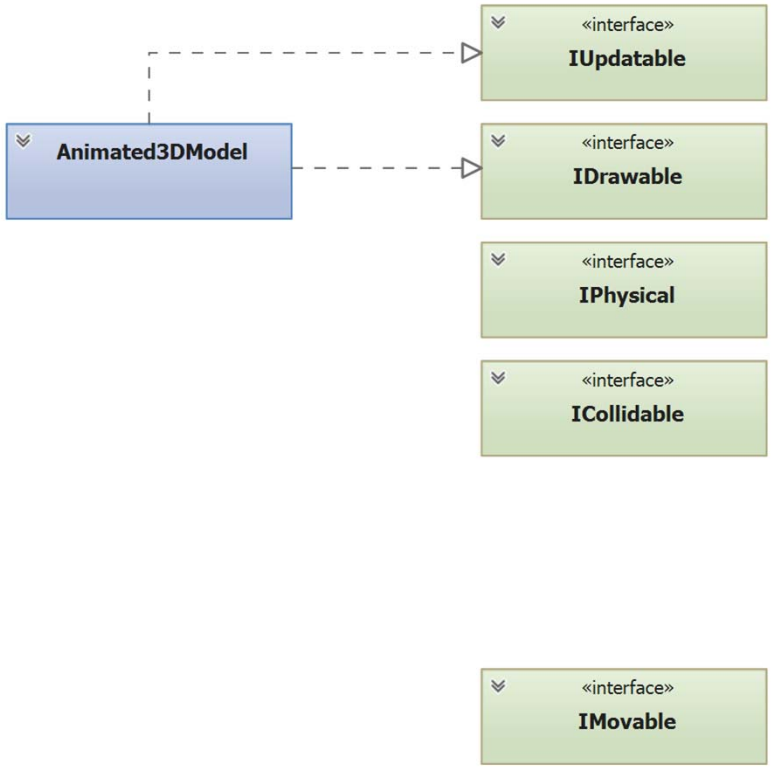


Wunsch

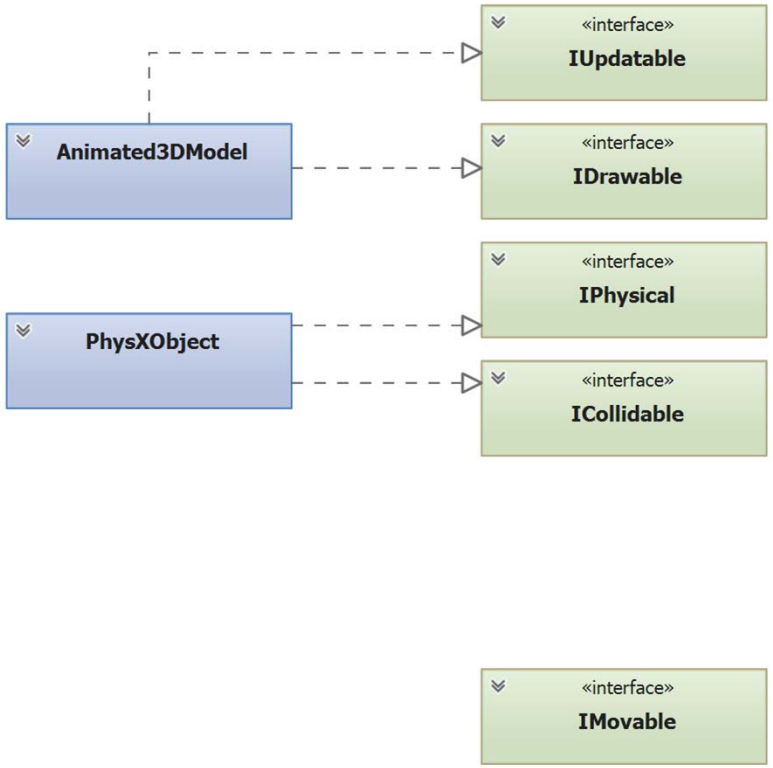
	Tree	Plattform	PlayerCharacter	Mob	Boss	Camera
GameObject	X	X	X	X	X	X
CollidableObject		X	X	X	X	
MoveableObject			X	X	X	X
PhysicsObject			X	X	X	
Enemy				X	X	

Komposition

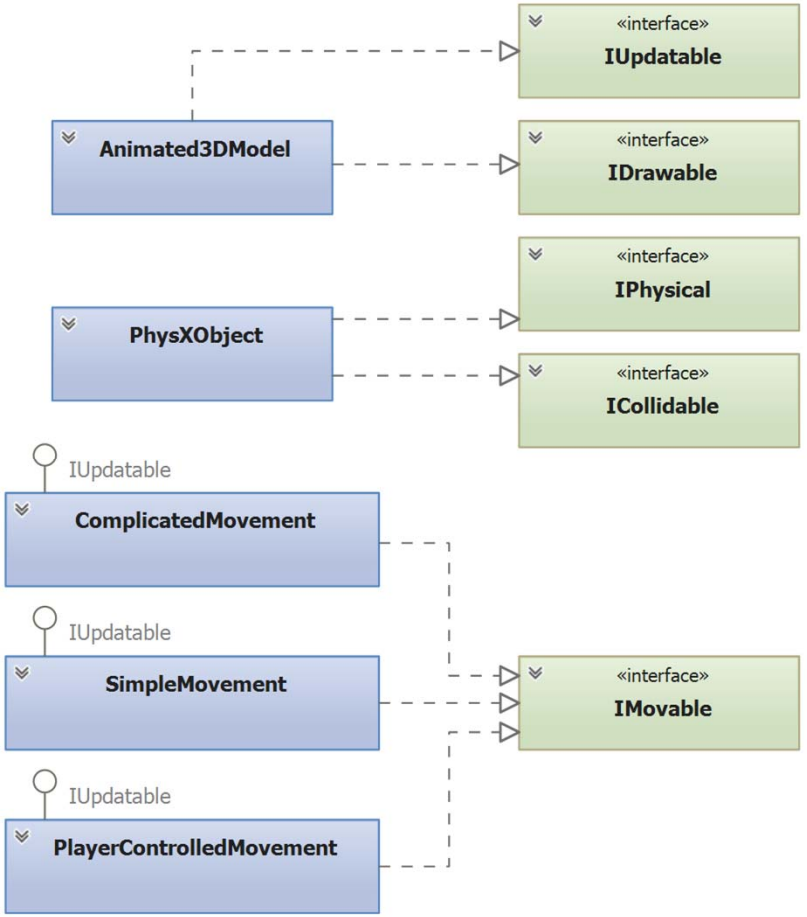
Komposition



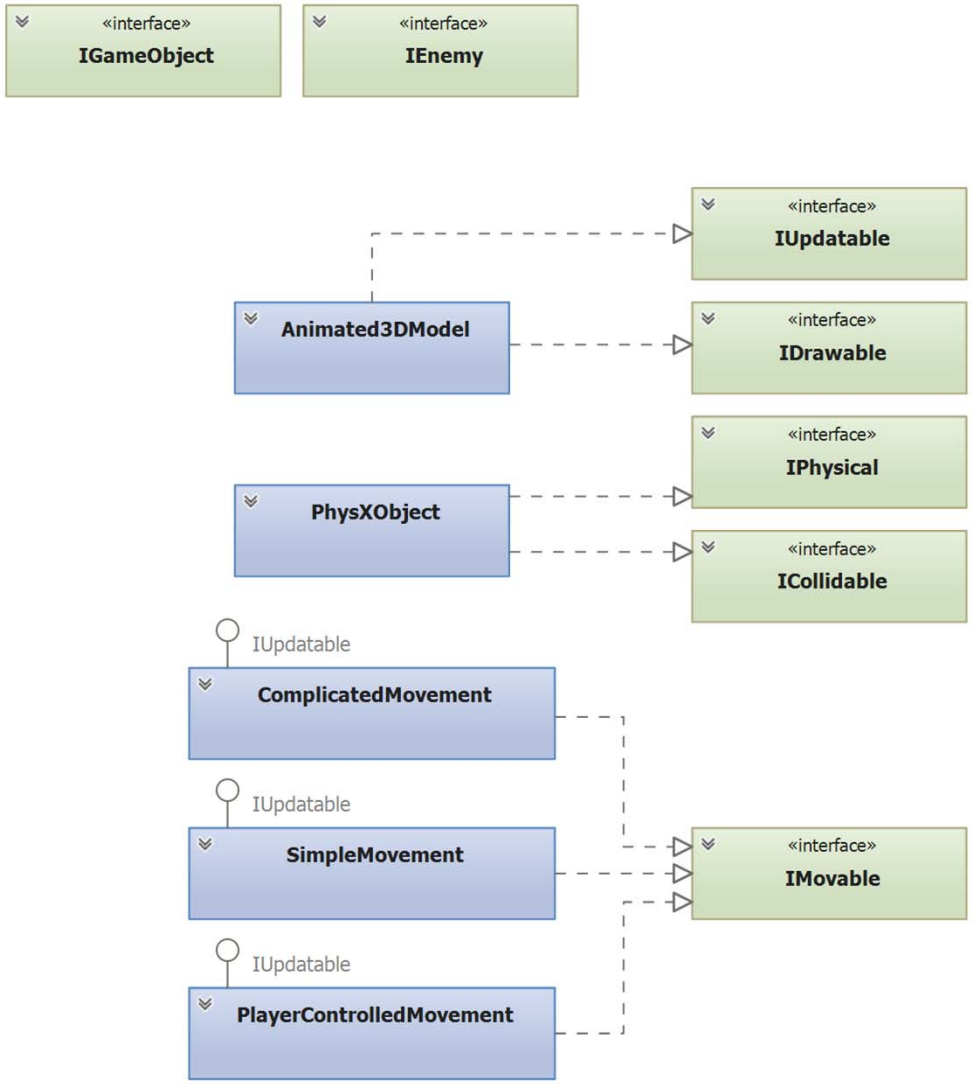
Komposition



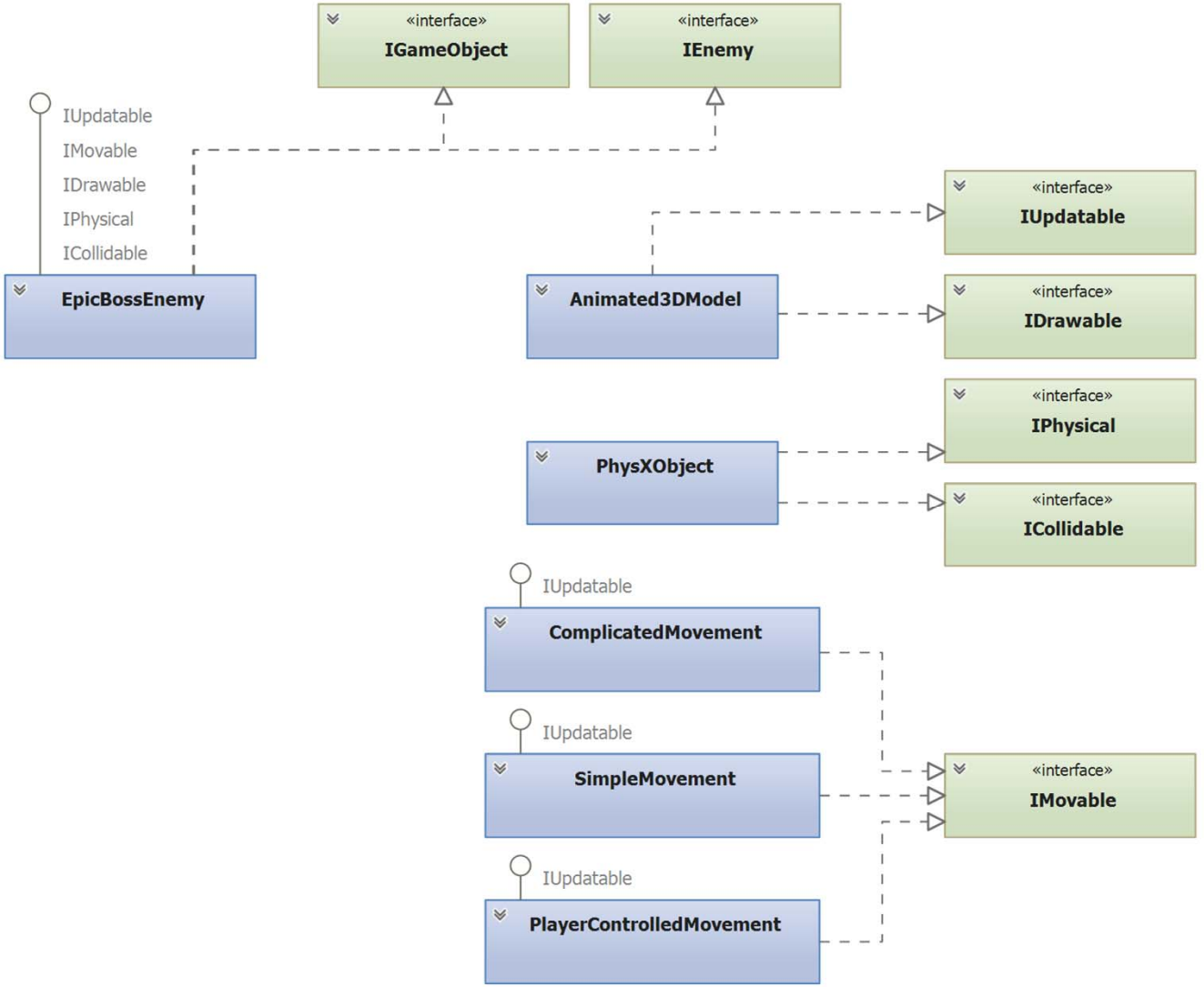
Komposition



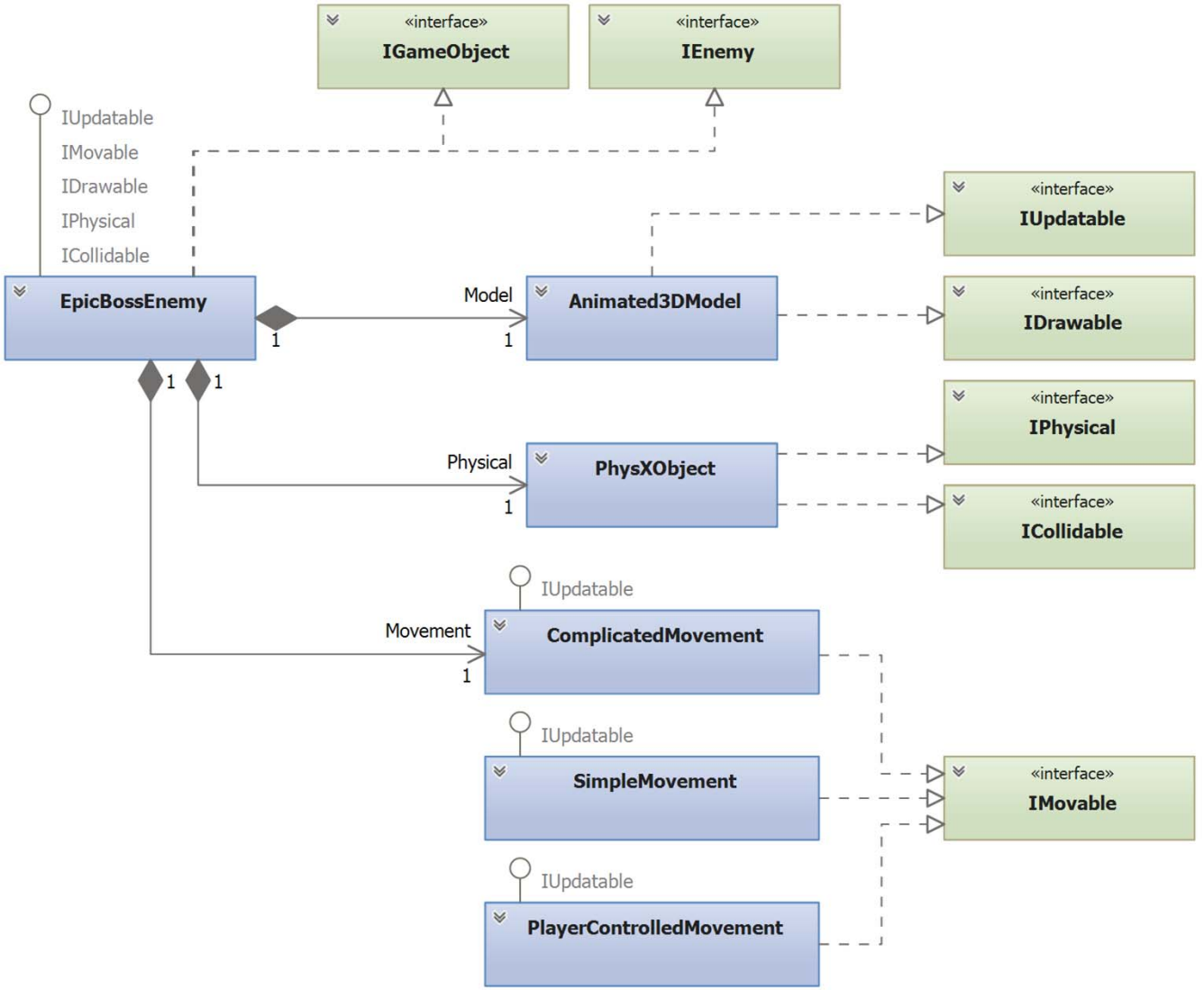
Komposition



Komposition



Komposition

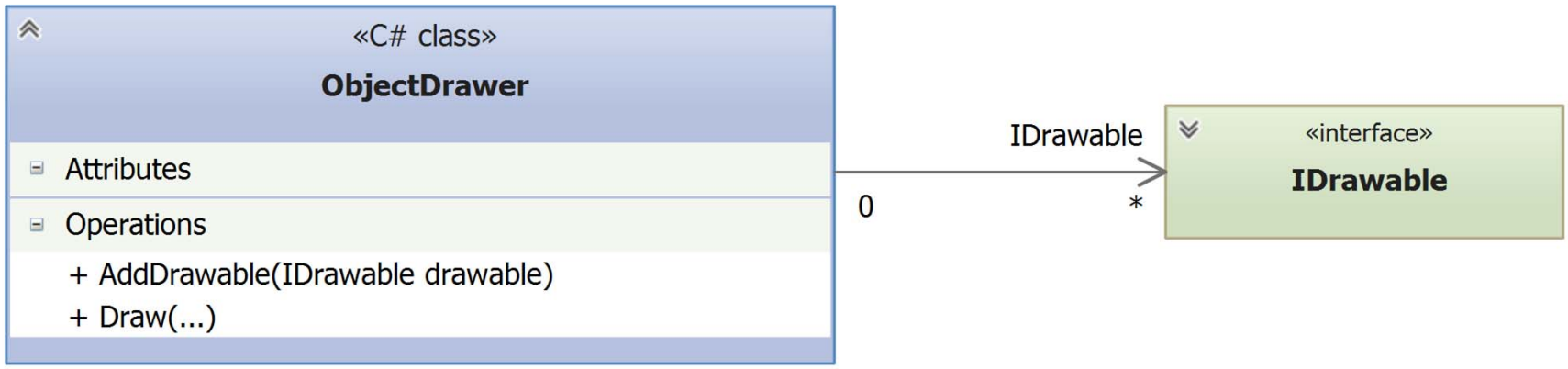


Komposition

```
public class EpicBossEnemy : IGameObject,
      IEnemy, IUpdatable, IDrawable, IMovable,
      IPhysical, ICollidable
{
    private IDrawable Model { get; }
    private IMovable Movement { get; }
    private IPhysical Physical { get; }
    private HealthBar HPBar { get; }

    public EpicBossEnemy()
    {
        Model = new Animated3DModel("Boss1");
        Movement = new ComplicatedMovement();
        Physical = new PhysXObject();
        HPBar = new HealthBar(1250);
    }
```

```
    public void Draw(GameTime t)
    {
        // Draw Model
        Model.Draw(t);
        // Draw HP Bar
        HPBar.Draw(t, Position);
    }

    public void Move()
    {
        Movement.Move();
    }

    public void Update()
    {
        Physical.Update();
        // Other Updates...
    }
}
```

Komposition




```
public class ObjectDrawer
{
   public ICollection<IDrawable> Drawables { get; }

   public void AddDrawable(IDrawable drawable)
   {
      Drawables.Add(drawable);
   }
```

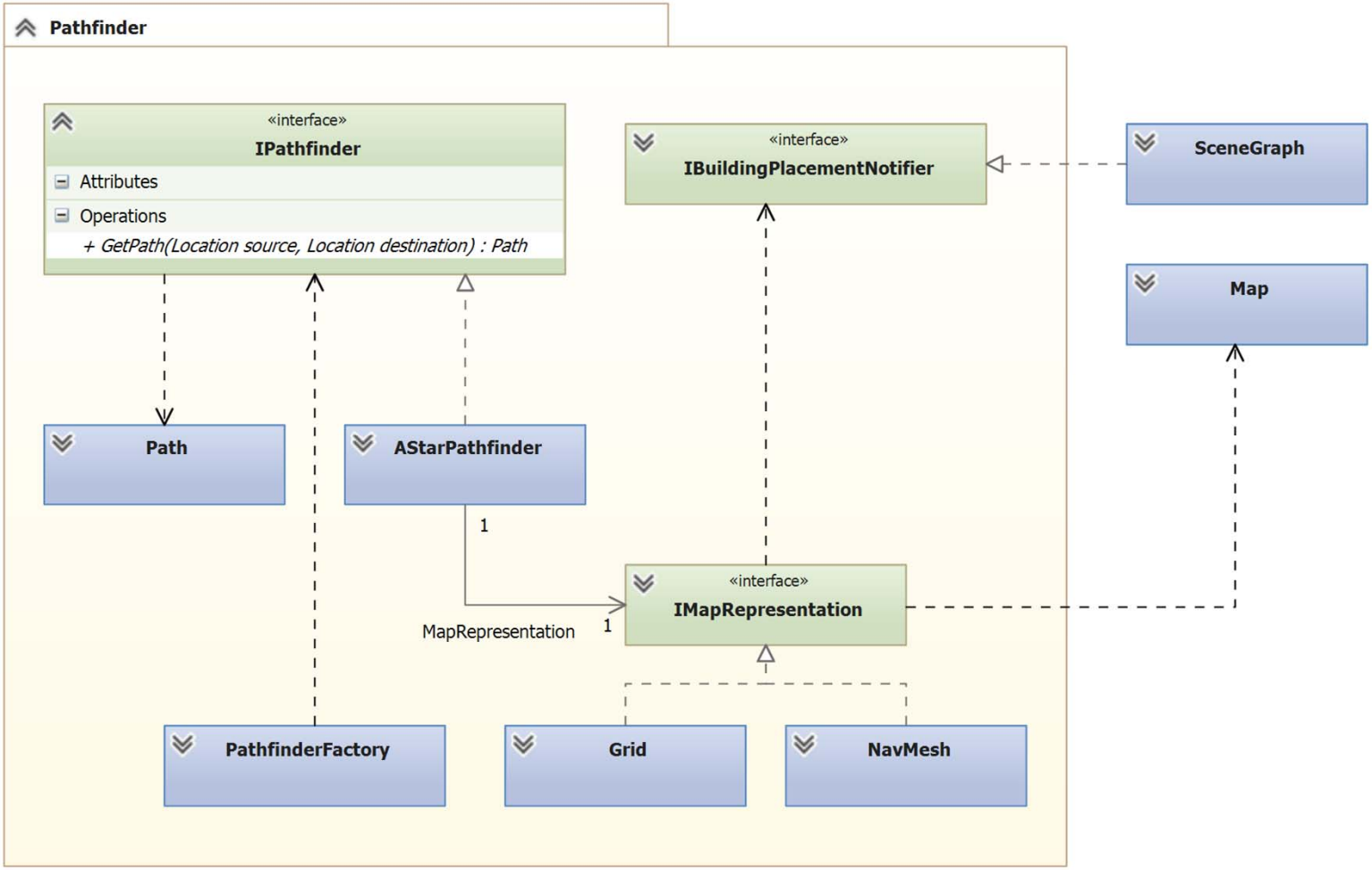
```
   public void Draw(GameTime t)
   {
      foreach (var drawable in Drawables)
      {
         drawable.Draw(t);
      }
   }
}
```

PUZZLETEIL III: PATHFINDING

Pathfinding

- Zwei Arten: Online und Offline Pathfinding
- Offline
  - Berücksichtige nur die Welt und vielleicht unbewegliche Objekte.
  - Im wesentlichen immer A* als Suchalgorithmus.
  - Viele Möglichkeiten für Weltrepräsentation:
    Grid, Hierachical Grid, Waypoint Graph, Navigation Mesh, …
- Online
  - Wie weiche ich beweglichen Objekten aus?
  - Viele (parametrisierbare) Möglichkeiten:
    Steering, Flocking, Flow Fields, …
  - Kann als Bewegungsmodul in Spielobjekten implementiert werden.

Pathfinding



ZUSAMMENBAU?

Ein weites Feld…

- Sehr viele Fragen bleiben offen.
  - Netzwerk-Multiplayer?
  - Sound?
  - Zeichnen und Grafikeffekte (Z-Buffer, Shader, Perspektiven, Partikel, Performance, …)?
  - Online-Pathfinding?
  - KI?
  - Laden/Speichern?
  - Player?
  - Mathematik?

- Kommen Sie in die Poolsprechstunde, um sich speziell für Ihr Spiel beraten zu lassen.

FRAGEN?

Quellen

- [Gregory, 2009] Gregory, J. (2009). Game Engine Architecture. A K Peters Limited. ISBN 1568814135, 9781568814131.
- [http://ksgamedev.wordpress.com/tag/maths/]