Grundlagen

Christopher Schleiden
i-chschl@microsoft.com


www.sechsta-sinn.de

Agenda

- Überblick
- Grundlage/Fenster erzeugen
- 2D Grafik laden/anzeigen
- Eingabe(n) abfragen
- 2D Sprite bewegen
- Kollionsabfrage
- Animation
- Sound

XNA ÜberBlick

History

- 2006 Dez – XNA Game Studio Express 1.0
- 2007 April – XNA Game Studio Express 1.0 Refresh
- 2007 Dez – XNA Game Studio 2.0
- 2008 Okt – XNA Game Studio 3.0
- 2009 März – XNA Game Studio 3.1

Was ist XNA?

- XNA (**X**NA's **N**ot **A**cronymed)
- Framework zur Spieleentwicklung (2D & 3D) für Windows, Xbox 360 und Zune (auch Zune HD)
- Übernimmt Funktionen wie grafische Ausgabe, Wiedergabe von Audio-Dateien, Abfragen von Eingabegeräten

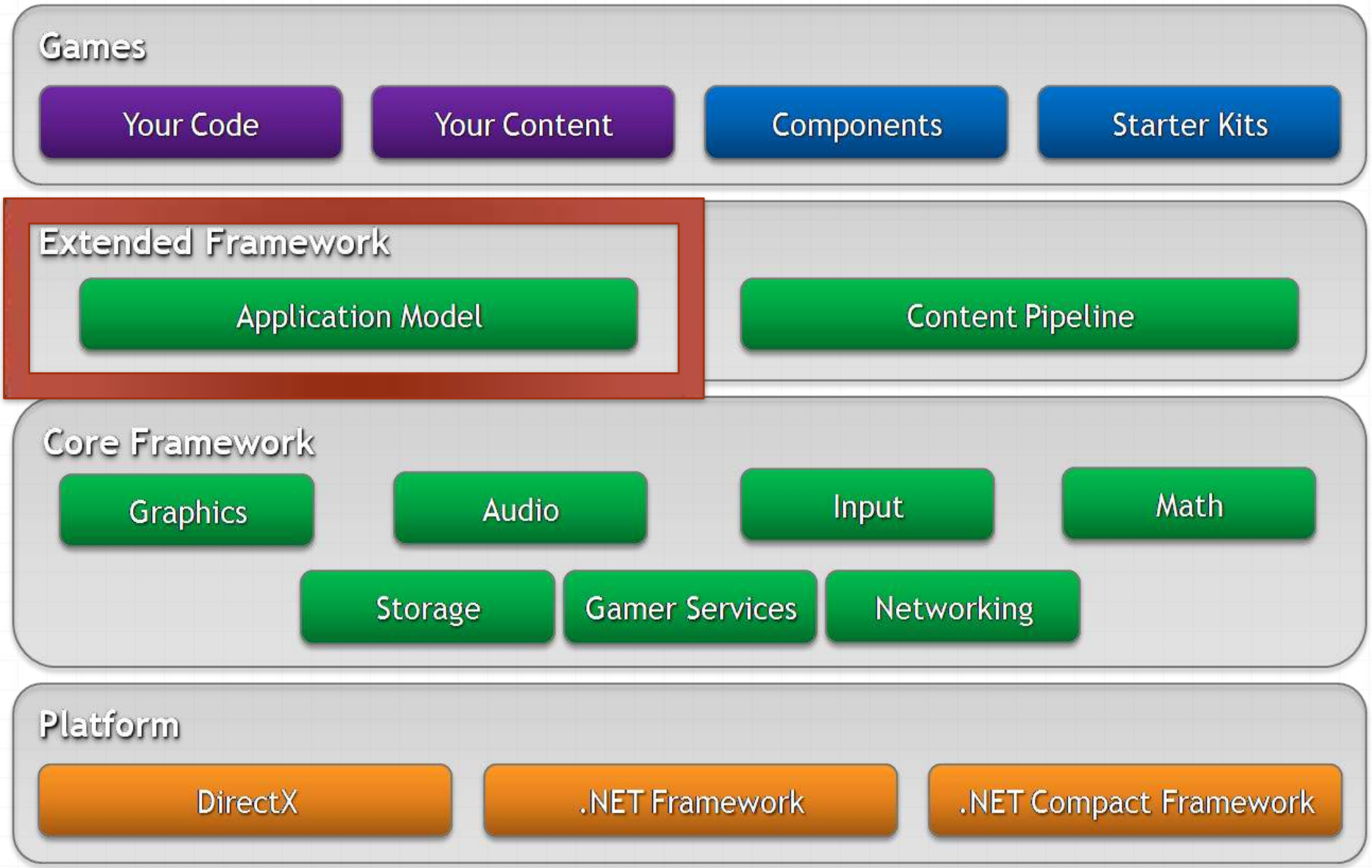
Was ist XNA nicht?

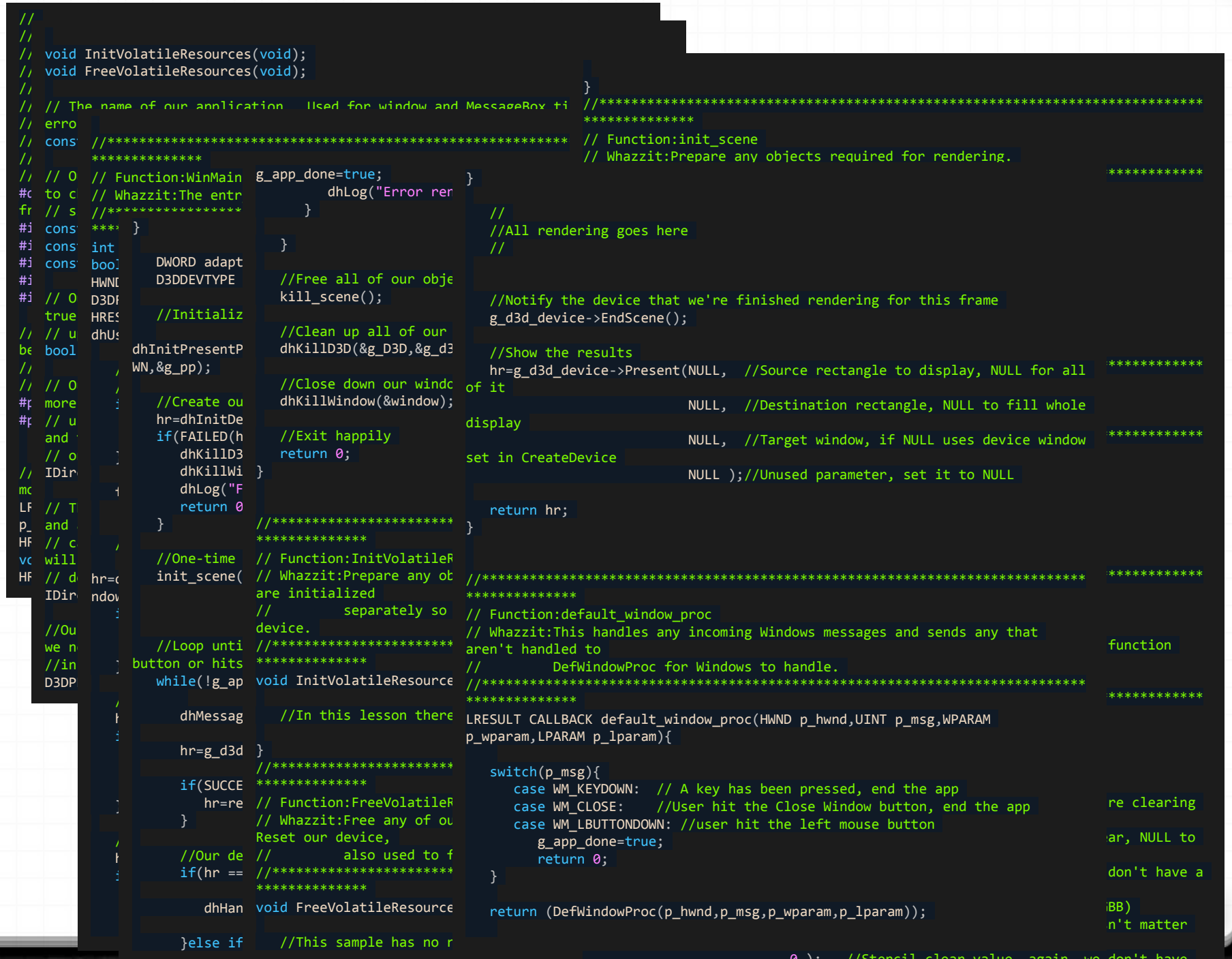
- “Spiele-Engine”/Midleware wie z.B. Unity, Unreal Engine oder CryEngine
  - Keine Spielmechnismen, Level Editoren, Kollisionsbehandlung etc.

- ABER:
  Viele Beispiele, komplette Spiele (Starter Kits) direkt von Microsoft zur freien Verwendung

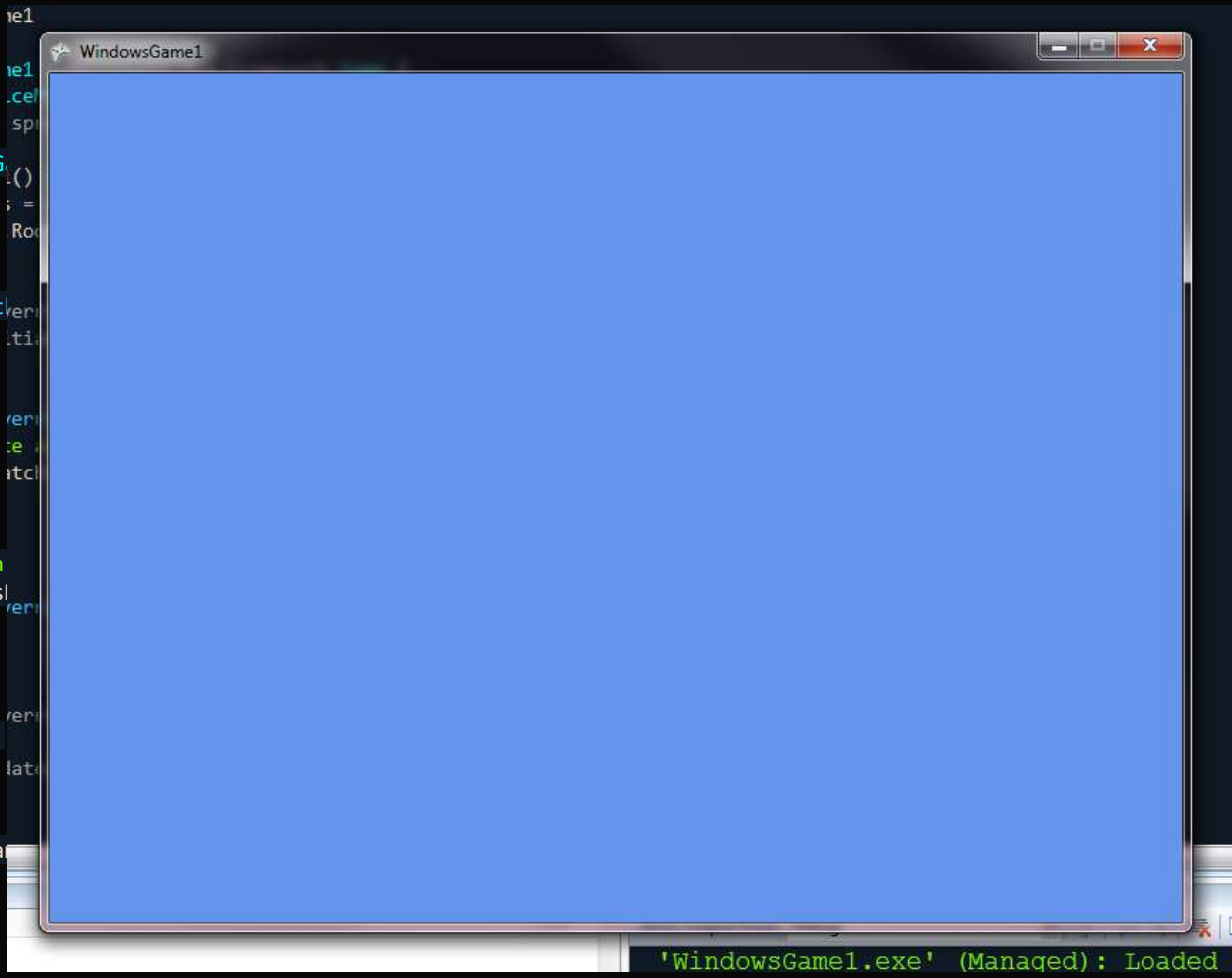
Entwicklung mit XNA

- Entwickelt wird in C#
  - Keine DirectX Kenntnisse notwendig
  - Einzige andere wichtige Sprache ist HLSL (**H**igh **L**evel **S**hader **L**anguage) zur Shaderprogrammierung
- XNA Anwendungen
  - Managed Code, alles wird von CLR ausgeführt
  - rufen .NET und DirectX Funktionen auf


Games
Your Code
Your Content
Components
Starter Kits
Extended Framework
Application Model
Content Pipeline
Core Framework
Graphics
Audio
Input
Math
Storage
Gamer Services
Networking
Platform
DirectX
.NET Framework
.NET Compact Framework

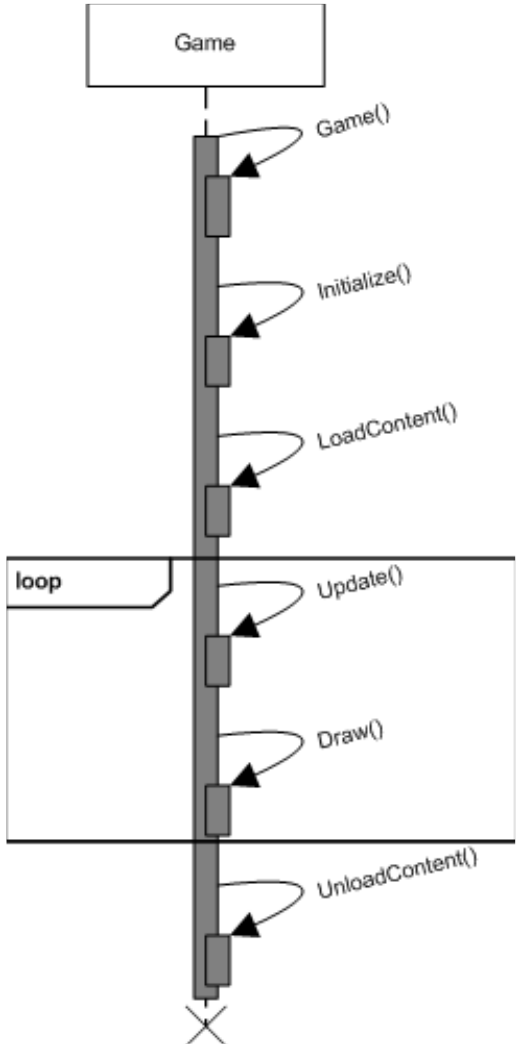


```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace WindowsGame1
{
    public class Game1 : Microsoft.Xna.Framework.G
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        public Game1() {
            graphics = new GraphicsDeviceManager(t
            Content.RootDirectory = "Content";
        }

        protected override void Initialize() {
            base.Initialize();
        }

        protected override void LoadContent() {
            // Create a new SpriteBatch, which can
            spriteBatch = new SpriteBatch(Graphics

        }

        protected override void UnloadContent() {

        }

        protected override void Update(GameTime ga

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime) {
            GraphicsDevice.Clear(Color.CornflowerBlue);


            base.Draw(gameTime);
        }
    }
}
```



Application Model

- Initialize()
- LoadContent()
- UnloadContent()
- Update()
- Draw()



Gameloop Bestandteile

- **Update**
  Aktualisiert die Spiellogik der Anwendung

- **Draw**
  Zeichnet veränderten Zustand der Anwendung auf den Bildschirm

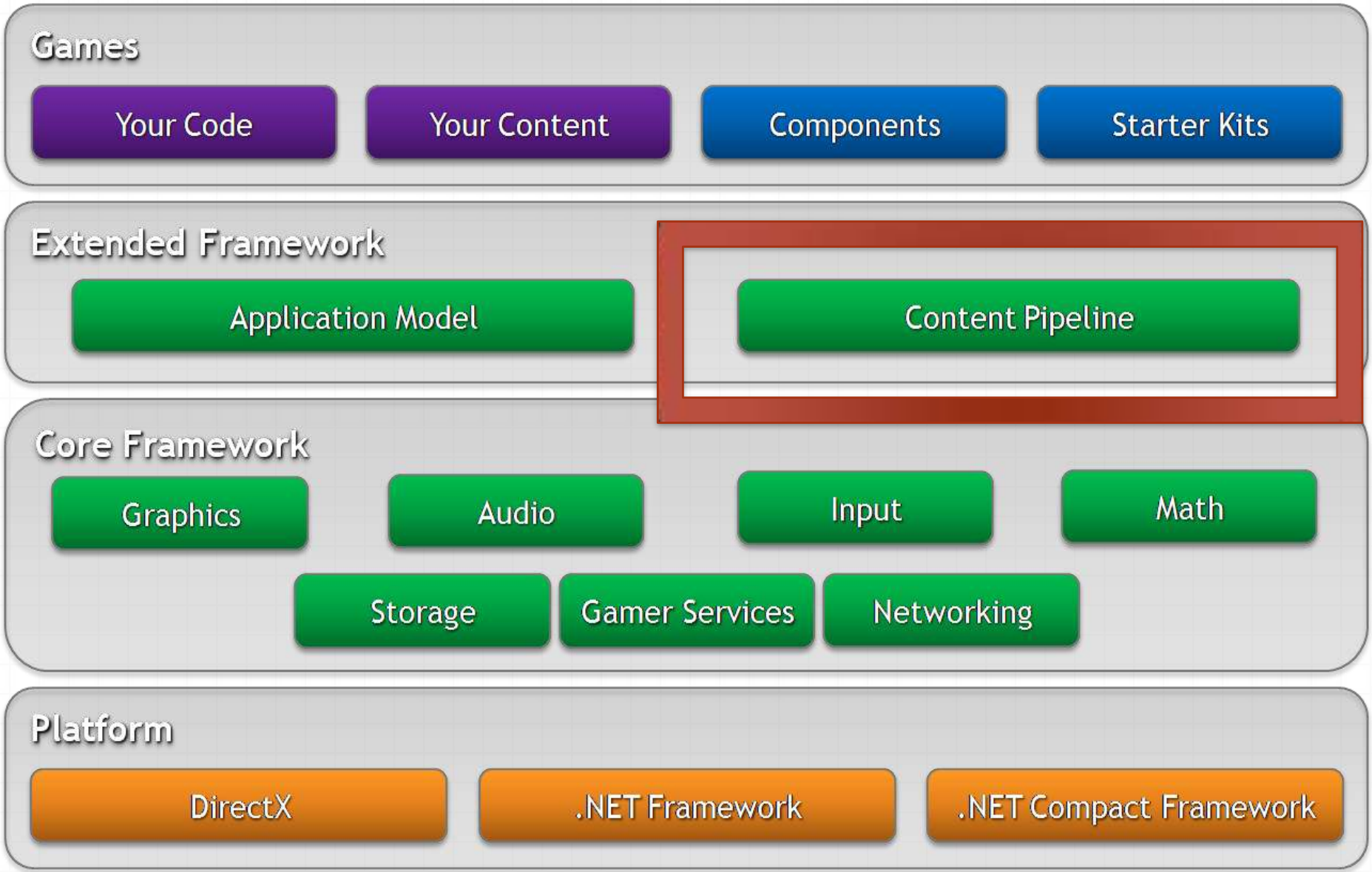
Timing

- Default: Fixed-Timestep mit Ziel-Framerate **60** FPS
- Pro Sekunde wird 60 mal **Update** aufgerufen
  - d.h. 16,67 Millisekunden pro Frame
- So oft wie möglich dazu noch **Draw**

Timing verwenden

- Update und Draw besitzen ein GameTime Objekt
  - z.B. Zeitspanne zwischen zwei Aufrufen von Update() oder Draw()

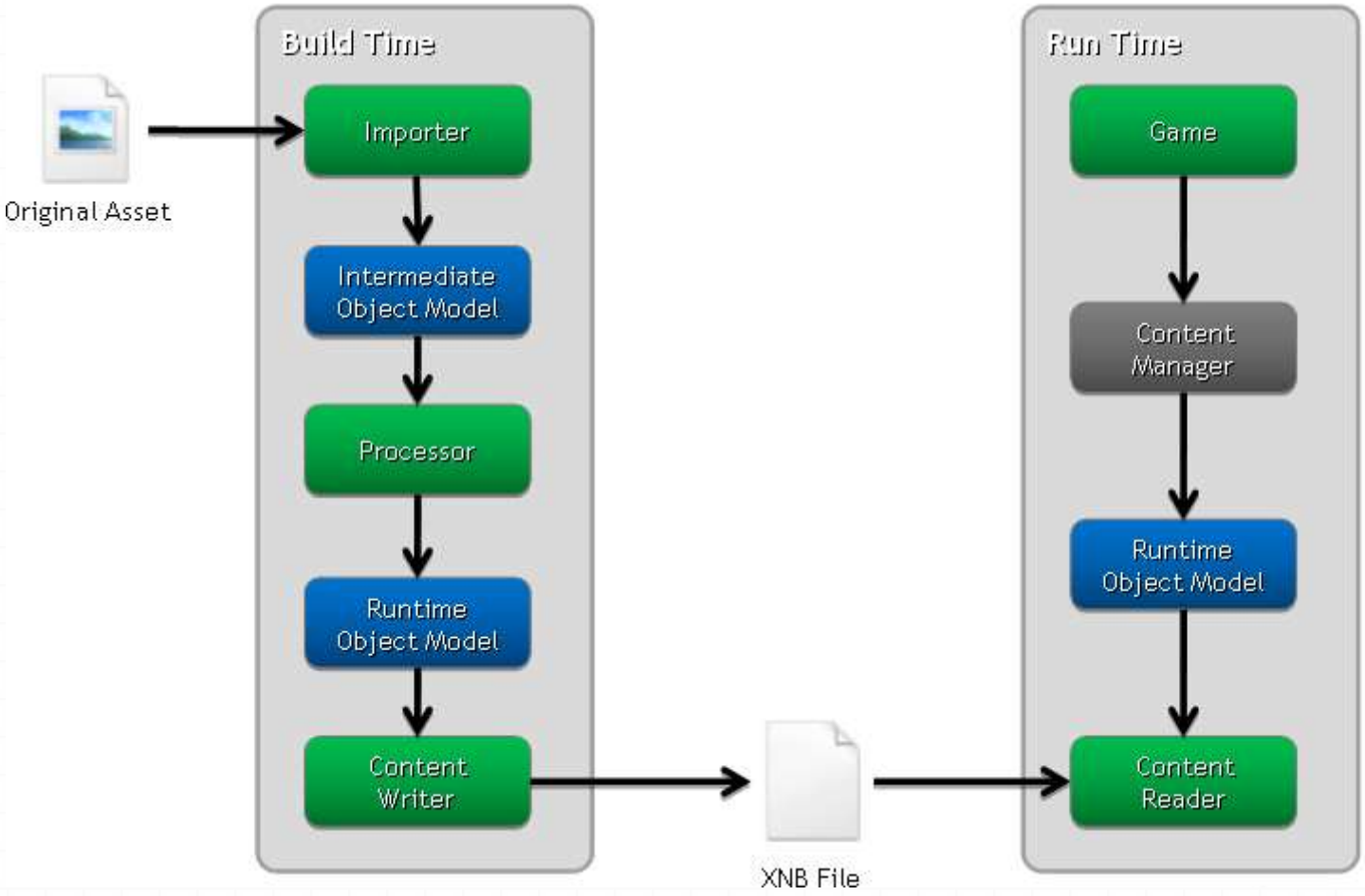
```
protected override void Update(GameTime
   gameTime)
{
   int elapsed =
   gameTime.ElapsedGameTime.Milliseconds;
}
```


Games
Your Code
Your Content
Components
Starter Kits
Extended Framework
Application Model
Content Pipeline
Core Framework
Graphics
Audio
Input
Math
Storage
Gamer Services
Networking
Platform
DirectX
.NET Framework
.NET Compact Framework

Content Pipeline

- Inhalte werden nicht im Original-Format geladen
- Inhalte werden mittels Content Pipeline in eigenes XNA Format übersetzt (XNA Binary, .xnb)
- Dabei werden diese für die Verwendung vorbereitet:
  - 3D Modelle in XNA eigene Datenstruktur laden
  - z.B. 3D-Modelle mit Textur zusammenfügen

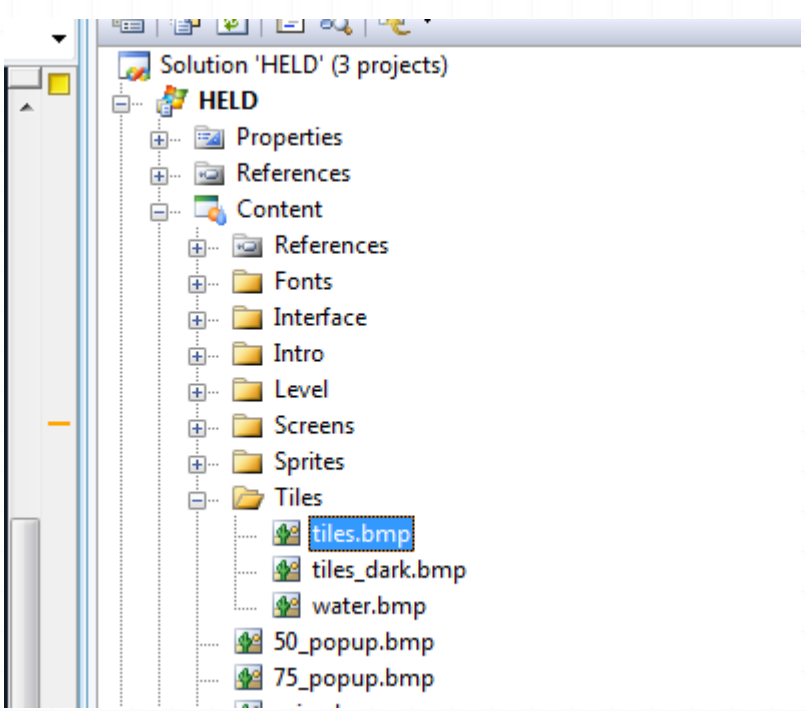
Content Pipeline: Flow

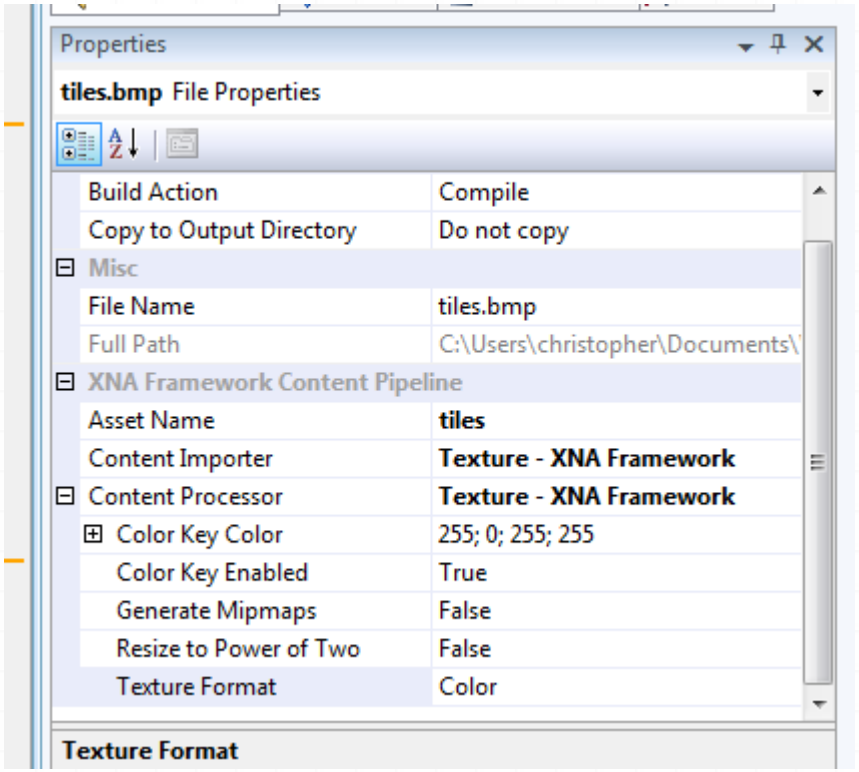


Vordefinierte Prozessoren

- Unterstütze Formate:
  - Grafiken (.bmp, .jpg, .png, .dds, .tga)
  - 3D Modelle (.x, .fbx, .obj)
  - Schriften (.spritefont)
  - Shader-Dateien (HLSL) (.fx)
  - Audio-Dateien (.wav, .wma, .mp3)
  - XACT-Soundprojekte (.xap)

Content Pipeline: Importer/Processor






Grafiken anzeigen

1. Inhalt laden

- Grafiken (Sprites) werden durch Klasse Texture2D repräsentiert

```
Texture2D mySprite =
   Content.Load<Texture2D>(“Held”);
```

Keine Dateiendung angeben!
Content in LoadContent laden

2. Anzeigen - SpriteBatch

- 2D Grafiken werden mittels **`SpriteBatch`** gezeichnet
- SpriteBatch kann 1 bis n Sprites mit gleichen Einstellungen zeichnen
- Umschlossen von `SpriteBatch.Begin()` und `SpriteBatch.End()`

2. Anzeigen – SpriteBatch.Draw

```
spriteBatch.Begin();

    spriteBatch.Draw( Texture2D, Vector2, Color );
                                …
    spriteBatch.Draw( Texture2D, Vector2, Color );

spriteBatch.End();
```

Aufgabe 1

- Sprite laden, anzeigen, in der Mitte des Bildschirms rotieren (um Mittelpunkt der Grafik) und Rot färben

1. XNA Gamestudio Projekt erstellen
2. Grafik in Projekt einfügen (Unterpunkt Content)
3. Laden mit Content.Load
4. Mittels SpriteBatch und Draw Funktion anzeigen
5. Rotation: *SpriteBatch.Draw ist mehrfach überladen*
6. Mitte des Bildschirms: *GraphicsDevice.Viewport*.....

Lösung – Aufgabe 1

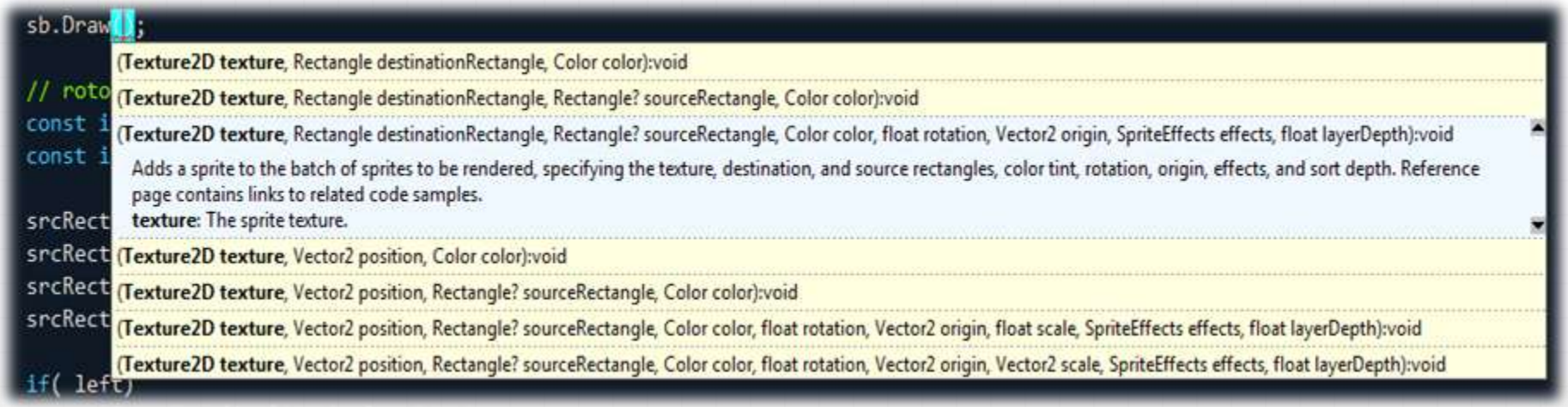
```
private Texture2D MySprite;
private float RotationAngle = 0.0f;


// Load our sprite through the content pipeline
MySprite = Content.Load<Texture2D>("Controller");


// Update rotation according to elapsed time
RotationAngle += (float)gameTime.ElapsedGameTime.TotalSeconds;
            RotationAngle %= MathHelper.Pi * 2.0f;
```

Lösung – Aufgabe 1

```
// Begin sprite drawing
spriteBatch.Begin();

// Position where the sprite should be displayed on the screen
Vector2 pos = new Vector2(GraphicsDevice.Viewport.Width / 2,
GraphicsDevice.Viewport.Height / 2);

// Center point of rotation
Vector2 origin = new Vector2(MySprite.Width / 2, MySprite.Height /
2);

// Draw the sprite
spriteBatch.Draw(MySprite, pos, null, Color.Red, RotationAngle,
origin, 1.0f, SpriteEffects.None,0f);

// End sprite drawing
spriteBatch.End();
```

SpriteBatch.Draw Funktion

- Ausschnitte
- Rotation
- Ursprung versetzen
- Skalierung
- Effekte (umdrehen der Grafik)
- Einfärben der Grafik
- Layer



Stapelverarbeitung von Sprites

- SpriteBatch.Begin()
  - Sprites anzeigen
- SpriteBatch.End()

- Begin (
  SpriteBlendMode blendMode,
  SpriteSortMode sortMode,
  SaveStateMode stateMode,
  Matrix transformMatrix
)

- Additive
- AlphaBlend
- None

- Deferred
- BackToFront
- FrontToBack
- Texture

- Immediate

- SaveState
- None

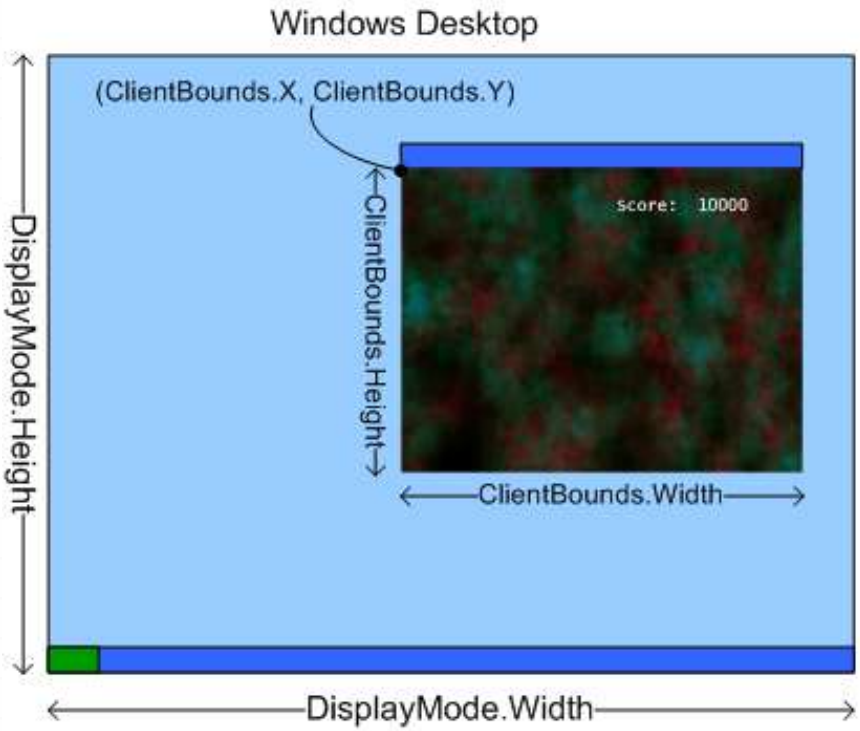
BackBuffer

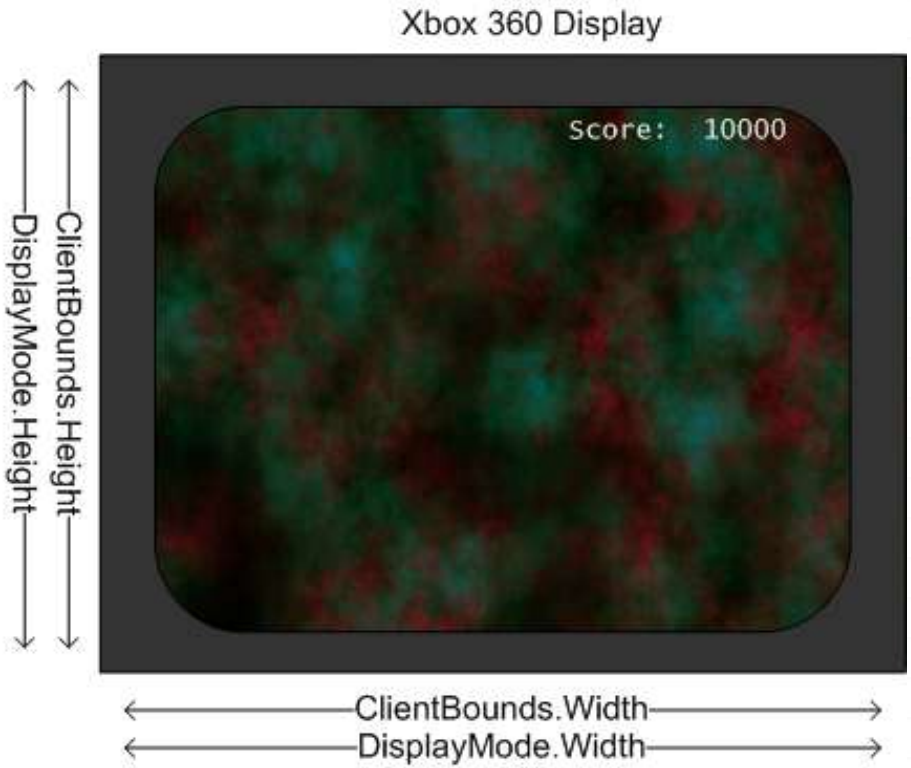
- Gezeichnet wird auf ein sog. RenderTarget
- RenderTarget ist ein Buffer in dem die Pixel eines Bildschirm gespeichert werden
- Default RenderTarget wird auch **BackBuffer** genannt


SAMSUNG


BackBuffer
FrontBuffer
SAMSUNG

Eigenschaften des Displays

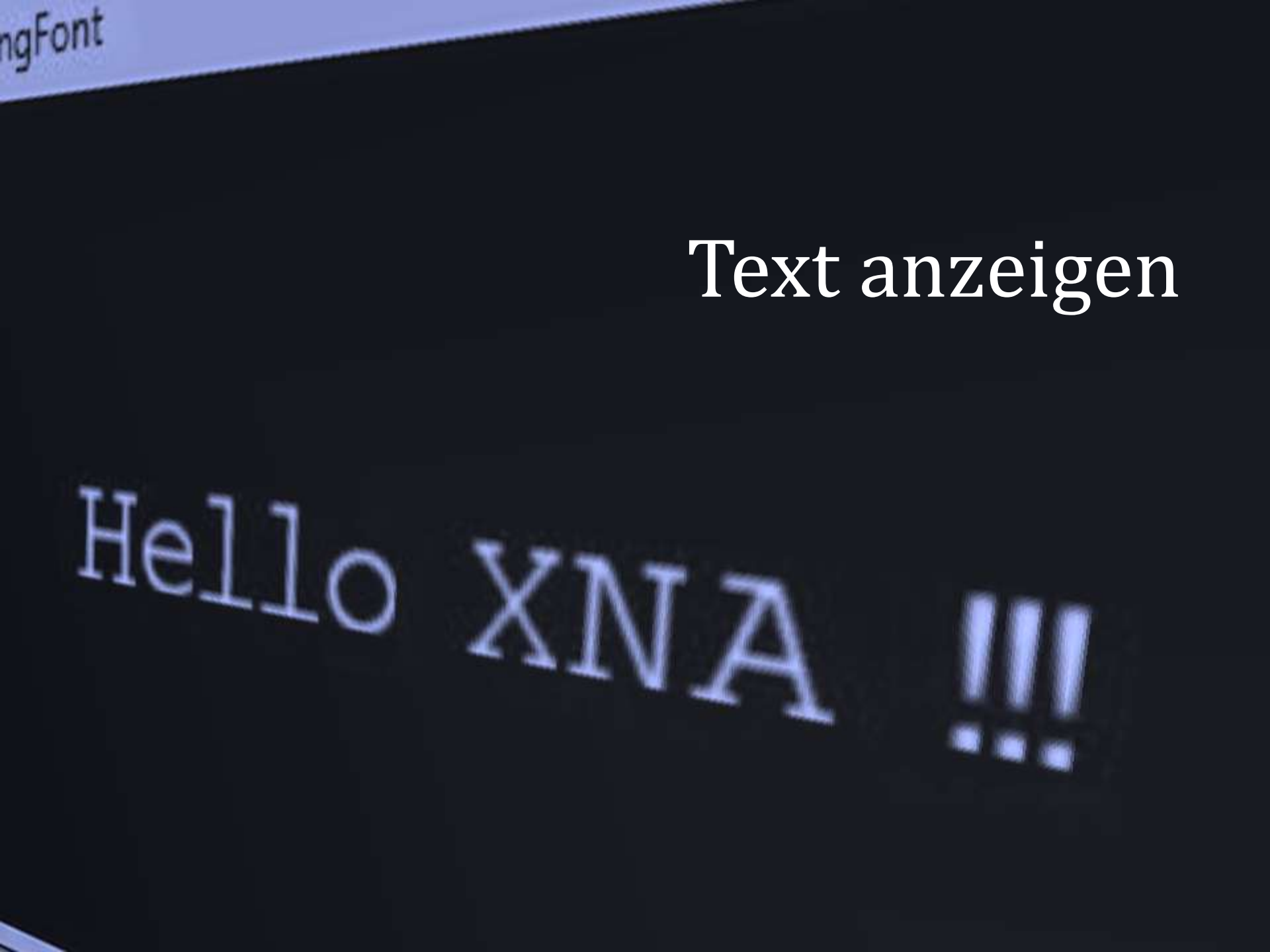




Display

- Auflösung ändern:
  - `graphicsDeviceManager.PreferredBackBufferWidth = 1280;`
  - `graphicsDeviceManager.PreferredBackBufferHeight = 720;`
- Vollbild
  - `graphicsDeviceManager.IsFullScreen = true;`
- Viewport
  - Teil eines RenderTargets, muss **nicht** gesamtes RenderTarget ausfüllen

Text anzeigen



Schrift anzeigen

- XNA verwendet normale Windows TrueType Fonts
- ContentPipeline erzeugt daraus Textur (SpriteFont)
  - Steuerung mittels XML-Datei
    - Font-Name
    - Font-Größe
    - Normal, Fett, Kursiv
    - Abstände
    - Zeichenbereiche

SpriteFont

- Laden

```
myFont = Content.Load<SpriteFont>(“XML-Datei”);
```

- Anzeigen

```
spriteBatch.DrawString(SpriteFont, String,
 Vector2, Color );
```

- `DrawString()` ist ähnlich überladen wie Draw

Nützliche Funktionen

- Höhe und Breite eines Strings abfragen
  `SpriteFont.MeasureString( String )`

- XNA Fonts
  - Kooten.ttf
  - Linds.ttf
  - Miramo.ttf
  - **Bold Miramob.ttf**
  - PERIC.TTF
  - PERICL.TTF
  - Pesca.ttf
  - **Pescab.ttf**

Eingabe

Eingabe

- XNA ermöglicht:
  - Mauseingaben — Klasse: `Mouse`
  - Tastatureingaben — Klasse: `Keyboard`
  - Xbox-Controller — Klasse: `Gamepad`
- Jeweils: `GetState()` Methode

Eingabe: Tastatur

- Tastenstatus abfragen

```
Keyboard.GetState().IsKeyDown(Keys.Up)
```

- Tasten**druck** abfragen

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

if( currentKBState.IsKeyUp(key) &&
  !previousKBState.IsKeyDown(key) ) { … }

previousKBState = currentKBState;
```

Eingabe: Xbox-Controller

- Thumbstick
  `GamePadState.ThumbSticks.Left / .Right`

- DPad / Button
  GamePadState.Buttons.DPadLeft …
  GamePadState.Buttons.A …

- Vibration
  `GamePad.SetVibration(…)`

Aufgabe 2

- Sprite mit Tastatur/Maus bewegen
- Position des Sprites auf Bildschirm anzeigen

SpriteFont

```
myFont = Content.Load<SpriteFont>(“Datei”);

spriteBatch.DrawString(myFont, … )
```

Lösung – Aufgabe 2

```
 // Get new keyboard state
KeyboardState keyboardState = Keyboard.GetState();

// Calculate the amount of moving with the elapsed time since the last frame, this
gives us framerate independent movement
float movement = (float)gameTime.ElapsedGameTime.TotalMilliseconds * 0.5f;

// Left/Right
if (keyboardState.IsKeyDown(Keys.Left)) {
      spritePosition.X -= movement;
}
if (keyboardState.IsKeyDown(Keys.Right)) {
      spritePosition.X += movement;
}
// Up/Down
if (keyboardState.IsKeyDown(Keys.Up)) {
      spritePosition.Y -= movement;
}
if (keyboardState.IsKeyDown(Keys.Down)) {
      spritePosition.Y += movement;
}
```

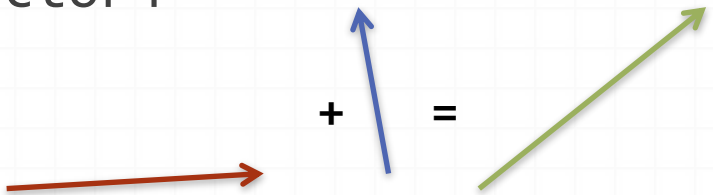
Lösung – Aufgabe 2

```
// Draw the sprite (White as color to disable any color effects)
spriteBatch.Draw(MySprite, spritePosition, Color.White);

// Text to display
string displayText = "Position: " + spritePosition.X + " : " +
     spritePosition.Y;

// Display text in upper left corner of viewport
Vector2 displayPosition = new Vector2(10, 10);

// Display text using our loaded Font and in red color
spriteBatch.DrawString(MyFont, displayText, displayPosition, Color.Red);
```

Kollisionsbehandlung

Mathematische Funktionen: Vektoren

- Klassen `Vector2`, `Vector3` und `Vector4`
  - Grundrechnenarten
    - Add(), Multiply(), etc.



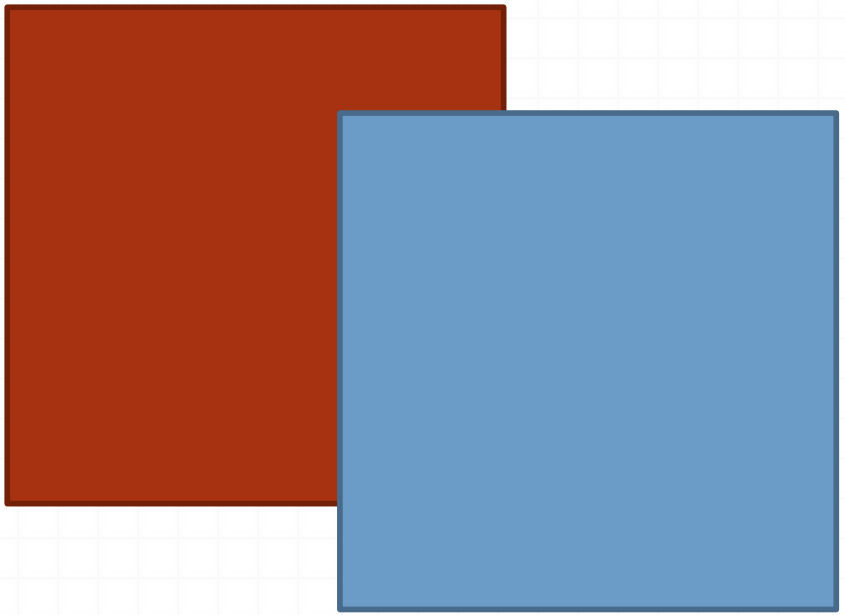
  - Vector2.Distance()
    - Abstand zwischen zwei Vektoren
  - Vector2.Length()
    - Länge eines Vektors
  - Vector2.Reflect()
    - Veränderter Vektor nach einer Kollision

Matrizen

- Klasse `Matrix` für Transformation
- Wird insbesondere bei 3D verwendet
- Hilfsfunktionen:
  - `Matrix.CreateRotation(...)`
  - `Matrix.CreateTranslation(...)`
  - …

Rectangles

```
Rectangle rc  = new Rectangle(X, Y, Width, Height);
Rectangle rc2 = new Rectangle(X2, Y2, Width, Height);

If( rc.Intersects(rc2) )
{
  // BAAAM!
}
```



Aufgabe 3

1. Kollisionen mit einem anderen Sprite im 2D-Raum feststellen und diese behandeln

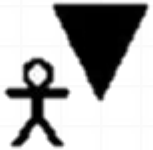
2. Kollisionserkennung implementieren, die verhindert, dass ein Sprite den sichtbaren Teil des Bildschirmes verläßt

Lösung – Aufgabe 3

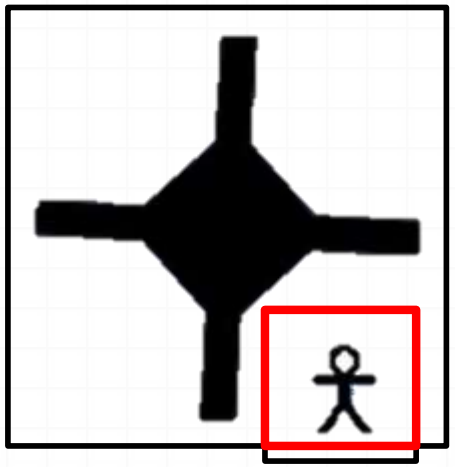
```
// zwei Rectangles anlegen
Rectangle rc = new Rectangle(…);
Rectangle rc2 = new Rectangle(…);

// Kollision ueberpruefen
if( rc.Intersects(rc2) ) {
  // Explosion
}

// Bildschirmrand
if(  position.X < 0
  || position.X > graphics.GraphicsDevice.Viewport.Width
  || position.Y < 0
  || position.Y > graphics.GraphicsDevice.Viewport.Height ) {
  // Sprite ausserhalb des Bildschirms
}
```

Nachtrag Aufgabe 3

- Pixel-Genaue Kollision zwischen Sprites



- Kollision zwischen gedrehten Rectangles



- Collision Series auf creators.xna.com

Sound & Musik

Sound

- Wiedergabe von Sound-Dateien (wav, mp3, wma)
- SoundEffect- und Song-Klasse
- Abspielen mittels Play() Methode
- Werden mittels der Content Pipeline geladen

Sound

- SoundEffect-Klasse bietet zusätzliche 3D-Sound Funktion (Play3D)
  - Sounds im 3D-Raum positionieren
  - Sounds pitchen
  - Sounds loopen

Musik

- MediaPlayer-Klasse Funktionen zum:
  - Abspielen, Stoppen, Resumen von Songs
- Abspielen einer Datei
  - MediaPlayer.Play(SongObject);
- Musik-Dateien werden in einer Endlosschleife abgespielt
- Kann durch eigene Musik von der Xbox360 „ersetzt“ werden

Aufgabe 4

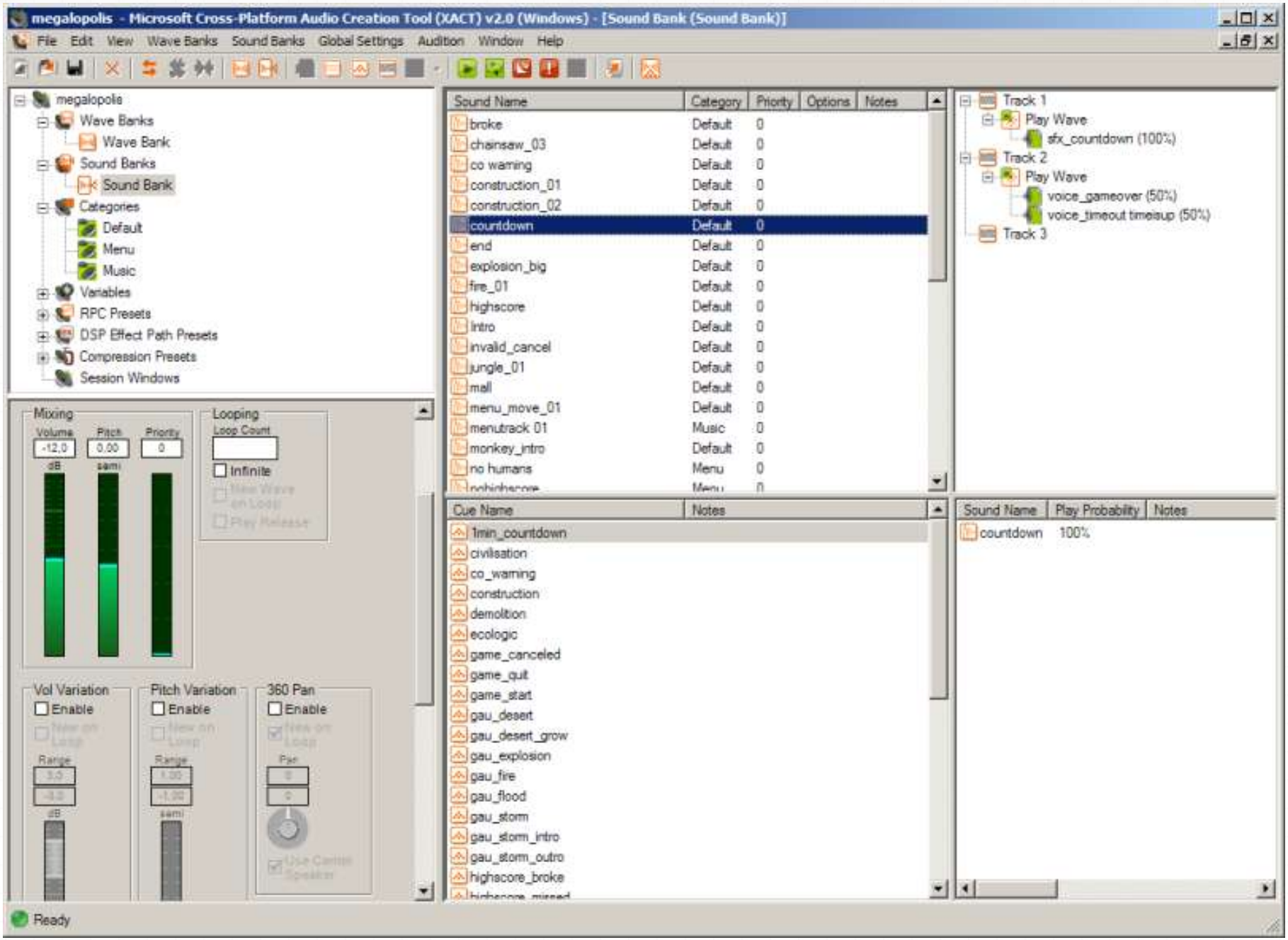
- Wiedergabe von Sounds
  - Bei Bewegung eines Objektes
  - Kollision eines Objektes
- Wiedergabe von Musik
  - Permanente Hintergrund-Musik

Lösung – Aufgabe 4

```
soundEffect = Content.Load<SoundEffect>("thunder");

KeyboardState keyboardState = Keyboard.GetState();
if( keyboardState.IsKeyDown( Keys.Space ) &&
lastKeyState.IsKeyUp( Keys.Space ) )
{
   soundEffect.Play();
}

soundSong = Content.Load<Song>("maid");

MediaPlayer.Play( soundSong );
```

Alternative XACT

- Cross-platform Audio Creation Tool
- Tool zum Erzeugen von Sound-Cues
- Erlaubt das Erstellen einer Vielzahl von Effekten und Sound-Kombinationen
- Erstellte Cues werden im Quelltext nur noch abgespielt, keine weitere Programmierung nötig


megalopolis - Microsoft Cross-Platform Audio Creation Tool (XACT) v2.0 (Windows) - [Sound Bank (Sound Bank)]
File Edit View Wave Banks Sound Banks Global Settings Audition Window Help
megalopolis
Wave Banks
Wave Bank
Sound Banks
Sound Bank
Categories
Default
Menu
Music
Variables
RPC Presets
DSP Effect Path Presets
Compression Presets
Session Windows
Sound Name Category Priority Options Notes
broke Default 0
chainsaw_03 Default 0
co warning Default 0
construction_01 Default 0
construction_02 Default 0
countdown Default 0
end Default 0
explosion_big Default 0
fire_01 Default 0
highscore Default 0
intro Default 0
invalid_cancel Default 0
jungle_01 Default 0
mall Default 0
menu_move_01 Default 0
menutrack 01 Music 0
monkey_intro Default 0
no humans Menu 0
Track 1
Play Wave
sfx_countdown (100%)
Track 2
Play Wave
voice_gameover (50%)
voice_timeout timeiup (50%)
Track 3
Mixing
Volume Pitch Priority
-12.0 0.00 0
dB semi
Looping
Loop Count
Infinite
Vol Variation
Enable
Pitch Variation
Enable
360 Pan
Enable
Cue Name Notes
1min_countdown
civilisation
co_warning
construction
demolition
ecologic
game_canceled
game_quit
game_start
gau_desert
gau_desert_grow
gau_explosion
gau_fire
gau_flood
gau_storm
gau_storm_intro
gau_storm_outro
highscore_broke
Sound Name Play Probability Notes
countdown 100%
Ready

Animationen

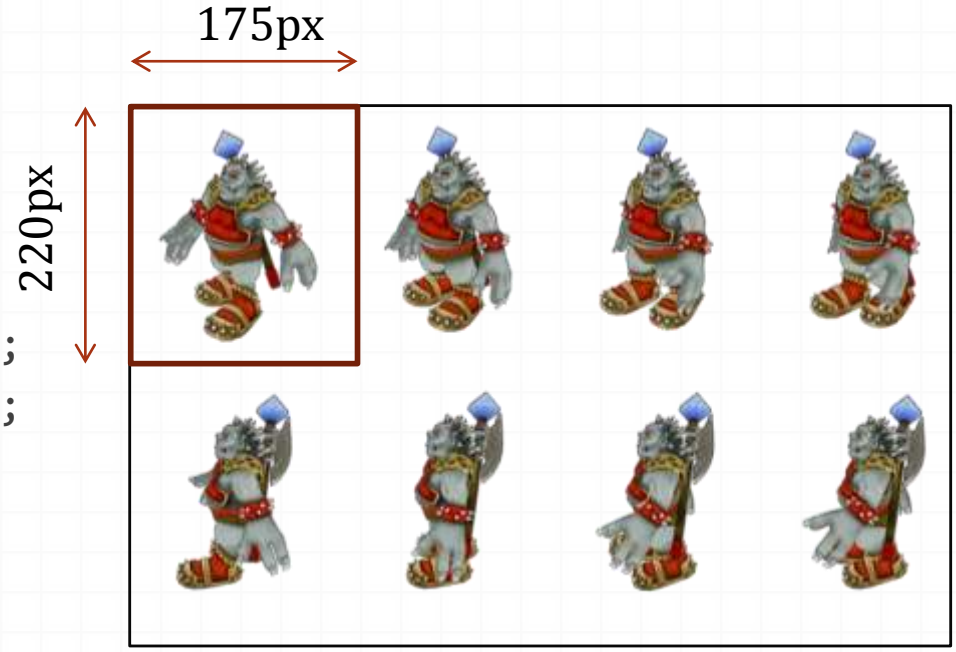
Animationen

- Charakteranimation typischerweise:
  viele einzelne Frames
- Möglichkeit 1: `Texure2D` pro Frame
- Sinnvoller:
  - Große Textur mit allen Frames
  - Auswahl über `Rectangle`

Animationsparameter

- Animations-Geschwindigkeit
- Anzahl der Frames
- Größe eines Frames

```
Rectangle rc = new Rectangle();
rc.Width = 175;
rc.Height = 220;

rc.X = (currentFrame % 4) * 175;
cc.Y = (currentFrame / 4) * 220;
```



Aufgabe 5 – Animation

1. Sprite „bwr.png“ laden
   - Breite: 175px
   - Höhe: 220px
   - 1 Animationsrichtung pro Zeile
   - 6 Frames pro Animation

2. Lauf-Animation anzeigen

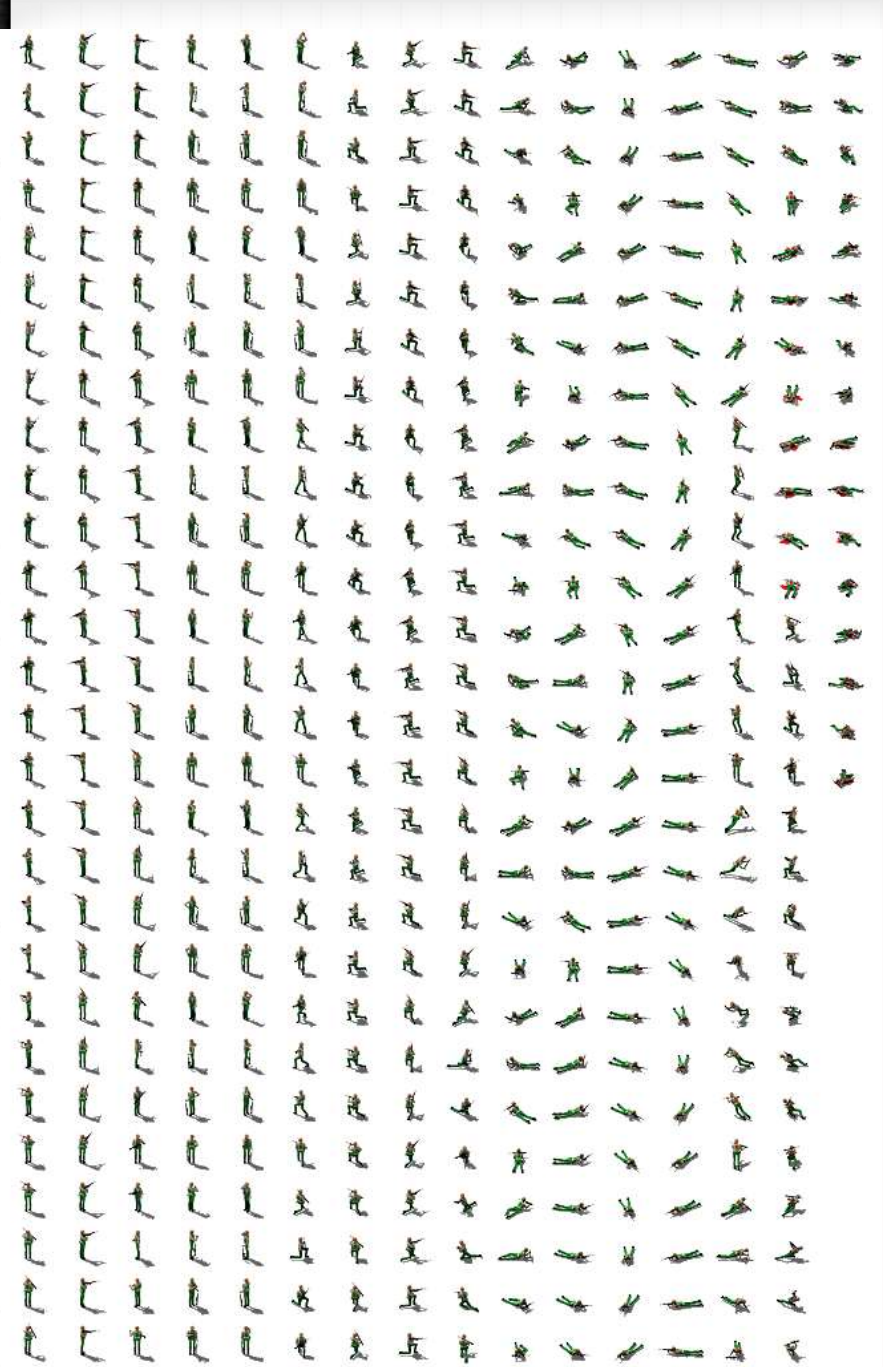
Lösung – Aufgabe 5

```
const int SpriteFrameWidth = 175;
const int SpriteFrameHeight = 220;
const int AnimationSpeed = 75;
const int AnimationNumFrames = 6;

int animationTimer;
int animationFrame;

// Update
int elapsed = gameTime.ElapsedGameTime.Milliseconds;
animationTimer += elapsed;
while( animationTimer > AnimationSpeed ) {
    animationTimer -= AnimationSpeed;
    animationFrame = (animationFrame + 1) % AnimationNumFrames;
}
```

Animationsparameter

Lösung – Aufgabe 5

```
// Draw
Rectangle rect = new Rectangle();

rect.Width = SpriteFrameWidth;
rect.Height = SpriteFrameHeight;

rect.X = animationFrame * SpriteFrameWidth;
rect.Y = 1 * SpriteFrameHeight;

spriteBatch.Draw( texture, new Vector2( 0, 0 ), rect,
    Color.White );
```



```
// animationsdatei für bunkermenschen soldat gewehr
// christopher schleiden 2009
// ---------------------------------------------------------------

// anzahl frames gesamt
NUMFRAMES 70

// able_to == ALL, MOVE, ATTACK, SHOOT, DIE, NULL

// ---------------------------------------------------------------------------------------------
----------
//n             name                          sta             num             req
                go2             able_to       delay                           stat
                callname
// ---------------------------------------------------------------------------------------------
----------
// AUFRECHT BEWEGEN
// ---------------------------------------------------------------------------------------------
----------
0               steht                         0               8               -1
                0               NULL          100                             STAN
                DEFAULT
1               begin_1                       8               8               -1
                2               MOVE          100                             STAN
                MOVE_NORMAL_START
2               begin_2                       16              8               1
                3               MOVE          100                             STAN
3               laufen_1        24            8               2               4
                MOVE            100                           STAND
                MOVE_NORMAL_RUN
4               laufen_2        32            8               3               5
                MOVE            100                           STAND
5               laufen_3        40            8               4               6
                MOVE            100                           STAND
6               laufen_4        48            8               5               7
                MOVE            100                           STAND
7               laufen_5        56            8               6               8
                MOVE            100                           STAND
8               laufen_6        64            8               7               3
                MOVE            100                           STAND
9               ende_1                        72              8               -1
                10              MOVE          100                             STAN
                MOVE_NORMAL_END
10              ende_2                        80              8               9
                0               MOVE          100                             STAN
// ---------------------------------------------------------------------------------------------
----------
// AUFRECHT STERBEN (HELDENTOD ;)
// ---------------------------------------------------------------------------------------------
----------
11              sterben_steht_1 456           8               0               12
                NULL            250                           DIE             DIE_
12              sterben_steht_2 464           8               11              13
                NULL            250                           DIE
13              sterben_steht_3 472           8               12              14
```

Game States

Spiel

Intro

Gameover

```
void Draw(…)
{
    if( isIntro ) {
        // draw Intro
    }

    if( !isIntro &&
        !isGameover ) {
        // draw game
    }

    if( isGameover ) {
        // draw game
        // over screen
    }
}
```

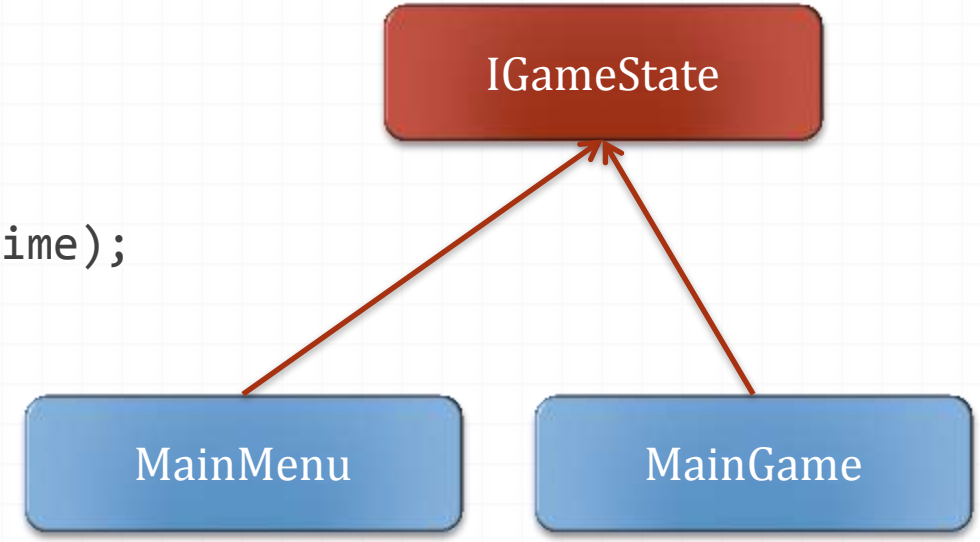
Was passiert,
wenn unser Spiel
komplexer wird?

GameStates

- Jeder *Teil* eines Spiels wird als extra GameState implementiert
- Gamestates erben von gemeinsamer Basisklasse
- Verwaltung durch Gamestate Manager

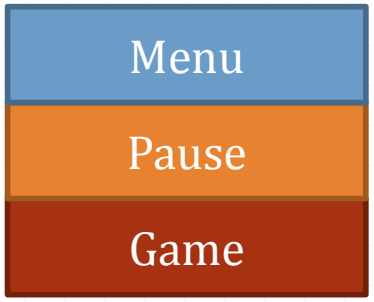
IGameState

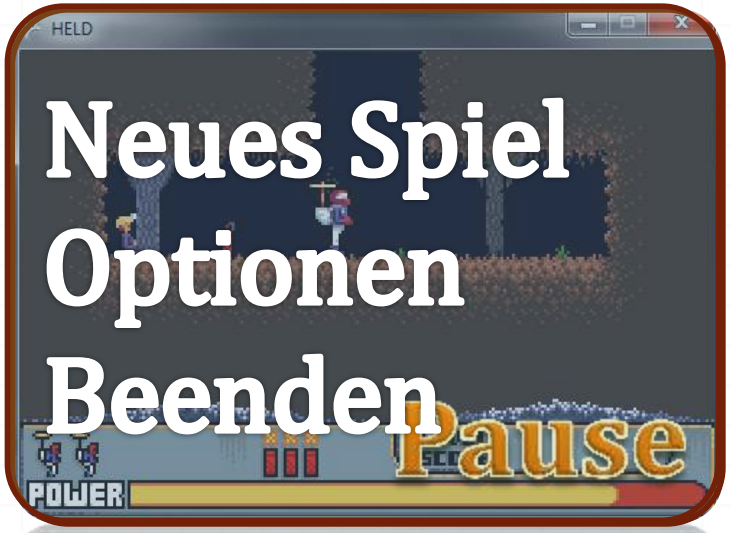
```
internal interface IGameState
{
  void LoadContent();

  void Update(GameTime gameTime);

  void HandleInput();

  void Draw(GameTime gameTime);
}
```



Verwaltung durch GSManager

- Zuständig für Erzeugen und Verwalten der Gamestates
- Gamestates liegen auf Stack
- Eingaben verarbeitet jeweils der oberste GameState
- Zeichenreihenfolge beachten!





Tilemap

```
#  - Kerkerwände
__ - Kerkertür
|  - Zellentür
S  - Schatz
%  - Wächter
@  - Held
```

```
####################
#                  #   S      #
#                  #          #
#                  #          #
#                % |          #
#                  ##########
#                  #          #
#                  #          #
#                  |          #
##_ _################
```


TileMapSample

```
private const int map_hoehe = 10;
private const int map_breite = 20;

private static char[,] tilemap = new char[map_hoehe, map_breite]
{
  { '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' },
  { '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', 'S', ' ', ' ', ' ', ' ', '#' },
  { '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#' },
  { '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#' },
  { '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '%', '|', ' ', ' ', ' ', ' ', ' ', ' ', '#' },
  { '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', '#', '#', '#', '#', '#', '#', '#', '#' },
  { '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#' },
  { '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#' },
  { '#', ' ', '@', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#' },
  { '#', '#', '_', '_', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' }
};
```

```
protected override void Draw(GameTime gameTime)
{
    graphics.GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin();

    // alle Zeilen durchlaufen
    for (int y = 0; y < map_hoehe; y++)
    {
        // alle Spalten durchlaufen
        for (int x = 0; x < map_breite; x++)
        {
            int frame = 0;
            // je nach ASCII-Zeichen in der Map, eine andere Textur zeichnen
            switch (tilemap[x,y])
            {
                case '#':   // Wand
                    // Frame fuer Wand setzen
                    frame = 12;
                    break;
                […]
            }
            Rectangle rc = […]; // Rectangle anhand von ‚frame‘ berechnen
            spriteBatch.Draw(TileTexture, new Vector2(tile_breite * x, tile_hoehe * y), rc,
                 Color.White);
        }
    }
    spriteBatch.End();
    base.Draw(gameTime);
}
```

Quellen

- Folien basieren teilweise auf Originalen von Ingo Köster
- Kollisionsbild ist CC von FlickR

Ressourcen - Grafiken

- http://reinerstileset.4players.de/
- http://www.ambrosine.com/resource.php#Graphics,%20Music,%20and%20Other%20Resources
- http://www.flyingyogi.com/fun/spritelib.html
- http://www.spriteworks.com/sworks.html

Ressourcen - XNA

- http://creators.xna.com/
- http://www.dreambuildplay.com/main/default.aspx

Danke für Eure Aufmerksamkeit!

Und jetzt…?

Wettbewerb!

Um XX:XX kürt die fachkundige Jury das beste/interessanteste/spielbarste Spiel!