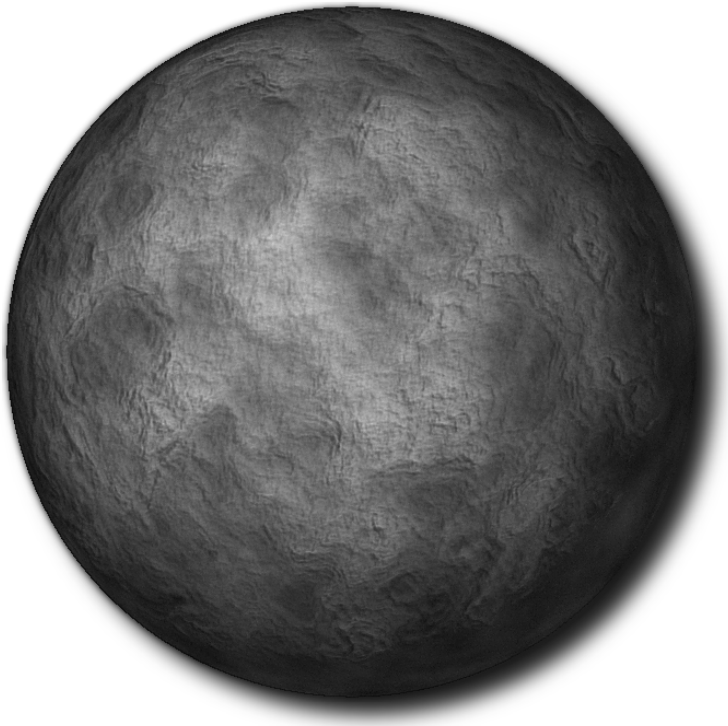


Franz-Haniel-Gymnasium

Boulderdash

Implementation eines Geschicklichkeitsspiels

Autor:
Matthias Linder

Kurs:
Informatik GK 12

Datum:
25. Februar 2009

Lehrer:
M. Tenhaven

Inhaltsverzeichnis

1 Einleitung

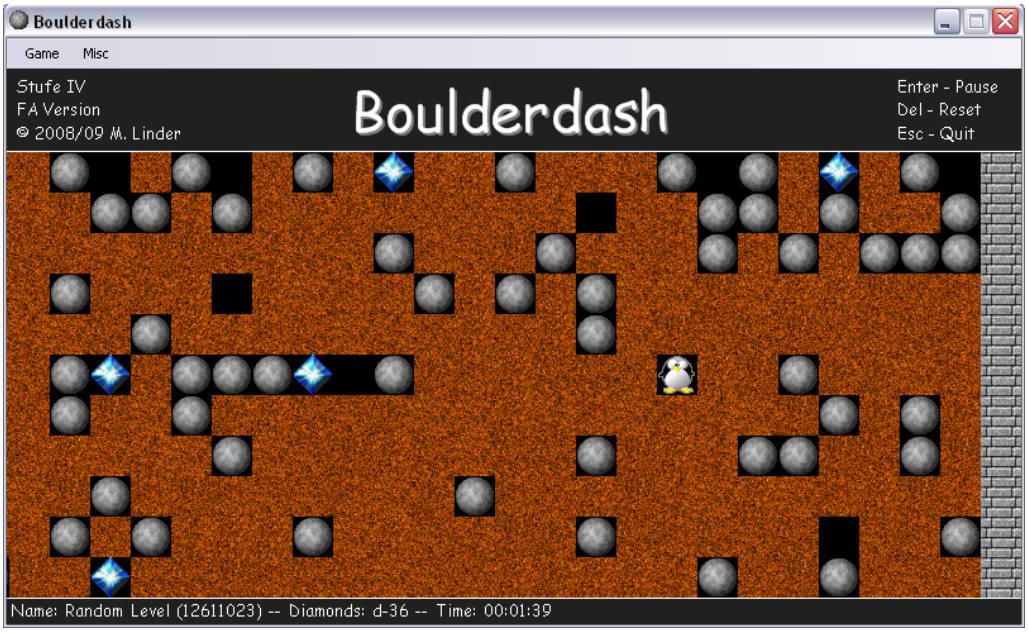


Abbildung 1: Implementation des Spiels „Boulderdash“

In der heutigen Zeit spielt die Informationstechnik eine wichtige Rolle. Viele Firmen, Organisationen und Medien verwenden Computer und Server – und damit die Informationstechnik – um ihre Präsenz im Internet und ihre Produktivität zu steigern. Doch auch im Haushalt hat sich der Computer durchgesetzt: Nahezu jeder Mensch hat heutzutage seinen eigenen „Home-Computer“, welchen er bei allen alltäglichen Dingen nutzt. Sei es dabei für die Arbeit, Musik, Nachrichten oder auch einfach nur zum Spielen.

Deshalb werde ich die Facharbeit der Implementierung des Geschicklichkeitsspiels „Boulderdash“ widmen, welches sich aufgrund seines simplen, aber fesselnden Spielprinzips und seiner Bekanntheit dafür qualifiziert. Dabei werde ich zunächst das Spielprinzip und einige Spielelemente vorstellen, um anschließend meine Umsetzung anhand des Quellcodes und verwendeter Algorithmen aufzeigen.

2 Das Genre „Boulderdash“

Der Begriff „Boulderdash“ steht nicht nur für ein einzelnes Spiel, sondern schon fast für ein ganzes Genre, welches inzwischen eine Reihe von einzelnen Boulderdash-Klons umfasst. Auch wenn sich die Spiele durch ihre Level und Spielelemente unterscheiden, so hat sich an dem Spielprinzip seit dem ersten Boulderdash-Spiel nichts mehr geändert.

Angefangen hat dieses Genre in 1984 mit dem ersten Boulderdash[1], programmiert von Peter Liepa (First Star Software), welches zunächst auf den zu dieser Zeit typischen Computern wie Amiga, C64 und Atari erschienen ist. Das erste Boulderdash enthielt die Hauptspielelemente, die ich später vorstellen werde, sowie einige zusätzliche Objekte wie z. B. Amöben. Später wurden weitere Fortsetzungen für den PC, kleinere Spielkonsolen und Handhelds veröffentlicht.

Dennoch hört die Boulderdash Reihe hier nicht auf. Durch die Verbreitung des Internets traten immer neue Boulderdash-Klons[2] in die Öffentlichkeit – und so findet man heutzutage sehr viele Spiele des Genres kostenlos im Internet.

2.1 Spielprinzip

Das Spielprinzip von Boulderdash ist denkbar einfach gehalten, und hat wahrscheinlich auch zu dem Erfolg des Genres beigetragen.

In dem Spiel steuert man eine Spielfigur mit der Tastatur durch verschiedene, zweidimensionale Höhlen, und muss alle im Level versteckten Diamanten aufsammeln. Dabei gilt es Rätsel durch Verschieben von Steinen und Freigraben von Sand zu lösen, und währenddessen herunterfallenden Steinen ausweichen.

Hat der Spieler alle notwendigen Diamanten eingesammelt, so kommt er in das nächste Level. Fällt jedoch ein Stein auf die Spielfigur, so stirbt diese und man muss das aktuelle Level wiederholen.

Je nach Spiel variieren einige Elemente des Spiels; so muss man im klassischem Boulderdash zum Beispiel zum Vollenden eines Levels nach dem Einsammeln aller Diamanten den Levelausgang suchen. In anderen Klons hingegen muss man die Level unter Zeitdruck spielen, so dass man nicht soviel Zeit zum Überlegen hat.

[1] vgl. Quellen [1]-[3]
[2] z. B. Rocks’n’Diamonds (s. Anhang)

2.2 Spielelemente

Jedes Boulderdash hat unterschiedliche Spielelemente. Doch alle Spiele dieses Genres haben die identischen[3] Hauptspielelemente, welche das Spiel erst zu einem Boulderdash machen:

Die Spielfigur

Bei „Rockford“[4] handelt es sich um den Spielcharakter, den man in Boulderdash steuern kann. Dabei kann sich die Figur durch Betätigen der Pfeiltasten auf der Tastatur in die vier verschiedenen Himmelsrichtungen bewegen; die diagonalen Nebenshimmelrichtungen sind dabei ausgenommen. Fällt ein Stein auf die Spielfigur, so stirbt diese und man muss das Level wiederholen.

Sand

Sand dient in Boulderdash als ein Element, welches Steine daran hindert herunterzufallen und Gegner blockiert. Er kann durch die Spielfigur durch einfaches „Durchlaufen“ entfernt werden.

Diamanten

Die Diamanten sind mitunter die wichtigsten Spielelemente in Boulderdash. Die Spielfigur muss (fast) alle Diamanten durch „Durchlaufen“ aufsammeln um das Level zu gewinnen. Sie werden oft in Steinansammlungen bzw. an schwer zu erreichende Stellen gelegt um den Schwierigkeitsgrad des Spiels zu erhöhen. Diamanten unterliegen der gleichen Spielphysik wie Steine.

Steine

Das zweit-wichtigste Spielelement ist der Stein. Steine hindern den Spieler bei seiner Passage durch das Level, unterliegen aber im Gegensatz zu festen Wänden der Schwerkraft und fallen daher nach unten, wenn sie nicht durch Sand oder andere Spielelemente begrenzt werden. Landen dabei zwei Steine aufeinander, so kann der obere Stein nach links oder rechts abrutschen (Lawinen), wenn sich dort kein anderes Spielelement befindet.

[3] Die dargestellten Grafiken sind der eigenen Implementation entnommen
[4] Hier „TuX“(Linuxmasköttchen)

2.3 Zusätzliche Spielelemente

Neben den primären Spielelementen gibt es in den meisten Boulderdash-Varianten noch weitere Spielelemente, welche sich je nach Spiel unterscheiden. Einige der möglichen Spielelemente – die ich in meiner eigentlichen Implementierung zwar umgesetzt habe, aber nur bei den Erweiterungsmöglichkeiten kurz erläutern werde – stelle ich nun dar:

Wände

Wände repräsentieren Hindernisse, welche alle Elemente blockieren und nicht entfernt werden können (eine Ausnahme bilden hier zerstörbare Wände, welche durch Dynamit zerstört werden können). Sie dienen vor allem als Levelgrenze, oder auch zum Einteilen eines Levels in verschiedene Gebiete.

Dynamit

Dynamit muss zunächst vom Spieler als Gegenstand eingesammelt werden. Besitzt der Spieler nun Dynamit, so kann er dieses durch Tastendruck an einer beliebigen Stelle ablegen. Nach einiger Zeit explodiert es, und kann so Hindernisse (wie z. B. Steine) aus dem Weg räumen, so dass durch ein Missgeschick verschüttete Diamanten eingesammelt werden können.

Gegner

Ein typisches weiteres Spielelement sind Gegner. Gegner sind Lebewesen, welche sich häufig durch freigelegte Tunnel bewegen und den Spieler töten, sobald sie diesen berühren. Dabei weist jeder Gegner oft seine eigenen Charakteristika und Bewegungsmuster auf. Gegner können wie der Spieler durch Steinschlag und Dynamit getötet werden.

Explosive Steine

Explosive Steine[5] verhalten sich prinzipiell so wie normale Steine; doch lösen sie eine Explosion aus, wenn ein Stein auf sie fällt, oder sie selber auf etwas anderes fallen. Deshalb ist Vorsicht im Umgang mit ihnen geboten. Im Gegensatz zu Dynamit stellen sie ein Hindernis dar und können nicht aufgesammelt werden.

[5] Im Spiel: „Flints“(Feuersteine)

3 Implementation des Spiels

Bei der Implementierung gilt es zunächst eine geeignete Programmiersprache und Entwicklungsumgebung zu wählen, welche zum einen einem objektorientierten Muster folgt, und zum anderen auch ein Potential für die Zukunft hat. Deshalb habe ich mich für die Programmiersprache C# (C-Sharp) entschieden, welche 2001 von Microsoft veröffentlicht wurde[6]. Die Sprache baut auf einen C-ähnlichen Syntax auf und vereinigt die Funktionen einiger anderer Programmiersprachen wie C++, Java und Delphi. In der Umsetzung verwende ich dabei die Version 2.0 des .NET Frameworks (Ansammlung von diversen Basisfunktionen) und den .NET 3.5 Compiler, welcher die Verwendung einiger neuerer Sprachkonstruktionen ermöglicht.

Da es sich bei C# um eine pseudo-interpretierte[7] Sprache handelt, wird zur Ausführung des Spiels das .NET Framework 2.0 benötigt.

3.1 Programmkonzept

Bei meiner Umsetzung habe ich mich für ein Konzept entschieden, welches auf einer strengen Trennung der diversen Programmfunktionen in einzelne Teilkomponenten aufbaut, und so einem leichten Erweiterungsverhalten am nächsten kommt. Insgesamt gibt es daher drei verschiedene Programmebenen:

1. Der *Programmkern* (Core), welcher die grundlegenden Programmfunktionen wie Grafik und Tastatur zur Verfügung stellt. Hier wird die Verbindung vom Spiel zu den Komponenten von Windows gelegt, so dass der Spielkern sich auf die Spiellogik begrenzen kann.

2. Der *Spielkern* (Game), welche das Spielverhalten und die Spieldarstellung übernimmt. Hier findet man den Levelaufbau und die Spielelemente, sowie das Verhalten und die Darstellungsmethoden derselben.

3. Die *Ausführungsebene* (Boulderdash), welche den Programmkern und das Spiel selber beim Programmstart initialisiert, und die äußere Steuerung des Spieles (Darstellungsfenster; Form) übernimmt.

[6] vgl. Quellen [4] und [5]

[7] C# Programmcode wird nur in eine Zwischensprache übersetzt, welche dann beim Programmstart in die ausführbare Version kompiliert wird.

Die Kommunikation zwischen den einzelnen Ebenen lässt sich dabei gut als Diagramm darstellen:

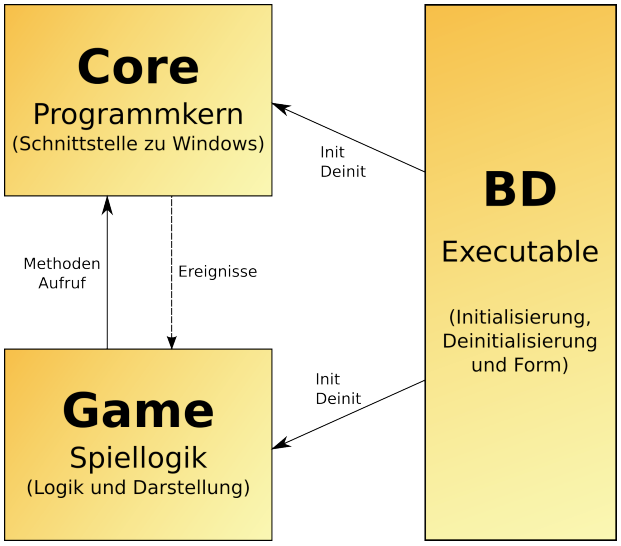


Abbildung 2: Aufbau und Einteilung des Programms

3.2 Umsetzung der Ausführungsebene

Der Aufbau der Ausführungsebene, welche im Endeffekt die ausführbare Datei bildet, lässt sich einfach beschreiben: Sie muss nur die zwei anderen Komponenten (Kern und Spiellogik) initialisieren, und beim Beenden des Programms diese wieder deinitialisieren. Insgesamt durchläuft die Ausführungsebene folgende Phasen:

1. **Initialisierung** des Programmkerns (Form wird erstellt).
2. **Initialisierung** der Spiellogik (Level wird erstellt/geladen).

3. **Wartezustand**. Warten, bis entweder der Programmkern (Form) oder die Spiellogik (Quit-Button) eine Abbruchbedingung liefert. Wenn dies geschieht wird in die nächste Phase gesprungen.

4. **Deinitialisierung** der Spiellogik.
5. **Deinitialisierung** des Programmkerns (Freigabe von verwendeten Resourcen).

3.3 Umsetzung des Programmkerns

Der Programmkern ist – wie vorher erwähnt – die Schnittstelle zwischen dem Spiel und Windows. Er stellt also zum einen die grundlegenden Funktionen eines Spiels, wie z. B. den Zeitgeber und das Threadmodell, und zum anderen sorgt er dafür, dass die Spiellogik selber ohne großen Aufwand den Spielinhalt darstellen kann.

Um moderneren Computern gerecht zu werden, habe ich mich im Verlauf des Projektes entschieden, die Umsetzung von Boulderdash auf einer *Multithreading*-Basis zu machen. Multithreading bedeutet, dass der Programmcode nicht sequentiell, sondern parallel abgearbeitet werden kann. Dies ist vor allem bei Mehrkernprozessoren effektiv, da diese sonst nur einen einzelnen Kern mit dem ganzen Programm beauftragen würden. Ich habe deshalb die zwei großen Anforderungsgebiete der Anwendung – Grafik und Spiellogik – auf zwei separate Threads gelegt, welche wie ein Zahnrad ineinander greifen und so die Aufgaben gleichzeitig erledigen.

3.3.1 Hauptkomponenten des Programmkerns

Anschließend möchte ich die drei wichtigsten Komponenten des Programmkerns beschreiben:

1. Der *Zeitgeber* dient dazu, dass sowohl auf langsamen, als auch auf schnellen Computern das Spiel immer mit der gleichen Geschwindigkeit abläuft. Außerdem werden die Threads passend pausiert, um eine hohe CPU-Last zu vermeiden.

2. Die *grafische Darstellung* implementiert „Doublebuffering“ um ein Flimmern zu vermeiden, welches auftreten würde, wenn die Zeichenfläche während des Zeichnen erneuert würde. Beim „DoubleBuffering“ wird der Spielinhalt zunächst auf eine verdeckte Zeichenfläche gezeichnet, und anschließend als Ganzes auf die Darstellungsfläche übertragen, so dass immer ein vollständiges Bild sichtbar ist.

3. Die *Tastatursteuerung* fängt Tastaturereignisse (Taste drücken, Taste loslassen, . . .) ab und leitet sie an entsprechende Events weiter, welche von der Spiellogik genutzt werden können.

Um die Wichtigkeit des Zeitgebers zu anzuzeigen, stelle ich nun einmal den Verlauf der Anwendung ohne, und einmal mit Zeitgeber auf jeweils einem schnellen und einem langsamen PC dar:

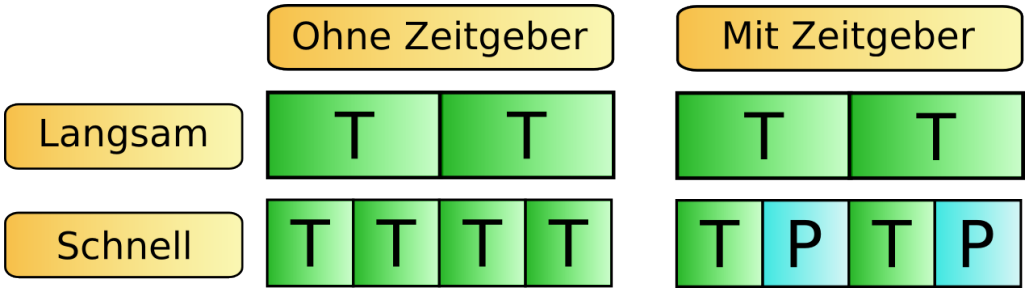


Abbildung 3: Tickverlauf auf verschiedenen PCs (T=Tick, P=Pause)

3.3.2 Darstellungskonzepte

Damit die Spiellogik die Tiles möglichst einfach und schnell darstellen kann, habe ich ein Tileset-System verwendet. Dabei werden mehrere Tiles innerhalb einer einzelnen Grafik gespeichert, so dass wenig Kontextwechsel notwendig sind und nicht erst aufwändig mehrere Texturen geladen werden müssen.

In den unten abgebildeten Grafiken sind nun die zwei wichtigsten genutzten Grafikkonzepte einmal dargestellt:

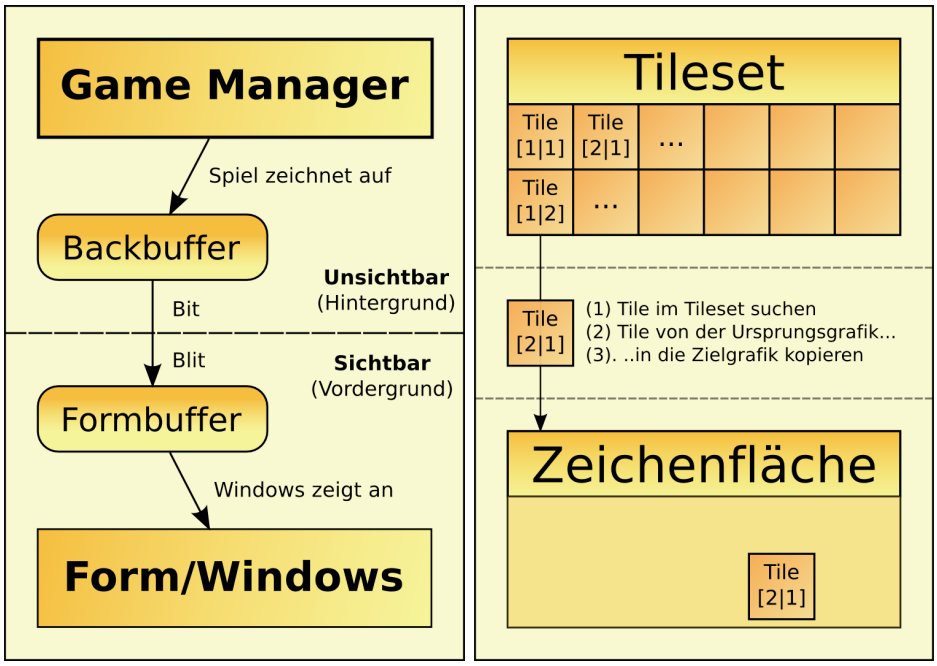


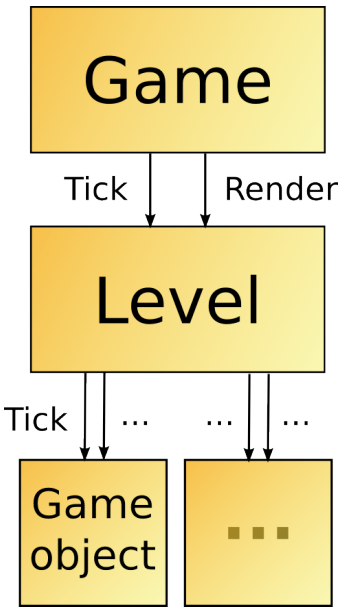
Abbildung 4: Doublebuffering (links) und Tilesets (rechts)

3.4 Umsetzung der Spiellogik

Die Spiellogik lässt sich – wie der Programmkern – in mehrere wichtige Komponenten gliedern:

- Die *Game*-Klasse dient als Verwalterklasse und ruft die Methoden der anderen Klassen auf.
- Die *Level*-Klasse stellt ein zweidimensionales Spielfeld zur Verfügung, und leitet Aufrufe der Game-Klasse an die Spielelemente weiter.
- Das *GameObject* ist die Basis eines jeden Spielelements, und stellt grundlegende Dinge wie Positionierung und Rendering.
- Die *Spielelemente*, welche vom GameObject erben (siehe 3.4.2), implementieren das für das jeweilige Element spezielle Spielverhalten.

Die zwei Threads (Grafik und Spiel) findet man auch in der Umsetzung der Spiellogik wieder: In regelmäßigen Intervallen werden die `Tick()` und `Render()` Methoden des Spielobjekts aufgerufen. `Tick()` sorgt dabei für die Spiellogik; `Render()` für die Darstellung. Alle Aufrufe werden dabei von dem Game-Thread (siehe 3.3) getätigt, und durch die einzelnen Komponenten von der *Game*-Klasse bis zu den einzelnen *GameObjects* geleitet, in welchen anschließend das objekt-spezifische Verhalten implementiert wird:



Game-Klasse:

```
Rufe Tick()/Render() am Level
m_World auf
```

Level-Klasse:

```
Für jedes i von 0 bis LevelBreite-1
 Für jedes j von 0 bis LevelHoehe-1
   Rufe GameObject.Tick()/Render() auf,
   welches in m_World an [i,j] liegt
```

Abbildung 5: Verlauf der Tick- und Renderaufrufe der beiden Threads

3.4.1 Aufbau des Spielfelds

Bei dem Spielfeld handelt es sich bei meiner Umsetzung um ein zweidimensionales Array, d. h. ein Array mit jeweils einer x- und einer y-Komponente. Dabei werden die Indizes in Tile-Einheiten (TE)[8] gemessen.

Um ein leichtes und überschaubares Kollisionsverhalten zu erreichen, habe ich mich dazu entschlossen pro Tile[9] nur ein einzelnes Objekt zu erlauben. Daraus folgt, dass ein Objekt nie gleichzeitig zwei Felder belegen kann, und immer in einem bestimmten Tile in der Welt zu finden ist (siehe 3.5.1).

Wand [0|0]	Wand [1|0]	Wand [2|0]	Wand [3|0]
Wand [0|1]		Stein [2|1]	
Wand [0|2]	Sand [1|2]	Sand [2|2]	Stein [3|2]

Abbildung 6: Beispiel eines Levels (Spielfeld, 4x3 TE)

3.4.2 Spielelemente und Schnittstellen

Nachdem das Prinzip der Spiellogik nun erläutert wurde, kann ich genauer auf die Spielelemente eingehen. Ich habe mich dazu entschieden, einen von Schnittstellen (Interfaces) und abstrakten Klassen geprägten Ansatz zu wählen, um Abhängigkeiten zwischen verschiedenen Spielelementen auf einer abstrakten Ebene darzustellen, so dass man das Spiel leicht um weitere Spielelemente erweitern kann ohne den ganzen Spielkern umzuschreiben.

Den verwendeten Datenstrukturen kann man dabei verschiedene Zwecke zuordnen:

GameObject: Das GameObject stellt alle grundlegenden Funktionen eines Spielelements zur Verfügung. Alle Spielelemente müssen von dieser Klasse erben.

Abstrakte Klassen: Die abstrakten Klassen (AbstractMoveable, AbstractStone, ...) implementieren ein für das Spielelement neues Verhalten, welches nicht einem Einzelnen, aber vielen Spielelementen zur Verfügung gestellt wird.

[8] 1 TE entspricht 32 Pixel
[9] Mit „Tile“ ist ein $[x,y]$ Vektor gemeint.

Interfaces: Die Interfaces (IPushable, ICrackable, ...) beschreiben die Eigenschaften eines bestimmten Spielelements, und aus diesen Eigenschaften folgt die Interaktion durch andere Spielelemente. Dadurch ist es nicht notwendig, dass sich die verschiedenen Spielelemente direkt kennen, sondern die Interaktion kann komplett über die Interfaces geregelt stattfinden.

Klassen: Alle Klassen, welche nicht abstrakt sind, sind im Spiel verfügbare Spielelemente, welche auch für das Spielelement spezifische Verhaltensweisen (z.B. Textur/Darstellung) umsetzen.

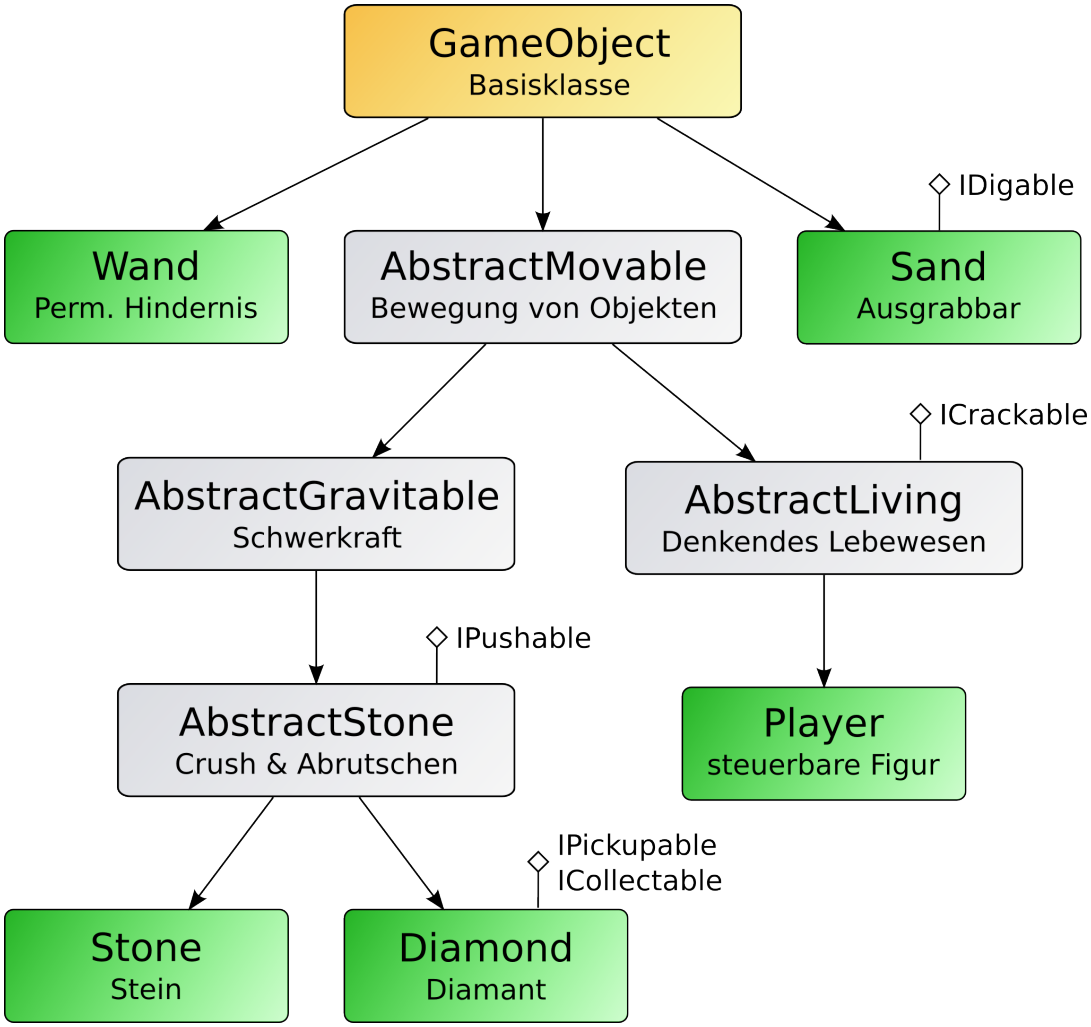


Abbildung 7: Alle Spielelemente der FA-Version

3.5 Wichtige Algorithmen der Spiellogik

Da nun der Aufbau und alle Hauptfunktionen des Spiels erklärt wurden, will ich nun einige spezielle Algorithmen erläutern, die an einigen Stellen verwendet werden. Dabei liegt der Fokus auf den Spielelementen und der restlichen Spiellogik.

3.5.1 Bewegung von Spielobjekten

Als erstes möchte ich dabei auf die Methoden zur Bewegung von Spielelementen eingehen; genauer gesagt auf die Klasse *AbstractMoveable*. Da die Spielwelt nicht nur statisch ist, sondern auch Steine und eine Spielfigur beinhaltet, welche bewegt werden wollen, benötigt man einen Weg um Elemente von einem Tile in ein anderes zu bewegen. Dabei muss die Bewegung zwei Voraussetzungen erfüllen:

1. Die Bewegung muss mit dem Kollisionsverhalten vereinbar sein, d. h. man braucht ein möglichst simples Bewegungsschema, bei dem man auch nur ein Tile überprüfen muss. Deshalb darf in einem Tile auch nur ein Objekt zur gleichen Zeit sein. Verfolgt man diesen Ansatz so würde sich eine tile-basierte Bewegung (immer mind. 32 Pixel) anbieten.

2. Doch sollte die Bewegung nicht nur einfach, sondern auch gut aussehend – und damit flüssig sein. Dies steht jedoch in einem Widerspruch zu einer tile-basierten Bewegung.

Daher habe ich mich für eine Mischung aus beiden Ansätzen entschieden, und dazu die tile-basierte Bewegung eingebaut, welche nur zur Anzeige durch die pixel-basierte Bewegung ergänzt wird. Bewegungen werden bei mir durch einen Richtungsvektor[10] beschrieben, welcher ein leichtes Überprüfen des Zielfeldes ermöglicht.

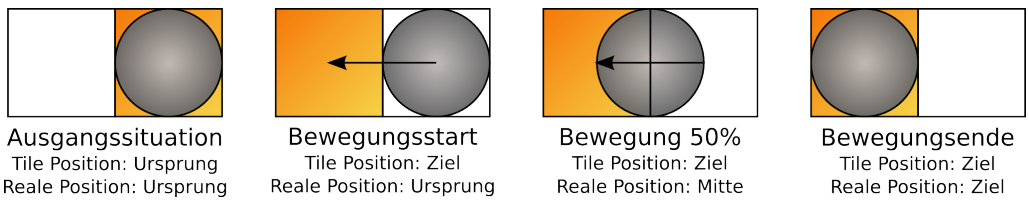


Abbildung 8: Darstellung des Bewegungskonzepts

[10] normalisierter Vektor, z. B. [1|0] oder [0|-1]

Der eigentliche Bewegungsvorgang, welcher von der Spiellogik wahrgenommen wird, läuft daher in den folgenden Schritten ab:

```
Falls wir uns schon bewegen, dann hier aufhören.
Prüfen, ob das Zielfeld belegt ist
(Zugriff auf das World-Array an [px+vx|py+vy])
Falls ja...
  Kollisionsmethode OnCollision aufrufen;
  dort findet elementspezifisches Verhalten statt.
  ggf. erneuter Bewegungsversuch;
Falls nein...
  Interne Position sofort auf das Zielfeld ändern.
  Alten World-Array Eintrag löschen; neuen setzen.
  Verbleibende XY-Pixel-Bewegung auf 32px setzen.
```

Anschließend wird, falls eine Bewegung abläuft, in jedem *Tick()*-Aufruf die pixel-basierte Bewegung durchgeführt um die entsprechende Darstellungsweise zu erreichen:

```
Falls wir uns bewegen...
  In der vergangenen Zeit mögliche Bewegung ausrechnen.
  (Streckenformel: move = elapsed_ms * speed_per_ms)
  Bewegungsweite auf die verbleibende Distanz limitieren.

  Verbleibende Bewegungsdistanz verringern;
  pixel-basierte X- bzw. Y-Position erhöhen.

  Prüfen, ob die verbleibende Distanz 0 ist
  Falls ja
    Bewegungsvorgang beenden.
```

3.5.2 Steuerung und Interaktionen der Spielfigur

Die Spielfigur ist eines der Elemente, welches ein gesondertes Verhalten implementiert. Sie kann vom Spieler durch Betätigung der Tastatur frei in die Himmelsrichtungen bewegt werden. Das Bewegen über die Tastatur ist dabei leicht erledigt:

- Wenn eine Bewegungstaste gedrückt wird, setzen wir die jeweilige Variable auf „*Wahr*“.
- Wenn eine Bewegungstaste losgelassen wird, setzen wir die jeweilige Variable auf „*Falsch*“.
- Bei jedem *Tick*-Aufruf prüfen wir, ob eine Taste gedrückt wurde und bewegen uns gegebenenfalls in diese Richtung.

Doch auch mit diesem Bewegungsverhalten ist die Spielfigur noch nicht komplett. Was fehlt noch? Ein Boulderdash wäre kein Boulderdash, wenn man zum Beispiel nicht einmal Steine verschieben könnte! Deshalb wird in der Kollisions-Ereignis-Methode „*OnCollision*“ für diese Fälle ein entsprechendes Verhalten eingebaut, welches auf dem schon erwähnten Schnittstellenprinzip basiert:

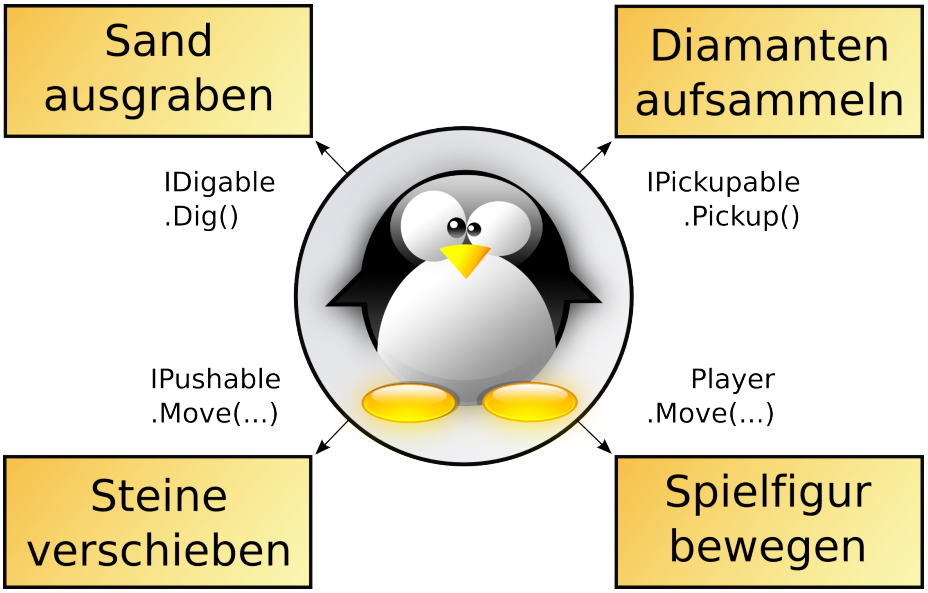


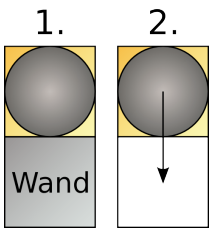
Abbildung 9: Interaktionsschnittstellen der Spielfigur

3.5.3 Steinphysik

Auch bei den Steinen überschreiben wir die Methode „*OnCollision*“ um die verschiedenen Eigenschaften des Steins zu realisieren:

- Ein Stein unterliegt den Einflüssen der Schwerkraft.
- Steine können, falls sie auf einem anderen Stein landen, nach links oder rechts abrutschen.
- Manche Spielelemente können zerschmettert werden, wenn ein Stein auf ihnen landet.

Abbildung 10: Schwerkraft bei Steinen




```
Falls wir uns nicht bewegen,
    Nach unten bewegen.
```

Hinweis: Wenn das Zielfeld belegt ist, so passiert bei dem *Move()*-Aufruf einfach nichts! („Fire-and-Forget“-Prinzip: Das Ergebnis des Aufrufs ist für den Code uninteressant. Alle nötigen Bedingungen werden in der Zielfunktion selber überprüft.)

Abbildung 11: Abrutschen bei Kollision (Stein)

```
Falls wir nicht fallen können,
und unter uns ein Stein ist:
  Falls das Feld Rechts-Unten frei ist:
   Nach rechts bewegen
  Falls das Feld Links-Unten ...
```

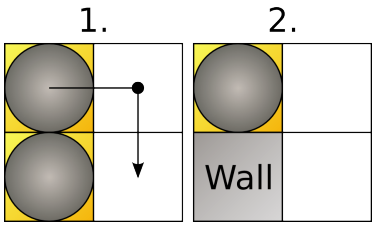
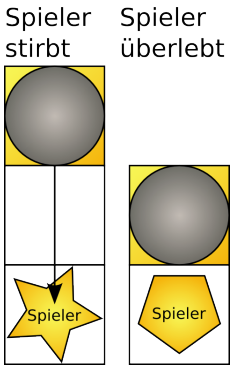


Abbildung 12: Zerschmettern von Gegenständen (Stein)




```
Für jedes erfolgreiche Fallen:
  Gefallene Tiles um 1 erhöhen
Sonst
  Gefallene Tiles auf 0 setzen
```

```
Bei Kollision mit Objekt
Prüfen, ob Gefallene Tiles > 0
Falls Ja
  Crack() Methode aufrufen
```

3.5.4 Bewegbares Sichtfeld

Da Level mit einer maximalen Größe, welche dem Spielfenster entspricht, häufig zu klein sind, ist es wünschenswert den Leveln eine Größe unabhängig von dem Spielfenster geben zu können, und dann nur einen Teilausschnitt anzuzeigen; nämlich den, in welchem sich der Spieler befindet. Doch wie erreicht man das?

In der 2D-Programmierung ist es häufig so, dass man nicht die Kamera (welche ja eigentlich gar nicht existiert, bzw. nur durch das Spielfenster dargestellt wird), sondern das Spielfeld selbst bewegt.

Auch muss man bedenken, dass wir ein flüssiges Bewegungsverhalten haben, und das Sichtfeld auch dementsprechend umsetzen wollen. Deshalb muss die Berechnung des Sichtfelds auf Pixel-Basis geschehen, was wiederum in einem Widerspruch zu den tile-basierten Renderoptimierungen steht, welche dazu dienen, nur einen Teilausschnitt – nämlich den sichtbaren Bereich – darstellen.

Beachtet man alle Probleme, so kommt man zu folgendem Algorithmus, welcher der Bestimmung des Sichtfeldes dient:

1. **Fokuspunkt**

   Den Fokuspunkt bestimmen. In unserem Fall folgen wir immer der Spielfigur, welche das wichtigste Spielelement darstellt.

   ```
   ViewportX = Player.RealX
   ViewportY = Player.RealY
   ```

2. **Zentrierung**

   Da das Sichtfeld auf den Spieler zentriert werden soll, aber wir das Sichtfeld durch den linken oberen Punkt beschreiben, müssen wir das Sichtfeld um die Hälfte der Bildschirmgröße nach links oben verschieben.

   ```
   ViewportX -= ViewportWidth / 2
   ViewportY -= ViewportHeight / 2
   ```

3. **Verhalten an den Kartenrändern**

   Manchmal ist es nicht erwünscht, dass man über die Kartenränder hinaus schauen kann. Deshalb ist es ratsam, dass man das Sichtfeld an diese Ränder anpasst, und dementsprechend limitiert.

```
Falls ViewportX kleiner als 0 ist,
  ViewportX auf 0 setzen.
Falls ViewportX plus die Sichtfeldbreite
größer als die Spielwelt in Pixeln ist,
  ViewportX auf die Spielweltbreite
  minus Sichtfeldbreite setzen.
```

   Hinweis: *Der Code wird jeweils für die x- und die y-Komponente ausgeführt; auch wenn hier nur die x-Komponente dargestellt ist.*

4. **Umkehrung der errechneten Werte**

   Da wir nicht die Kamera, sondern stattdessen das Spielfeld bewegen, muss man bedenken, dass wir die Translation[11] aus der Sicht des Spielfelds sehen müssen. Wir bewegen nicht die Kamera nach rechts, sondern das Spielfeld nach links – also genau in die entgegengesetzte Richtung. Deshalb müssen die errechneten Sichtfeldpositionen umgekehrt werden, sodass sich diese im Negativen befinden:

```
ViewportX = -ViewportX
ViewportY = -ViewportY
```

5. **Optimierungen für die Kartendarstellung**

   Wir haben nun die passenden Transformationskoordinaten ausgerechnet, und könnten so schon die Spielwelt passend darstellen. Doch haben wir noch ein Problem: Wenn die Spielwelt zu groß wird, wird das gesamte Spiel langsamer, da bisher alle Tiles[12] dargestellt werden – egal ob sie sichtbar sind oder nicht. Deshalb rechnen wir noch ein Tilesichtfeld aus, welches die sichtbaren Tiles repräsentiert. Dabei muss man beachten, dass durch die flüssigen Bewegungen auch halbe Tiles sichtbar sein können.

[11] Transformation eines Vektors durch Addition/Subtraktion
[12] Ein Tile entspricht einem Element des World-Arrays an einer bestimmten Stelle

```
TileStartX = ViewportX / 32
TileEndX = TileStartX + ViewportBreite / 32 + 1
```

Hinweis: *Der Programmcode baut darauf auf, dass das Sichtfeld als Integer dargestellt wird, und so bei Division nach unten gerundet wird.*

6. **Darstellung** Nun muss nur noch das Sichtfeld dargestellt werden. Dazu wird, wie zuvor beschrieben, einfach das World-Array von *TileStartX/Y* bis *TileEndX/Y* mit zwei For-Schleifen durchlaufen und jedes Spielelement mit dem zuvor berechnetem *ViewportX/Y*-Offset gerendert.

Anschließend möchte ich noch kurz die wichtigsten Schritte des Algorithmus grafisch darstellen. Das schwarze Rechteck symbolisiert den sichtbaren Bereich; das Füllmuster steht für die im Code gerenderte Spielwelt.

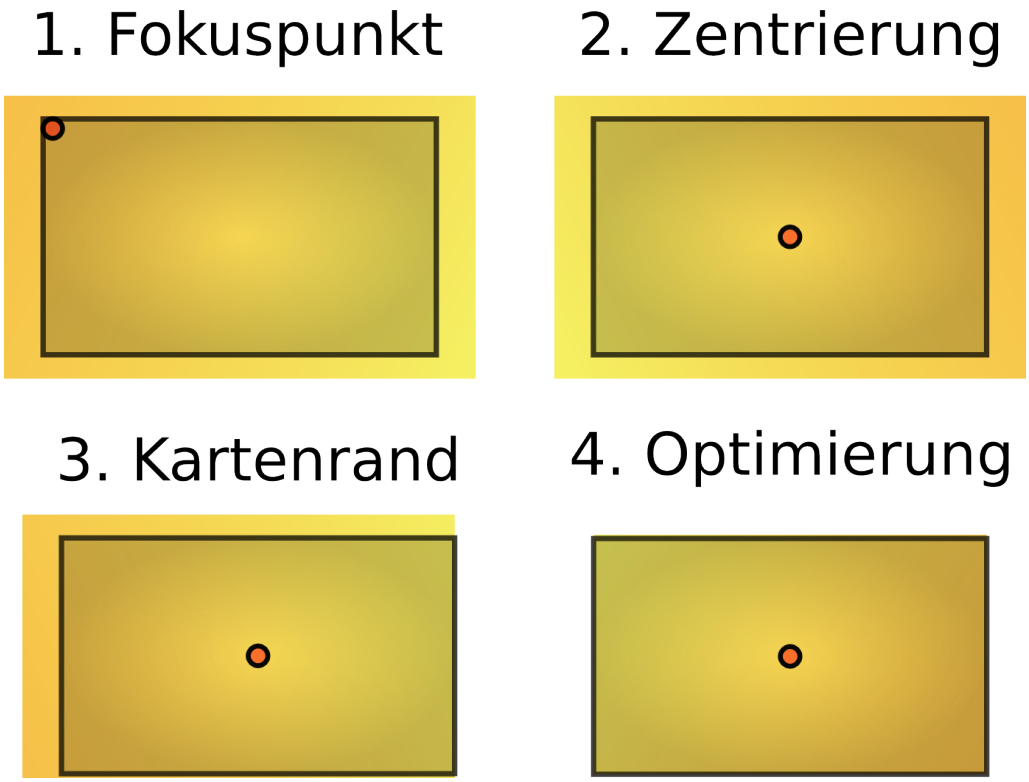


Abbildung 13: Darstellung der Teilstufen des Algorithmus

3.5.5 Generierung eines zufälligen Levels

Da ich in der Facharbeitsversion aufgrund des Umfangs den Leveleditor entfernen musste, habe ich stattdessen zufällig generierte Level eingebaut. Dabei waren folgende Notwendigkeiten zu beachten:

- Die Level sollten mit hoher Wahrscheinlichkeit schaffbar sein.
- Man darf nicht mit einer Dead-End Situation starten, so dass der Level einfach nicht schaffbar ist (sofortiger Tod/Steine im Weg).
- Das Level muss nach dem Tod der Spielfigur erneut ladbar sein, dass heißt man braucht reproduzierbare Level.

Deshalb habe ich mich für einen Ansatz entschieden, welcher jedem Level einen sogenannten „Seed“ zuweist, mit welchem immer der gleiche Level erzeugt wird. Beim Erzeugen des Levels werden folgende Schritte in der angegebenen Reihenfolge ausgeführt:

1. Falls ein Seed übergeben wird, diesen benutzen; ansonsten einen Seed basierend auf der Systemzeit generieren. Den Zufallszahlengenerator mit diesem Seed erstellen.
2. Level mit zufälliger Breite und Höhe erstellen und mit Sand füllen.
3. Zufällig im Level Steine verteilen; die Anzahl der Steine errechnet sich aus der Kartengröße.
4. Zufällig im Level leere Blöcke verteilen; Steine können überschrieben werden.
5. Zufällig die Diamanten verteilen; dabei können Elemente wieder überschrieben werden.
6. Den Rand des Levels erzeugen, indem die Rand-Tiles mit dem „Wand“-Element gefüllt werden.
7. Die Spielfigur an eine zufällige Position setzen und die anliegenden Felder mit Sand füllen, damit der Spieler einen möglichst einfachen Start hat.
8. Den Level im Game-Manager laden.

4 Erweiterungsmöglichkeiten

Während der Entwicklung meiner Facharbeit habe ich zwei verschiedene Versionen von Boulderdash erstellt: Die eine, die ich in der Facharbeit beschrieben habe – und eine zweite, welche deutlich mehr Spielelemente und Funktionen umfasst, aber welche aufgrund ihres Ausmaßes nicht in dieser Facharbeit beschreibbar war.

Die endgültige Version befindet sich auf der CD-ROM im Anhang. Einige Bildschirmfotos der Erweiterungen sieht man am Ende dieses Kapitels.

Dennoch möchte ich noch kurz ein paar Erweiterungen und deren Funktion vorstellen:

4.1 Leveleditor

Der Leveleditor – mitunter die wichtigste Änderung gegenüber der Facharbeits-Version – besteht aus zwei Komponenten:

Die erste Komponente ist die Fähigkeit, Level auf der Festplatte zu speichern und von dieser wieder zu laden. Dabei wird auf die Serialisations-Klassen des .NET Frameworks zurück gegriffen, welche es einem ermöglicht, mit dem *[Serializable]*-Attribut gekennzeichnete Klassen automatisch in einem Binär-Format auf der Festplatte zu speichern. Dadurch ist es nicht notwendig, extra ein eigenes Map-Format zu entwickeln.

Die zweite Komponente ist der eigentliche Leveleditor, welcher auf einem zusätzlichem Fenster aufbaut, das beim Editieren eines Levels neben dem Spiel läuft. Dort werden über *Reflektion*[13] die verschiedenen Spielelemente dargestellt, und können so durch die verschiedenen Zeichentools erstellt werden. Dabei sind folgende Zeichentools vorhanden:

- Das **Punkt/Pinsel-Tool** erlaubt es dem Nutzer durch Bewegen der Maus einzelne Felder zu zeichnen.
- Das **(Füll-)Rechteck-Tool** zeichnet von dem Punkt, an dem man die Maustaste zu drücken beginnt, bis zu dem Punkt, an dem man sie wieder loslässt, ein (gefülltes) Rechteck.
- Das **Füll-Tool** ersetzt iterativ[14] alle anliegenden Tiles, welche dem Ursprungselement entsprechen, mit dem neuen Element.

[13] Analyse des Programmcodes zur Laufzeit
[14] Ein rekursiver Ansatz war wegen der Stackgröße nicht möglich.

4.2 Gegner

Gegner sind mitunter wichtige Spielelemente, da sie im Gegensatz zu den anderen Spielelementen nicht statisch sind, d. h. sich von selber bewegen. Dadurch wird der Spieler in einen Zugszwang getrieben, und braucht eine gewisse Geschicklichkeit und Reaktionszeit um den Level zu schaffen.

Ich habe Gegner implementiert, indem ich von der *AbstractLiving*-Klasse erben lasse und in der *Think()* Methode verschiedene Bewegungsmuster implementiert habe:

- Der **ColliderEnemy** bewegt sich solange in eine Richtung, bis er mit einem Objekt kollidiert. Bei einer Kollision wird eine zufällige, neue Richtung ermittelt.
- Der **RandomEnemy** bewegt sich jedes mal in eine zufällige Richtung, und bietet so ein unvorhersehbares Verhalten.
- Der **GoLeft-** bzw. **GoRightEnemy** versucht, immer in der entsprechenden Richtung an einer Wand entlang zu laufen.

4.3 Weitere Möglichkeiten

Es gibt aber auch einige Möglichkeiten, die ich nicht umgesetzt habe. Einige davon könnten sein:

- Geräusche (Sounds) bei den Aktionen der Spielelemente
- Animationen (z.b. Rotation) bei Steinen
- Ein Mehrspielermodus
- Eine Highscore-Liste
- Leben und ein „Game Over“-Verhalten
- ...und viele mehr.

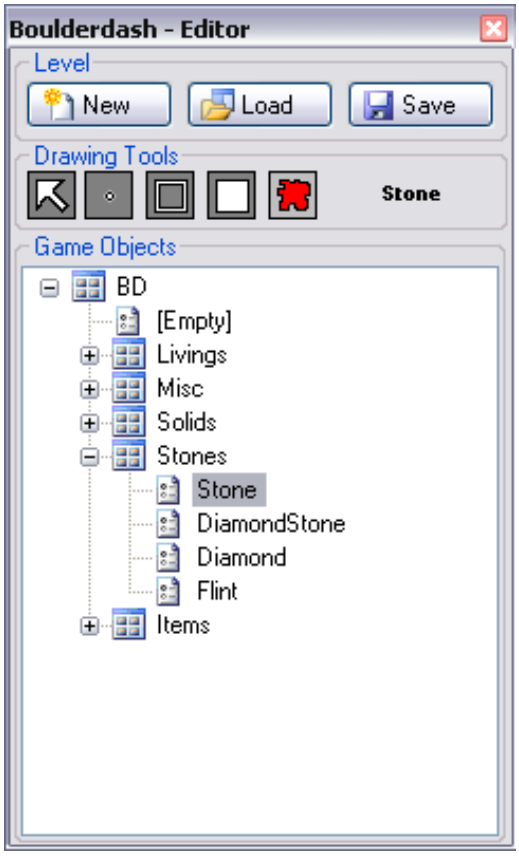


Abbildung 14: Leveleditor-Fenster

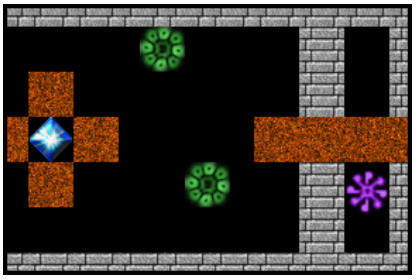


Abbildung 15: Gegner

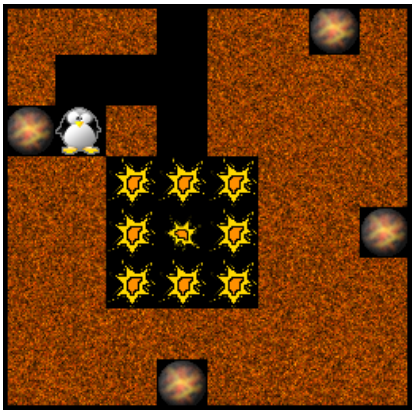


Abbildung 16: Flints

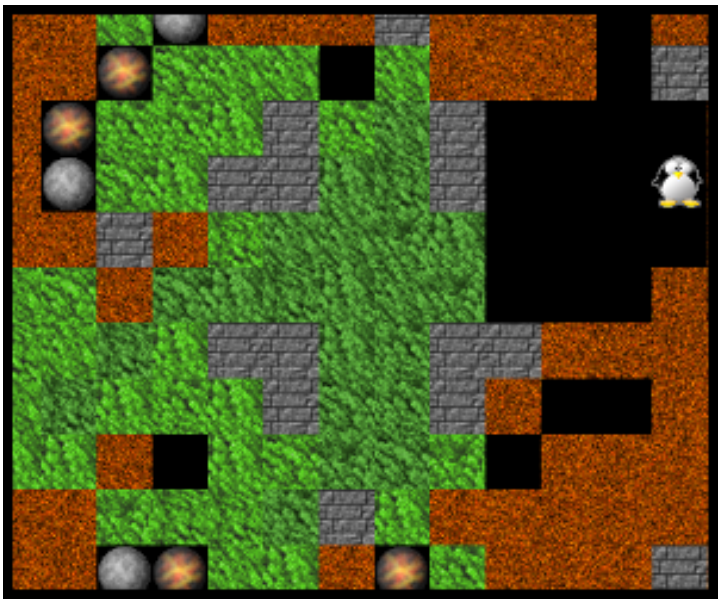


Abbildung 17: Amöben/Pflanzen

5 Anhang

Im Anhang dieser Facharbeit befinden sich auf den folgenden Seiten:

- ein Abbildungsverzeichnis
- die verwendeten Quellen
- die zur Erstellung verwendeten Programme
- der Quellcode der gekürzten Facharbeitsversion
- eine CD mit allen verwendeten Dateien der Facharbeit, sowie den verschiedenen Versionen
- die Erklärung der selbstständigen Erstellung der Facharbeit

Abbildungsverzeichnis

Quellen

Informationen über Boulderdash

[1] http://de.wikipedia.org/wiki/Boulder_Dash
[2] http://www.c64-wiki.de/index.php/Boulder_Dash
[3] http://www.boulder-dash.com/boulderdash.htm

Informationen über C# und .NET

[4] http://de.wikipedia.org/wiki/CSharp
[5] http://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung
[6] http://openbook.galileocomputing.de/csharp/

Hinweis: Alle Quellen befinden sich in gespeicherter Form auf der CD.

Verwendete Programme

- Ubuntu 9.04 (Betriebssystem)
- Microsoft Visual Studio 2008 C# Express Edition (Entwicklungsumgebung)
- .NET Framework 2.0 (Laufzeitumgebung)
- Inkscape Vector Graphics Editor (Diagramme)
- GIMP - GNU Image Manipulation Program (Grafik)
- Doxygen (Befehlsreferenz-Erstellung)
- Kile (Latex Entwicklungsumgebung)

Bei der Produktion dieser Facharbeit sind keine Pinguine verletzt worden!

Quellcode

- Core.dll -

Core/Core.cs

```
1  using System;
2  using System.Threading;
3  using System.Windows.Forms;
4
5  namespace BDCore
6  {
7    /// <summary>
8    /// Delegate für das Tick Event
9    /// </summary>
10   /// <param name="c"></param>
11   public delegate void TickDelegate(GameControl c);
12
13   /// <summary>
14   /// Boulderdash Core
15   /// </summary>
16   public static class Core
17   {
18     #region Properties
19
20     /// <summary>
21     /// Gibt wieder, ob das Fenster noch angezeigt wird
22     /// </summary>
23     public static bool FormShown { get; private set; }
24
25     /// <summary>
26     /// Start Time
27     /// </summary>
28     private static long StartTime { get; set; }
29
30     /// <summary>
31     /// Vergangene CoreTime in Millisekunden
32     /// </summary>
33     public static long Time
34     {
35       get { return GetTicks() - StartTime; }
36     }
37
38     /// <summary>
39     /// Time String im hh:mm:ss.ms Format
40     /// </summary>
41     public static string TimeString
42     {
43       get { return new TimeSpan(0, 0, 0, 0, (int) Time).ToString(); }
44     }
45
46     ///<summary>
47     /// Gibt wieder, ob der Kern aktuell initialisiert ist oder nicht
48     ///</summary>
49     public static bool Running { get; private set; }
50
51     ///<summary>
52     /// FPS des Form-Threads (Nicht exakt, aber angenähert)
53     ///</summary>
54     public static double FPS { get; private set; }
55
56     ///<summary>
57     /// Game Control
58     ///</summary>
59     public static GameControl GameControl { get; private set; }
60
61     /// <summary>
62     /// Game Form
63     /// </summary>
64     public static Form GameForm { get; private set; }
65
66     #endregion
67
68     #region Init/Deinit
69
70     /// <summary>
71     /// Initialisiert das Kernprogramm
72     /// </summary>
73     /// <param name="c">GameControl</param>
74     public static void Init(GameControl c)
75     {
76       if (Running)
77         return;
78       if ((GameForm = c.FindForm()) == null)
79         return;
80
81       //Attribute ändern
82       Running = true;
83       GameControl = c;
84       FormShown = true;
85
86       //Zeitgeber starten
87       StartTime = GetTicks();
88
89       //Form einstellen
90       GameForm.FormClosing += OnFormClosing;
91
92       //Graphic Core
93       Graphic.Init();
94
95       //Keyboard
96       Keyboard.Init();
97
98       //Anwendung starten
99       Thread ft = new Thread(ApplicationThread);
100      ft.IsBackground = true;
101      ft.SetApartmentState(ApartmentState.STA); //FormThread = Single Thread COM Model
102      ft.Start();
103
104      //Auf den Thread "warten"
105      Thread.Sleep(0);
```

```
106       }
107
108       /// <summary>
109       /// Deinitialisiert den Kern
110       /// </summary>
111       public static void Deinit()
112       {
113         if (!Running)
114           return; //Bereits deinitialisiert
115
116         Running = false;
117
118         while (FormShown)
119           Thread.Sleep(300); //Thread Exit
120
121         GameForm.FormClosing -= OnFormClosing;
122
123         //Keyboard
124         Keyboard.Deinit();
125
126         //Graphic
127         Graphic.Deinit();
128       }
129
130       #endregion
131
132       #region Form Thread
133
134       /// <summary>
135       /// Wird jeden AppTick aufgerufen
136       /// </summary>
137       public static event TickDelegate OnApplicationTick;
138
139       /// <summary>
140       /// Form (and Graphic) Thread
141       /// </summary>
142       private static void ApplicationThread()
143       {
144         const int LOOP = 1000/33; //33 Render FPS
145
146         //1. Form öffnen
147         Thread.Sleep(0);
148         GameForm.Show();
149         Application.DoEvents();
150
151         //2. Haupt Form-Loop
152         while (Running && FormShown)
153         {
154           long start = Time;
155
156           //In dem unteren Bereich wird die Zeit
157           //effektiv gemessen
158           {
159             //Form und Grafik liegen auf dem gleichen Thread,
160             //da sich die beiden Threads synchron verlaufen müssen
161             //(sonst: unnötiger Leistungseinsatz/"Datenverlust")
162             // /!\ Change: Nun über ein Event geregelt.
163             if (OnApplicationTick != null)
164               OnApplicationTick(GameControl);
165             Application.DoEvents(); //Verarbeiten von Windows Nachrichten (Message Loop)
166           }
167           int elapsed = (int) (Time - start);
168
169           //Wir warten die verbleibende Zeit, um auf
170           //unsere angegebenen FPS zu kommen
171           //Thread pausieren, um die CPU nicht zu sehr auszulasten
172           //Daher ist Sleep() hier nicht "möglich"
173           Sleep(Math.Max(LOOP - elapsed, 0));
174
175           FPS = (9*FPS + (1000.0/Math.Max((Time - start), 1)))/10.0;
176         }
177
178         if (FormShown && !GameForm.Disposing)
179           GameForm.Close();
180
181         Application.DoEvents();
182       }
183
184       /// <summary>
185       /// Aufgerufen, wenn das Hauptfenster geschlossen wird
186       /// </summary>
187       /// <param name="sender"></param>
188       /// <param name="e"></param>
189       private static void OnFormClosing(object sender, FormClosingEventArgs e)
190       {
191         FormShown = false;
192       }
193
194       #endregion
195
196       /// <summary>
197       /// Wartet für die angegebene Zeit (ms)
198       /// </summary>
199       /// <param name="ms"></param>
200       public static void Sleep(int ms)
201       {
202         if (ms < 0)
203           return; //oops!
204
205         //Thread Sleep ist ungenauer, verbraucht aber auch deutlich weniger Performance
206         Thread.Sleep(ms);
207         return;
208       }
209
210       /// <summary>
211       /// Gibt die aktuellen Ticks in MS wieder
212       /// </summary>
```

```
213     /// <returns></returns>
214     private static long GetTicks()
215     {
216       //Stopwatch bietet mehr Funktionalität, ist aber nicht immer gleich schnell
217       //(Siehe "Resourcen\TimerTest" - 60sec -> 45 sec)
218 
219       //DateTime - Vorteil: Zeit stimmt - Nachteil: Geringe Genauigkeit (~15ms)
220       return DateTime.UtcNow.Ticks/10000;
221     }
222   }
223 }
```

Core/Graphic.cs

```
1  using System.Drawing;
2  using System.Windows.Forms;
3  using BDCore.Helpers;
4 
5  namespace BDCore
6  {
7    /// <summary>
8    /// Render Delegate
9    /// </summary>
10   /// <param name="t"></param>
11   public delegate void RenderDelegate(Texture t);
12 
13   /// <summary>
14   /// Verantwortlich für das Zeichnen des Spielinhaltes
15   /// </summary>
16   public static class Graphic
17   {
18     #region Attributes/Properties
19 
20     private static Texture m_BackBuffer; //Draw Buffer
21 
22     /// <summary>
23     /// Gibt die aktuelle, verfügbare Zeichenfläche wieder
24     /// </summary>
25     public static Graphics Surface
26     {
27       get { return m_BackBuffer.Graphics; }
28     }
29 
30     /// <summary>
31     /// Width
32     /// </summary>
33     public static int Width
34     {
35       get
36       {
37         if (m_BackBuffer == null)
38           return 0;
39 
40         return m_BackBuffer.Bitmap.Width;
41       }
42     }
43 
44     /// <summary>
45     /// Height
46     /// </summary>
47     public static int Height
48     {
49       get
50       {
51         if (m_BackBuffer == null)
52           return 0;
53 
54         return m_BackBuffer.Bitmap.Height;
55       }
56     }
57 
58     #endregion
59 
60     #region Init/Deinit
61 
62     /// <summary>
63     /// Init
64     /// </summary>
65     public static void Init()
66     {
67       //Buffer erstellen
68       m_BackBuffer = new Texture(Core.GameControl.ClientSize);
69 
70       //Events
71       Core.OnApplicationTick += OnAppTick;
72       Core.GameControl.Paint += OnPaint;
73     }
74 
75     /// <summary>
76     /// Deinit
77     /// </summary>
78     public static void Deinit()
79     {
80       //Events..
81       Core.OnApplicationTick -= OnAppTick;
82       Core.GameControl.Paint -= OnPaint;
83 
84       //Buffer freigeben
85       m_BackBuffer.Dispose();
86 
87       //Tilesets
88       foreach (Tileset t in Tileset.Tilesets.Values)
89         t.Dispose();
90     }
91 
```

```
92      #endregion
93
94      #region Render Methods
95
96      /// <summary>
97      /// Render Event
98      /// </summary>
99      public static event RenderDelegate Render;
100
101     /// <summary>
102     /// Zeichnet den Spielinhalt auf das Zielobjekt
103     /// </summary>
104     /// <param name="target"></param>
105     public static void RenderToGraphics(Graphics target)
106     {
107       //1. Render on Backbuffer (Spiel zeichnen)
108       if (Render != null) Render(m_BackBuffer);
109       else m_BackBuffer.Graphics.Clear(Color.DimGray);
110
111       //2. Render on Form (Anzeigen)
112       Blit(m_BackBuffer, target);
113     }
114
115     /// <summary>
116     /// Copies the content from one buffer into another
117     /// </summary>
118     /// <param name="t">Source</param>
119     /// <param name="g">Destination</param>
120     private static void Blit(Texture t, Graphics g)
121     {
122       //# Leer: 1000 FPS || 1000 FPS
123       //# Draw Image: 333 FPS || 200 FPS
124       //# Draw Partial: 283 FPS
125       //# Bit Blit: ~285 FPS || 41 FPS
126
127       //# Draw Unscaled: 333 FPS || 200 FPS
128       g.DrawImageUnscaled(t.Bitmap, 0, 0);
129     }
130
131     private static void OnPaint(object sender, PaintEventArgs e)
132     {
133       RenderToGraphics(e.Graphics);
134     }
135
136     private static void OnAppTick(GameControl c)
137     {
138       //Direct Paint
139       RenderToGraphics(c.CreateGraphics()); //Schneller als .Refresh()
140     }
141
142     #endregion
143   }
144 }
```

Core/Keyboard.cs

```
1  using System;
2  using System.Windows.Forms;
3
4  namespace BDCore
5  {
6    /// <summary>
7    /// Keyboard
8    /// </summary>
9    public static class Keyboard
10   {
11     /// <summary>
12     /// Wird vor dem KeyDown Ereigniss ausgelöst
13     /// </summary>
14     public static event PreviewKeyDownEventHandler KeyPress;
15
16     /// <summary>
17     /// Wird ausgelöst, wenn die Taste runter gedrückt wird
18     /// </summary>
19     public static event KeyEventHandler KeyDown;
20
21     /// <summary>
22     /// Wird ausgelöst, wenn die Taste losgelassen wird
23     /// </summary>
24     public static event KeyEventHandler KeyUp;
25
26     /// <summary>
27     /// Init
28     /// </summary>
29     public static void Init()
30     {
31       Core.GameControl.PreviewKeyDown += OnPreviewKeyDown;
32       Core.GameControl.KeyDown += OnKeyDown;
33       Core.GameControl.KeyUp += OnKeyUp;
34     }
35
36     /// <summary>
37     /// Aufgerufen, sobald eine Taste gedrückt wird
38     /// </summary>
39     /// <param name="sender"></param>
40     /// <param name="e"></param>
41     private static void OnPreviewKeyDown(object sender, PreviewKeyDownEventArgs e)
42     {
43       if (KeyPress != null)
44         KeyPress(sender, e);
45     }
46
47     /// <summary>
48     /// Aufgerufen, sobald eine Taste herunter gedrückt wird
49     /// </summary>
```

```
50      /// <param name="sender"></param>
51      /// <param name="e"></param>
52      private static void OnKeyDown(object sender, KeyEventArgs e)
53      {
54        if (KeyDown != null)
55          KeyDown(sender, e);
56
57        //Mono (Linux) löst nur KeyDown, keine KeyPress Events aus
58        if (Environment.OSVersion.Platform == PlatformID.Unix && KeyPress != null)
59          KeyPress(sender, new PreviewKeyDownEventArgs(e.KeyCode));
60      }
61
62      /// <summary>
63      /// Aufgerufen, sobald eine Taste los gelassen wird
64      /// </summary>
65      /// <param name="sender"></param>
66      /// <param name="e"></param>
67      private static void OnKeyUp(object sender, KeyEventArgs e)
68      {
69        if (KeyUp != null)
70          KeyUp(sender, e);
71      }
72
73      /// <summary>
74      /// Deinit
75      /// </summary>
76      public static void Deinit()
77      {
78        Core.GameControl.KeyDown -= OnKeyDown;
79        Core.GameControl.KeyUp -= OnKeyUp;
80        Core.GameControl.PreviewKeyDown -= OnPreviewKeyDown;
81      }
82    }
83  }
```

Core/Helpers/Draw.cs

```
1   using System;
2   using System.Drawing;
3   using System.Drawing.Imaging;
4
5   namespace BDCore.Helpers
6   {
7     /// <summary>
8     /// Drawing Methods
9     /// Methoden zum Zeichnen auf ein Graphics Obj
10    /// </summary>
11    internal static class Draw
12    {
13      /// <summary>
14      /// Pixel Format [Hier: Natives GDI+ Format (Am schnellsten)]
15      /// </summary>
16      public const PixelFormat FORMAT = PixelFormat.Format32bppPArgb;
17
18      /// <summary>
19      /// Kopiert den Inhalt des Ursprungsbildes in ein neues, kompatibles Format
20      /// Gibt das Ursprungsbild frei
21      /// </summary>
22      /// <param name="src"></param>
23      /// <returns></returns>
24      public static Image GetCompatible(Image src)
25      {
26        Bitmap res = new Bitmap(src.Width, src.Height, FORMAT);
27
28        using (Graphics g = Graphics.FromImage(res))
29          g.DrawImage(src, 0, 0, src.Width, src.Height);
30
31        src.Dispose();
32        return res;
33      }
34
35      /// <summary>
36      /// Überträgt den Ausschnitt der Ursprungsgrafik in die Zielgrafik
37      /// </summary>
38      /// <param name="source"></param>
39      /// <param name="dest"></param>
40      /// <param name="srcRect"></param>
41      /// <param name="destRect"></param>
42      public static void Blit2D(Texture source, Texture dest, RectangleF srcRect, RectangleF
            destRect)
43      {
44        try
45        {
46          dest.Graphics.DrawImage(source.Bitmap, destRect, srcRect, GraphicsUnit.Pixel);
47        }
48        catch (ArgumentException)
49        {
50          //Render tick while form was closed
51        }
52      }
53    }
54  }
```

Core/Helpers/ISprite.cs

```
1   using System.Drawing;
2
3   namespace BDCore.Helpers
4   {
5     /// <summary>
6     /// Darstellbares Objekt
7     /// </summary>
```

```
8   public interface ISprite
9   {
10    /// <summary>
11    /// Stellt das Sprite an der angegebenen Position auf dem Graphics Objekt dar
12    /// </summary>
13    /// <param name="t">Zieltextur</param>
14    /// <param name="elapsed">Vergangene Zeit</param>
15    /// <param name="position">Zielposition</param>
16    void Render(Texture t, int elapsed, Point position);
17  }
18 }
```

Core/Helpers/Tileset.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Drawing;
4  using System.IO;
5
6  namespace BDCore.Helpers
7  {
8    /// <summary>
9    /// Tileset, welches mehrere Tiles enthält
10   /// </summary>
11   public sealed class Tileset : IDisposable
12   {
13     #region Instance
14
15     /// <summary>
16     /// Lädt ein Tileset (Öffnet die angegebene Datei)
17     /// </summary>
18     /// <param name="file"></param>
19     private Tileset(string file)
20     {
21       File = Path.GetFileNameWithoutExtension(file);
22
23       Source = new Texture(Image.FromFile(file));
24
25       //Eigenschaften setzen
26       WidthTiles = Source.Width/Tile.TILE_WIDTH;
27       HeightTiles = Source.Height/Tile.TILE_HEIGHT;
28     }
29
30     /// <summary>
31     /// File
32     /// </summary>
33     public string File { get; private set; }
34
35     /// <summary>
36     /// Source Texture
37     /// </summary>
38     public Texture Source { get; private set; }
39
40     /// <summary>
41     /// Width in Tiles
42     /// </summary>
43     public int WidthTiles { get; private set; }
44
45     /// <summary>
46     /// Height in Tiles
47     /// </summary>
48     public int HeightTiles { get; private set; }
49
50     public void Dispose()
51     {
52       Source.Dispose();
53     }
54
55     /// <summary>
56     /// Gibt das Tile an der angegebenene Position wieder
57     /// </summary>
58     /// <param name="x"></param>
59     /// <param name="y"></param>
60     /// <returns></returns>
61     public Tile GetTile(int x, int y)
62     {
63       return new Tile(this, Utils.Bound(x, 1, WidthTiles), Utils.Bound(y, 1, HeightTiles));
64     }
65
66     #endregion
67
68     #region Static
69
70     /// <summary>
71     /// Initialisiert die Tilesets-Liste
72     /// </summary>
73     static Tileset()
74     {
75       Tilesets = new Dictionary<string, Tileset>();
76     }
77
78     /// <summary>
79     /// Liste aller geladenen Tilesets
80     /// </summary>
81     public static Dictionary<string, Tileset> Tilesets { get; private set; }
82
83     /// <summary>
84     /// Lädt ein Tileset
85     /// </summary>
86     /// <param name="file"></param>
87     /// <returns></returns>
88     public static Tileset LoadTileset(string file)
89     {
90       if (!System.IO.File.Exists(file))
91         return null;
```

```
92
93          Tileset t = new Tileset(file);
94          Tilesets.Add(t.File, t);
95          return t;
96        }
97
98        /// <summary>
99        /// Ruft ein Tile ab
100       /// </summary>
101       /// <param name="tileset">Tilesetnamen</param>
102       /// <param name="x">Tile X (1-Based)</param>
103       /// <param name="y">Tile Y (1-Based)</param>
104       /// <returns></returns>
105       public static Tile Get(string tileset, int x, int y)
106       {
107         if (!Tilesets.ContainsKey(tileset))
108           throw new ArgumentNullException("tileset", "Tileset not found: " + tileset);
109
110         return Tilesets[tileset].GetTile(x, y);
111       }
112
113       #endregion
114     }
115 }
```

Core/Helpers/Tile.cs

```
1  using System.Drawing;
2
3  namespace BDCore.Helpers
4  {
5    /// <summary>
6    /// Einzelnes Tile
7    /// Existiert innerhalb eines Tilesets
8    /// </summary>
9    public class Tile : ISprite
10   {
11     /// <summary>
12     /// Hoehe eines Tiles
13     /// </summary>
14     public const int TILE_HEIGHT = 32;
15
16     /// <summary>
17     /// Breite eines einzelnen Tiles
18     /// </summary>
19     public const int TILE_WIDTH = TILE_HEIGHT;
20
21     /// <summary>
22     /// Erstellt ein neues Tile
23     /// </summary>
24     /// <param name="s"></param>
25     /// <param name="x"></param>
26     /// <param name="y"></param>
27     public Tile(Tileset s, int x, int y)
28     {
29       TileSet = s;
30       TileX = x;
31       TileY = y;
32     }
33
34     /// <summary>
35     /// Tileset
36     /// </summary>
37     public Tileset TileSet { get; private set; }
38
39     /// <summary>
40     /// Tile X
41     /// </summary>
42     public int TileX { get; private set; }
43
44     /// <summary>
45     /// Tile Y
46     /// </summary>
47     public int TileY { get; private set; }
48
49     #region ISprite Members
50
51     public void Render(Texture t, int elapsed, Point position)
52     {
53       Rectangle src = new Rectangle((TileX - 1)*TILE_WIDTH, (TileY - 1)*TILE_HEIGHT, TILE_WIDTH,
           TILE_HEIGHT);
54       Rectangle dest = new Rectangle(position.X, position.Y, src.Width, src.Height);
55       Draw.Blit2D(TileSet.Source, t, src, dest);
56     }
57
58     #endregion
59   }
60 }
```

Core/Helpers/Texture.cs

```
1  using System;
2  using System.Drawing;
3
4  namespace BDCore.Helpers
5  {
6    /// <summary>
7    /// Texture (Bilddatei)
8    /// </summary>
9    public sealed class Texture : IDisposable
10   {
11     /// <summary>
```

```
12      /// Initialisiert eine Textur und gibt das Ursprungsbild frei
13      /// </summary>
14      /// <param name="i"></param>
15      public Texture(Image i)
16      {
17        Bitmap = Draw.GetCompatible(i);
18
19        Width = Bitmap.Width;
20        Height = Bitmap.Height;
21
22        Graphics = Graphics.FromImage(Bitmap);
23      }
24
25      /// <summary>
26      /// Init
27      /// </summary>
28      /// <param name="size"></param>
29      public Texture(Size size) : this(new Bitmap(size.Width, size.Height))
30      {
31        Graphics.Clear(Color.Black);
32      }
33
34      /// <summary>
35      /// Width
36      /// </summary>
37      public int Width { get; private set; }
38
39      /// <summary>
40      /// Height
41      /// </summary>
42      public int Height { get; private set; }
43
44      /// <summary>
45      /// Source Image
46      /// </summary>
47      public Image Bitmap { get; private set; }
48
49      /// <summary>
50      /// Graphics
51      /// </summary>
52      public Graphics Graphics { get; private set; }
53
54      #region IDisposable Members
55
56      public void Dispose()
57      {
58        Bitmap.Dispose();
59        Graphics.Dispose();
60        GC.SuppressFinalize(this); //Standard .NET Implementierung
61      }
62
63      #endregion
64    }
65  }
```

Core/Helpers/Fonts.cs

```
1   using System.Drawing;
2
3   namespace BDCore.Helpers
4   {
5     /// <summary>
6     /// Container-Klasse für verwendete Schriftarten
7     /// </summary>
8     public static class Fonts
9     {
10      public static readonly Font Comic10 = new Font("Comic Sans MS", 10);
11      public static readonly Font Comic15 = new Font("Comic Sans MS", 15);
12      public static readonly Font Comic25 = new Font("Comic Sans MS", 25);
13      public static readonly Font Comic33 = new Font("Comic Sans MS", 33);
14    }
15  }
```

Core/Helpers/Utils.cs

```
1   using System;
2
3   namespace BDCore.Helpers
4   {
5     /// <summary>
6     /// Helfer Klasse
7     /// </summary>
8     public static class Utils
9     {
10      /// <summary>
11      /// Random Number Generator
12      /// </summary>
13      private static readonly Random RandomGen = new Random();
14
15      /// <summary>
16      /// Clipped die Zahl in den angegebenen Bereich
17      /// </summary>
18      /// <param name="val"></param>
19      /// <param name="min"></param>
20      /// <param name="max"></param>
21      /// <returns></returns>
22      public static int Bound(int val, int min, int max)
23      {
24        return Math.Min(Math.Max(val, min), max);
25      }
26
27      /// <summary>
```

```
28      /// Clipped die Zahl in den angegebenen Bereich
29      /// </summary>
30      /// <param name="val"></param>
31      /// <param name="min"></param>
32      /// <param name="max"></param>
33      /// <returns></returns>
34      public static float Bound(float val, float min, float max)
35      {
36        return Math.Min(Math.Max(val, min), max);
37      }
38
39      /// <summary>
40      /// Gibt eine zufällige Zahl zwischen MIN und MAX wieder (inklusiv)
41      /// </summary>
42      /// <param name="min"></param>
43      /// <param name="max"></param>
44      /// <returns></returns>
45      public static int Random(int min, int max)
46      {
47        return RandomGen.Next(min, max + 1);
48      }
49
50      /// <summary>
51      /// Random Chance
52      /// </summary>
53      /// <param name="perc">Percent</param>
54      /// <returns></returns>
55      public static bool Chance(int perc)
56      {
57        return Random(0, 100) <= perc;
58      }
59    }
60  }
```

Core/GameControl.cs

```
1  using System;
2  using System.Windows.Forms;
3
4  namespace BDCore
5  {
6    /// <summary>
7    /// Game Control
8    /// </summary>
9    public partial class GameControl : Control
10   {
11     /// <summary>
12     /// Constructor
13     /// </summary>
14     public GameControl()
15     {
16       InitializeComponent();
17
18       //Flags für das "eigene" Zeichnen setzen
19       SetStyle(ControlStyles.UserPaint, true);
20     }
21
22     protected override void OnPaint(PaintEventArgs e)
23     {
24       base.OnPaint(e);
25
26       if (DesignMode)
27         e.Graphics.Clear(BackColor);
28     }
29
30     protected override void OnPaintBackground(PaintEventArgs pevent)
31     {
32       //Verhindern, dass der Hintergrund gezeichnet wird
33     }
34   }
35 }
```

Core/GameControl.Designer.cs

```
1  ï»¿namespace BDCore
2  {
3      partial class GameControl
4      {
5          /// <summary>
6          /// Required designer variable.
7          /// </summary>
8          private System.ComponentModel.IContainer components = null;
9
10         /// <summary>
11         /// Clean up any resources being used.
12         /// </summary>
13         /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</
               param>
14         protected override void Dispose(bool disposing)
15         {
16             if (disposing && (components != null))
17             {
18                 components.Dispose();
19             }
20             base.Dispose(disposing);
21         }
22
23         #region Component Designer generated code
24
25         /// <summary>
26         /// Required method for Designer support - do not modify
27         /// the contents of this method with the code editor.
```

```
28          /// </summary>
29          private void InitializeComponent()
30          {
31              this.SuspendLayout();
32              //
33              // GameControl
34              //
35              this.BackColor = System.Drawing.Color.DimGray;
36              this.MinimumSize = new System.Drawing.Size(800, 600);
37              this.Size = new System.Drawing.Size(800, 600);
38              this.ResumeLayout(false);
39
40          }
41
42          #endregion
43      }
44  }
```

Quellcode

- Game.dll -

Game/Game.cs

```
1  using System;
2  using System.Drawing;
3  using System.IO;
4  using System.Threading;
5  using System.Windows.Forms;
6  using BDCore;
7  using BDCore.Helpers;
8  using BDGame.World;
9  using BDGame.World.Livings;
10
11 namespace BDGame
12 {
13   /// <summary>
14   /// Boulderdash Game
15   /// </summary>
16   public static class Game
17   {
18     #region Constants
19
20     /// <summary>
21     /// Viewport X Offset
22     /// </summary>
23     internal const int VIEWPORT_OFFSET_X = 0;
24
25     /// <summary>
26     /// Viewport Y Offset
27     /// </summary>
28     internal const int VIEWPORT_OFFSET_Y = 66;
29
30     #endregion
31
32     #region Attributes + Properties
33
34     /// <summary>
35     /// World
36     /// </summary>
37     private static Level m_World;
38
39     ///<summary>
40     /// GameMgr Running?
41     ///</summary>
42     public static bool Running { get; private set; }
43
44     /// <summary>
45     /// Level
46     /// Get: Gibt das aktuelle Level wieder
47     /// Set: Gibt das alte Level frei, setzt das neue Level und initialisiert es
48     /// </summary>
49     internal static Level World
50     {
51       get { return m_World; }
52       private set
53       {
54         lock (m_World ?? new object())
55         {
56           //Asynchronität beim Levelwechsel vermeiden
57
58           //Alte Welt freigeben
59           if (m_World != null)
60             m_World.Unload();
61
62           //Zuweisen
63           GameTime = 0;
64           m_World = value;
65
66           //Neue Welt initialisieren
67           m_World.Prepare();
68         }
69       }
70     }
71
72     /// <summary>
73     /// Paused?
74     /// </summary>
75     internal static bool Paused { get; set; }
76
77     /// <summary>
78     /// Player
79     /// (Controlled)
80     /// </summary>
81     internal static Player Controller { get; set; }
82
83     /// <summary>
84     /// Elapsed Game Time
85     /// </summary>
86     internal static int GameTime { get; private set; }
87
88     /// <summary>
89     /// Viewport Width in Pixels
90     /// </summary>
91     internal static int ViewportPixelsX
92     {
93       get { return Graphic.Width; }
94     }
95
96     /// <summary>
97     /// Viewport Height in Pixels
98     /// </summary>
99     internal static int ViewportPixelsY
100    {
101      get { return Graphic.Height - 66 - 22; }
102    }
103
104    /// <summary>
105    /// Viewport Width in Tiles
```

```
106     /// </summary>
107     internal static int ViewportTilesX
108     {
109       get { return ViewportPixelsX/Tile.TILE_WIDTH; }
110     }
111
112     /// <summary>
113     /// Viewport Height in Tiles
114     /// </summary>
115     internal static int ViewportTilesY
116     {
117       get { return ViewportPixelsY/Tile.TILE_HEIGHT; }
118     }
119
120     #endregion
121
122     #region Init, Deiinit und GameThread
123
124     /// <summary>
125     /// Init
126     /// </summary>
127     public static void Init()
128     {
129       if (Running)
130         return;
131
132       Running = true;
133
134       //Daten Laden
135       LoadData();
136
137       Thread t = new Thread(GameThread);
138       t.IsBackground = true; //Thread ist nicht der Hauptthread
139       t.Start();
140
141       //Events
142       Graphic.Render += RenderGame;
143       Keyboard.KeyPress += OnKeyPress;
144
145       //Level
146       DoRandomLevel();
147     }
148
149     /// <summary>
150     /// Deinit
151     /// </summary>
152     public static void Deinit()
153     {
154       if (!Running)
155         return;
156
157       if (World != null)
158         World.Unload();
159
160       //Events
161       Keyboard.KeyPress -= OnKeyPress;
162       Graphic.Render -= RenderGame;
163       Running = false;
164     }
165
166     /// <summary>
167     /// Game Thread
168     /// </summary>
169     private static void GameThread()
170     {
171       const int LOOP = 1000/66; //66 Game FPS
172
173       int loopElapsed = 0;
174
175       while (Running)
176       {
177         long start = Core.Time;
178
179         //Spieltick
180         TickGame(loopElapsed);
181
182         //Vergangene Zeit durch den (das?) Tick
183         int codeElapsed = (int) (Core.Time - start);
184
185         //Wir warten die verbleibende Zeit, um auf
186         //unsere angegebenen FPS zu kommen
187         Core.Sleep(LOOP - codeElapsed);
188
189         //Vergangene Zeit + Wartezeit => Loop
190
191         //Da die Sleep-Genauigkeit schwankt, nochmal die vergangene Spielzeit
192         //vom letzten GameTick berechnen
193         loopElapsed = (int) (Core.Time - start); //Sleep dazurechnen
194       }
195     }
196
197     #endregion
198
199     #region Load, Reset, Next .. Level
200
201     /// <summary>
202     /// Daten laden
203     /// </summary>
204     private static void LoadData()
205     {
206       //Alle gefundenen Grafiken im Dataverzeichnis laden
207       foreach (string file in Directory.GetFiles("data", "*.png"))
208         Tileset.LoadTileset(file);
209     }
210
211     #endregion
212
```

```
213     #region Tick
214
215     /// <summary>
216     /// Game Tick
217     /// </summary>
218     private static void TickGame(int elapsed)
219     {
220       if (!Paused && World != null)
221       {
222         //"Huh? Where's our world? Gone?"
223         lock (m_World)
224         {
225           //AktuelleZeit - StartVomLetzenTick = Vergangene Zeit
226           World.Tick(elapsed);
227           GameTime += elapsed;
228         }
229       }
230     }
231
232     #endregion
233
234     #region Render
235
236     /// <summary>
237     /// Get Time String
238     /// </summary>
239     /// <param name="elapsed"></param>
240     /// <returns></returns>
241     private static string GetTimeString(int elapsed)
242     {
243       TimeSpan s = new TimeSpan(0, 0, 0, 0, elapsed);
244       return string.Format("{0:00}:{1:00}:{2:00}",
245                            s.Hours, s.Minutes, s.Seconds);
246     }
247
248     /// <summary>
249     /// Stellt die gesamte Zeichenfläche dar
250     /// "Alles-Neu-Render-Prinzip" wegen ..
251     /// a) Werten, welche sich ständig ändern
252     /// b) Scrolling
253     /// </summary>
254     /// <param name="t"></param>
255     private static void RenderGame(Texture t)
256     {
257       //"Emptyness!"
258       Color bg = Color.FromArgb(32, 32, 32);
259       Graphics g = t.Graphics;
260       g.Clear(bg);
261
262       #region Spielfeld
263
264       if (World != null)
265       {
266         lock (m_World)
267         {
268           //Viewport 2D
269           RectangleF vp2D = new RectangleF(VIEWPORT_OFFSET_X, VIEWPORT_OFFSET_Y, ViewportPixelsX,
                  ViewportPixelsY);
270
271           //Spielfeld nun rendern
272           //Dabei nur im Spielfenster zeichnen erlauben
273           g.SetClip(vp2D);
274           {
275             //Game
276             g.Clear(Color.Black);
277
278             //### Sichtfeld berechnen [Pixel] ###
279             float x = 0; //Top Left X
280             float y = 0; //Top Left Y
281
282             if (Controller != null)
283             {
284               //Center XY
285               float cx = Controller.RealX + Tile.TILE_WIDTH/2f;
286               float cy = Controller.RealY + Tile.TILE_HEIGHT/2f;
287               x = cx - vp2D.Width/2f;
288               y = cy - vp2D.Height/2f;
289             }
290
291             //Clippen
292             x = Utils.Bound(x, 0, Math.Max(0, World.Width*Tile.TILE_WIDTH - ViewportPixelsX));
293             y = Utils.Bound(y, 0, Math.Max(0, World.Height*Tile.TILE_HEIGHT - ViewportPixelsY));
294
295             //Falls die Welt kleiner als der Sichtbereich ist, zentrieren....
296             //- In der Facharbeitsversion nicht notwendig. -
297
298
299             //Spielfeld in die entgegen gesetzte Richtung verschieben
300             //2D Koordinaten - Int um Zeichenfehler zu vermeiden
301             int realX = (int) (vp2D.X - x);
302             int realY = (int) (vp2D.Y - y);
303             Point rel = new Point(realX, realY);
304
305             //### Sichbare Tiles berechnen [Tiles] ###
306             int xStart = (int) (x/Tile.TILE_WIDTH);
307             int yStart = (int) (y/Tile.TILE_HEIGHT);
308             int xEnd = xStart + ViewportTilesX + 1; //scrollen => überzeichnen
309             int yEnd = yStart + ViewportTilesY + 1;
310
311             //Clippen
312             xStart = Math.Max(0, xStart);
313             yStart = Math.Max(0, yStart);
314             xEnd = Math.Min(World.Width, xEnd);
315             yEnd = Math.Min(World.Height, yEnd);
316
317             //### Rendern ###
318             World.Render(t, rel, xStart, yStart, xEnd, yEnd, 0);
```

```
319             }
320             g.ResetClip();
321 
322             //Spielfeld Rand
323             g.DrawRectangle(Pens.White, vp2D.X - 1, vp2D.Y - 1, vp2D.Width + 2, vp2D.Height + 2);
324           }
325         }
326 
327         #endregion
328 
329         #region Texte
330 
331         //Linebreak (Zeilenumbruch)
332         const string LB = "\r\n";
333 
334         #region Oben
335 
336         //Links Oben (FPS + Copyright)
337         g.DrawString("Stufe IV" + LB +
338                      "FA Version" + LB +
339                      "© 2008/09 M. Linder",
340                      Fonts.Comic10, Brushes.White, 6, 4);
341 
342         //Oben (Titel)
343         const string n = "Boulderdash";
344         int w = (Graphic.Width/2) - (int) g.MeasureString(n, Fonts.Comic33).Width/2;
345         g.DrawString(n, Fonts.Comic33, Brushes.Gray, w + 2, 2 + 2); //Schatten
346         g.DrawString(n, Fonts.Comic33, Brushes.White, w, 2); //Front
347 
348         //Rechts Oben (Tastenbelegung)
349         g.DrawString("Enter - Pause" + LB +
350                      "Del - Reset" + LB +
351                      "Esc - Quit",
352                      Fonts.Comic10, Brushes.White, Graphic.Width - 100, 4);
353 
354         #endregion
355 
356         #region Unten
357 
358         //Unten
359         if (World != null)
360         {
361           g.DrawString(string.Format("Name: {0} -- Diamonds: d-{1:00} -- Time: {2}",
362                                      World.Name,
363                                      World.DiamondsRemaining,
364                                      GetTimeString(GameTime)),
365                        Fonts.Comic10, Brushes.White, 2, Graphic.Height - 2 - Fonts.Comic10.Height);
366         }
367 
368         #endregion
369 
370         #endregion
371       }
372 
373       #endregion
374 
375       #region Keyboard Control
376 
377       private static void OnKeyPress(object sender, PreviewKeyDownEventArgs e)
378       {
379         //Welche Taste wurde gedrückt?..
380         switch (e.KeyCode)
381         {
382             //######## Pause ########
383           case Keys.Enter:
384             {
385               DoPause();
386             }
387             break;
388             //######## Quit ########
389           case Keys.Escape:
390             {
391               DoQuit();
392             }
393             break;
394             //######## Random Level ########
395           case Keys.Delete:
396             {
397               DoResetLevel();
398             }
399             break;
400         }
401         //Spielersteuerung ist im Playerobjekt zu finden.
402       }
403 
404       #endregion
405 
406       #region Spielablauf Steuerung [Zugreifbar von ausserhalb]
407 
408       /// <summary>
409       /// Pause Key
410       /// </summary>
411       public static void DoPause()
412       {
413         Paused = true; //Messagebox hält den Thread an
414         MessageBox.Show("- Pause -", "Boulderdash", MessageBoxButtons.OK);
415         Paused = false;
416       }
417 
418       /// <summary>
419       /// Quit Key
420       /// </summary>
421       public static void DoQuit()
422       {
423         Deinit(); //Anwendung macht den Rest
424       }
425 
```

```
426     /// <summary>
427     /// Level resetten
428     /// </summary>
429     public static void DoResetLevel()
430     {
431       World = Level.GenerateRandomLevel(World.RandomSeed);
432     }
433
434     /// <summary>
435     /// Random Level
436     /// </summary>
437     public static void DoRandomLevel()
438     {
439       World = Level.GenerateRandomLevel(null);
440     }
441
442     #endregion
443   }
444 }
```

Game/World/Level.cs

```
1  using System;
2  using System.Drawing;
3  using BDCore.Helpers;
4  using BDGame.World.Interfaces;
5  using BDGame.World.Livings;
6  using BDGame.World.Solids;
7  using BDGame.World.Stones;
8
9  namespace BDGame.World
10 {
11   /// <summary>
12   /// Level
13   /// </summary>
14   [Serializable] //Speicherbar
15   internal sealed class Level
16   {
17     /// <summary>
18     /// Creates a new level
19     /// </summary>
20     /// <param name="w"></param>
21     /// <param name="h"></param>
22     /// <param name="name"></param>
23     public Level(int w, int h, string name)
24     {
25       Tiles = new GameObject[w,h];
26       Name = name;
27       Width = w;
28       Height = h;
29     }
30
31     ///<summary>
32     /// Name of the level
33     ///</summary>
34     public string Name { get; private set; }
35
36     ///<summary>
37     /// Breite des Levels
38     ///</summary>
39     public int Width { get; private set; }
40
41     ///<summary>
42     /// Hoehe des Levels
43     ///</summary>
44     public int Height { get; private set; }
45
46     /// <summary>
47     /// Tiles
48     /// (Objekte der Spielwelt)
49     /// </summary>
50     private GameObject[,] Tiles { get; set; }
51
52     /// <summary>
53     /// Diamonds Remaining
54     /// </summary>
55     public int DiamondsRemaining { get; set; }
56
57     /// <summary>
58     /// Zufallszahlenseed
59     /// </summary>
60     public int RandomSeed { get; set; }
61
62     /// <summary>
63     /// Gibt das Tile an der angegebenen Position wieder
64     /// Überladener Access-Operator
65     /// </summary>
66     /// <param name="x"></param>
67     /// <param name="y"></param>
68     /// <returns></returns>
69     public GameObject this[int x, int y]
70     {
71       get
72       {
73         x = Utils.Bound(x, 0, Width - 1);
74         y = Utils.Bound(y, 0, Height - 1);
75
76         return Tiles[x, y];
77       }
78       set
79       {
80         x = Utils.Bound(x, 0, Width - 1);
81         y = Utils.Bound(y, 0, Height - 1);
82
83         Tiles[x, y] = value;
```

```
84        }
85      }
86
87      /// <summary>
88      /// Game Tick
89      /// </summary>
90      /// <param name="elapsed"></param>
91      public void Tick(int elapsed)
92      {
93        //Spielwelt von Unten-Links nach Oben-Rechts durchgehen
94        //Da Steine nach unten fallen, sollten die unteren zuerst
95        //bewegt werden, um ein eintreten des Slippy-Effektes
96        //zu vermeiden
97        //foreach wegen notwendiger Durchlaufsrichtung nicht möglich
98        for (int y = Height - 1; y >= 0; y--)
99        {
100         for (int x = 0; x < Width; x++)
101         {
102           GameObject obj = Tiles[x, y];
103
104           if (obj == null || !obj.Active)
105             continue; //"No need to tick..!"
106
107           //ActiveObjects++;
108           obj.Tick(elapsed);
109         }
110       }
111     }
112
113     /// <summary>
114     /// Render Tick
115     /// </summary>
116     /// <param name="t"></param>
117     /// <param name="offset"></param>
118     /// <param name="xStart">Inclusive StartX</param>
119     /// <param name="yStart">Inclusive StartY</param>
120     /// <param name="xEnd">Exclusive EndX</param>
121     /// <param name="yEnd">Exclusive EndY</param>
122     /// <param name="elapsed">Not Implemented (Animations)</param>
123     public void Render(Texture t, Point offset, int xStart, int yStart, int xEnd, int yEnd, int
          elapsed)
124     {
125       //Spielwelt von Oben-Links nach Unten-Rechts durchgehen
126       for (int y = yStart; y < yEnd; y++)
127       {
128         for (int x = xStart; x < xEnd; x++)
129         {
130           GameObject obj = this[x, y];
131
132           if (obj == null)
133             continue; //"Nothing to render..!"
134
135           obj.Render(t, offset, elapsed);
136         }
137       }
138     }
139
140     /// <summary>
141     /// Add
142     /// Fügt ein Object zur Welt hinzu
143     /// </summary>
144     /// <param name="o"></param>
145     public void AddEditor(GameObject o)
146     {
147       lock (Tiles)
148       {
149         o.World = this;
150         o.Load(); //Sprites laden
151         Tiles[o.TileX, o.TileY] = o;
152       }
153     }
154
155     /// <summary>
156     /// Aktiviert Objekte in der Umgebung
157     /// </summary>
158     /// <param name="x"></param>
159     /// <param name="y"></param>
160     public void Activate(int x, int y)
161     {
162       //Betroffene Umgebung
163       //3x3
164       GameObject[] box3x3 = new[]
165                             {
166                               this[x - 1, y - 1],
167                               this[x, y - 1],
168                               this[x + 1, y - 1],
169                               this[x - 1, y],
170                               this[x, y],
171                               this[x + 1, y],
172                               this[x - 1, y + 1],
173                               this[x, y + 1],
174                               this[x + 1, y + 1],
175                             };
176
177       foreach (GameObject  o in box3x3)
178       {
179         if (o != null && o.CanActivate())
180           o.Activate();
181       }
182     }
183
184     /// <summary>
185     /// Prepare
186     /// Bereitet die Spielwelt vor
187     /// - Aktiviert alle Objekte
188     /// - Zählt alle Diamanten
189     /// </summary>
```

```
190     public void Prepare()
191     {
192       int dias = 0;
193
194       //Spielwelt aktivieren
195       for (int y = 0; y < Height; y++)
196       {
197         for (int x = 0; x < Width; x++)
198         {
199           GameObject obj = Tiles[x, y];
200
201           if (obj is ICollectable)
202             dias++;
203
204           if (obj == null || !obj.CanActivate())
205             continue; //"No need to activate..!"
206
207           //obj.Deactivate(); //ggf. erst deaktivieren
208           obj.Activate();
209         }
210       }
211
212       //Nur 90% notwendig - Jeden 10. Diamanten auslassen
213       //Subtrahieren, damit bei 9 Diamanten -> 9 / 10 = 0  -> kein Abzug
214       DiamondsRemaining = dias - dias/10;
215     }
216
217     /// <summary>
218     /// Setzt alle Objekte in einen inaktiven Zustand
219     /// </summary>
220     public void Unload()
221     {
222       for (int y = 0; y < Height; y++)
223       {
224         for (int x = 0; x < Width; x++)
225         {
226           GameObject obj = Tiles[x, y];
227
228           if (obj == null || !obj.Active)
229             continue; //"No need to deactivate..!"
230
231           obj.Deactivate();
232         }
233       }
234     }
235
236     /// <summary>
237     /// Zufallslevel
238     /// </summary>
239     /// <param name="seed">Nullbarer Startwert</param>
240     public static Level GenerateRandomLevel(int? seed)
241     {
242       //Falls der Seed nicht gesetzt ist, Systemzeit nehmen
243       int s = seed ?? Environment.TickCount;
244       Random rng = new Random(s);
245
246       //Level Generieren..
247       Level l = new Level(rng.Next(30, 100), rng.Next(30, 75), "Random Level ("+s+")");
248       l.RandomSeed = s;
249
250       //Sand
251       for (int i = 0; i < l.Width; i++)
252         for (int j = 0; j < l.Height; j++)
253           l.AddEditor(new Sand(i, j));
254
255       //Steine
256       int fac = (int) Math.Sqrt(l.Width*l.Height);
257
258       int steine = fac*10;
259       for (int i = 1; i <= steine; i++)
260         l.AddEditor(new Stone(rng.Next(1, l.Width - 1), rng.Next(1, l.Height - 1)));
261
262       //Leere Blöcke
263       int empty = fac*2;
264       for (int i = 1; i <= empty; i++)
265         l[rng.Next(1, l.Width - 1), rng.Next(1, l.Height - 1)] = null;
266
267       //Diamanten
268       int dias = fac*2/3;
269       for (int i = 1; i <= dias; i++)
270         l.AddEditor(new Diamond(rng.Next(1, l.Width - 1), rng.Next(1, l.Height - 1)));
271
272       //Rand
273       for (int lx = 0; lx < l.Width; lx++)
274       {
275         l.AddEditor(new Wall(lx, 0));
276         l.AddEditor(new Wall(lx, l.Height - 1));
277       }
278
279       for (int ly = 1; ly < l.Height - 1; ly++)
280       {
281         l.AddEditor(new Wall(0, ly));
282         l.AddEditor(new Wall(l.Width - 1, ly));
283       }
284
285       //Player
286       int pX = rng.Next(3, l.Width - 3);
287       int pY = rng.Next(3, l.Height - 3);
288
289       for (int i = pX - 1; i <= pX + 1; i++)
290         for (int j = pY - 1; j <= pY + 1; j++)
291           l.AddEditor(new Sand(i, j)); //Damit der Spieler sicher ist
292
293       l.AddEditor(new Player(pX, pY));
294
295       return l;
296     }
```

```
297   }
298 }
```

Game/World/GameObject.cs

```
  1 using System;
  2 using System.Drawing;
  3 using BDCore.Helpers;
  4
  5 namespace BDGame.World
  6 {
  7   /// <summary>
  8   /// Abstract Game Object
  9   /// </summary>
 10   internal abstract class GameObject
 11   {
 12     protected const int MAX_IDLE_TIME = 1000; //1 sec
 13
 14     /// <summary>
 15     /// Sprite
 16     /// </summary>
 17     private ISprite m_Sprite;
 18
 19     /// <summary>
 20     /// Abstract Constructor
 21     /// </summary>
 22     /// <param name="x"></param>
 23     /// <param name="y"></param>
 24     protected GameObject(int x, int y)
 25     {
 26       TileX = x;
 27       TileY = y;
 28     }
 29
 30     /// <summary>
 31     /// Idle Counter
 32     /// </summary>
 33     public int IdleCounter { get; protected set; }
 34
 35     /// <summary>
 36     /// Sprite (Darstellung)
 37     /// </summary>
 38     public ISprite Sprite
 39     {
 40       get { return m_Sprite; }
 41       protected set { m_Sprite = value; }
 42     }
 43
 44     /// <summary>
 45     /// Spielwelt
 46     /// </summary>
 47     public Level World { get; set; }
 48
 49     /// <summary>
 50     /// Aktuelle TileX Position
 51     /// </summary>
 52     public int TileX { get; protected set; }
 53
 54     /// <summary>
 55     /// Aktuelle TileY Position
 56     /// </summary>
 57     public int TileY { get; protected set; }
 58
 59     /// <summary>
 60     /// Aktiv?
 61     /// </summary>
 62     public bool Active { get; protected set; }
 63
 64     /// <summary>
 65     /// Tick
 66     /// </summary>
 67     /// <param name="elapsed"></param>
 68     public virtual void Tick(int elapsed)
 69     {
 70     }
 71
 72     /// <summary>
 73     /// Render
 74     /// </summary>
 75     /// <param name="t"></param>
 76     /// <param name="offset"></param>
 77     /// <param name="elapsed"></param>
 78     public virtual void Render(Texture t, Point offset, int elapsed)
 79     {
 80       if (Sprite != null)
 81       {
 82         int x = offset.X + TileX*Tile.TILE_WIDTH;
 83         int y = offset.Y + TileY*Tile.TILE_HEIGHT;
 84         Sprite.Render(t, elapsed, new Point(x, y));
 85       }
 86     }
 87
 88     /// <summary>
 89     /// Kann das Objekt aktiviert werden?
 90     /// </summary>
 91     /// <returns></returns>
 92     public virtual bool CanActivate()
 93     {
 94       return false;
 95     }
 96
 97     /// <summary>
 98     /// Deaktiviert das Objekt
 99     /// </summary>
100     public virtual void Deactivate()
```

```
101     {
102       Active = false;
103     }
104
105     /// <summary>
106     /// Activate
107     /// </summary>
108     public virtual void Activate()
109     {
110       Active = true;
111       ResetIdle();
112     }
113
114     /// <summary>
115     /// Lädt die für das Objekt wichtigen Daten (neu)
116     /// </summary>
117     public abstract void Load();
118
119     /// <summary>
120     /// Idle Counter
121     /// </summary>
122     public virtual void Idle(int elapsed)
123     {
124       if (!Active)
125         return;
126
127       IdleCounter -= elapsed;
128
129       if (IdleCounter <= 0)
130         Deactivate();
131     }
132
133     /// <summary>
134     /// Reset Idle Counter
135     /// </summary>
136     public virtual void ResetIdle()
137     {
138       IdleCounter = MAX_IDLE_TIME;
139     }
140
141     /// <summary>
142     /// Entfernt das Objekt aus der Spielwelt
143     /// </summary>
144     public virtual void Remove()
145     {
146       World.Activate(TileX, TileY);
147       World[TileX, TileY] = null;
148       Deactivate();
149       //"Goodbye."
150     }
151   }
152 }
```

Game/World/Interfaces/ICollectable.cs

```
1  namespace BDGame.World.Interfaces
2  {
3    /// <summary>
4    /// Fuer den Sieg einzusammelnder Gegenstand
5    /// </summary>
6    internal interface ICollectable
7    {
8    }
9  }
```

Game/World/Interfaces/ICrackable.cs

```
1  namespace BDGame.World.Interfaces
2  {
3    /// <summary>
4    /// Objekte, die auf Steinfall reagieren
5    /// </summary>
6    internal interface ICrackable
7    {
8      /// <summary>
9      /// Crack
10     /// "oeffnet" das Objekt
11     /// </summary>
12     void Crack();
13   }
14 }
```

Game/World/Interfaces/IDigable.cs

```
1  namespace BDGame.World.Interfaces
2  {
3    /// <summary>
4    /// Ausgrabbares Objekt
5    /// </summary>
6    internal interface IDigable
7    {
8      /// <summary>
9      /// Dig
10     /// Gräbt das Tile aus
11     /// </summary>
12     void Dig();
13   }
14 }
```

Game/World/Interfaces/IPickupable.cs

```
1  using BDGame.World.Livings;
2
3  namespace BDGame.World.Interfaces
4  {
5    /// <summary>
6    /// Aufnehmbares Item
7    /// </summary>
8    internal interface IPickupable
9    {
10     /// <summary>
11     /// Pickup
12     /// </summary>
13     void Pickup();
14   }
15 }
```

Game/World/Interfaces/IPushable.cs

```
1  namespace BDGame.World.Interfaces
2  {
3    /// <summary>
4    /// Schiebbarer Gegenstand
5    /// Nur auf Moveable anwendbar
6    /// </summary>
7    internal interface IPushable
8    {
9    }
10 }
```

Game/World/Abstract/AbstractMoveable.cs

```
1  using System;
2  using System.Drawing;
3  using BDCore.Helpers;
4  using BDGame.Helpers;
5
6  namespace BDGame.World.Abstract
7  {
8    /// <summary>
9    /// Bewegbares Objekt
10   /// </summary>
11   internal abstract class AbstractMoveable : GameObject
12   {
13     /// <summary>
14     /// Gibt die Geschwindigkeit einer Bewegung auf der X Achse an
15     /// </summary>
16     public const float X_UNITS_PER_SECOND = Tile.TILE_WIDTH*8;
17
18     /// <summary>
19     /// Gibt die Geschwindigkeit einer Bewegung auf der Y Achse an
20     /// </summary>
21     public const float Y_UNITS_PER_SECOND = Tile.TILE_HEIGHT*8;
22
23     protected AbstractMoveable(int x, int y) : base(x, y)
24     {
25       RealX = x*Tile.TILE_WIDTH;
26       RealY = y*Tile.TILE_HEIGHT;
27     }
28
29     /// <summary>
30     /// Real X Position
31     /// "genauere" X Position
32     /// </summary>
33     public float RealX { get; protected set; }
34
35     /// <summary>
36     /// Real Y Position
37     /// "genauere" Y Position
38     /// </summary>
39     public float RealY { get; protected set; }
40
41     /// <summary>
42     /// Remaining Movement X
43     /// Verbleibende Bewegung auf der X Achse
44     /// Absolut
45     /// </summary>
46     public float RemainingMovementX { get; protected set; }
47
48     /// <summary>
49     /// Remaining Movement Y
50     /// Verbleibende Bewegung auf der Y Achse
51     /// Absolut
52     /// </summary>
53     public float RemainingMovementY { get; protected set; }
54
55     /// <summary>
56     /// Aktuelle Richtung
57     /// </summary>
58     public Point Velocity { get; protected set; }
59
60     /// <summary>
61     /// Moving
62     /// Bewegt sich das Objekt?
63     /// </summary>
64     public bool Moving
65     {
66       get { return Velocity != Direction.None; }
67     }
68
69     /// <summary>
```

```
70      /// Move
71      /// Beginnt einen Bewegungsvorgang
72      /// </summary>
73      /// <param name="d"></param>
74      /// <returns></returns>
75      public bool Move(Point d)
76      {
77        if (Moving)
78          return false; //Wird schon bewegt
79
80        GameObject col;
81
82        //Kollisionserkennung
83        if (CheckCollision(d, out col))
84        {
85          if (col != null)
86            OnCollision(col, d);
87          return false;
88        }
89
90        MoveInternal(d);
91        return true;
92      }
93
94      /// <summary>
95      /// Move Internal
96      /// Vollendet die Bewegung (benoetigt fuer das Snapping)
97      /// </summary>
98      /// <param name="d"></param>
99      protected virtual void MoveInternal(Point d)
100     {
101       Velocity = d;
102       RemainingMovementX = Tile.TILE_WIDTH;
103       RemainingMovementY = Tile.TILE_HEIGHT;
104       MoveTile(d);
105     }
106
107     /// <summary>
108     /// Move Tile (Gamelogik)
109     /// Bewegt das Objekt in ein anderes Tile
110     /// </summary>
111     /// <param name="d"></param>
112     public void MoveTile(Point d)
113     {
114       //Sofort auf das neue Tile bewegen
115       //(notwendig, um das gewuenschte Kollisionsverhalten zu erzielen)
116       World[TileX, TileY] = null;
117       World[TileX + d.X, TileY + d.Y] = this;
118
119       //Alte Umgebung
120       World.Activate(TileX, TileY);
121
122       //Neue Umgebung
123       TileX += d.X;
124       TileY += d.Y;
125       World.Activate(TileX, TileY);
126     }
127
128     /// <summary>
129     /// Prueft, ob in der angegebenen Richtung
130     /// ein anderes Object vorhanden ist, oder
131     /// das Objekt gegen den Spielfeldrand stoeÄŸt
132     /// </summary>
133     /// <param name="vel"></param>
134     /// <param name="obj"></param>
135     /// <returns>True wenn eine Kollision stattfindet</returns>
136     public bool CheckCollision(Point vel, out GameObject obj)
137     {
138       //Randverhalten
139       int destX = TileX + vel.X;
140       int destY = TileY + vel.Y;
141
142       if (destX < 0 || destY < 0 || destX >= World.Width || destY >= World.Height)
143       {
144         obj = null;
145         return true;
146       }
147
148       obj = World[destX, destY];
149       return obj != null;
150     }
151
152     /// <summary>
153     /// OnCollision
154     /// Aufgerufen, wenn das aktuelle Objekt mit einem anderen Objekt kollidiert
155     /// (Nur bei Selbst initiierter Aktion)
156     /// </summary>
157     /// <param name="vel"></param>
158     /// <param name="o"></param>
159     protected virtual void OnCollision(GameObject o, Point vel)
160     {
161     }
162
163     public override void Tick(int elapsed)
164     {
165       //Bewegung (Darstellung)
166       if (Velocity != Direction.None)
167       {
168         //Distanz errechnen
169         float moveX = X_UNITS_PER_SECOND*elapsed/1000.0f;
170         float moveY = Y_UNITS_PER_SECOND*elapsed/1000.0f;
171
172         //Limitieren
173         moveX = Math.Min(moveX, RemainingMovementX);
174         moveY = Math.Min(moveY, RemainingMovementY);
175
176         //Limit "deakkumulieren"
```

```
177           RemainingMovementX -= moveX;
178           RemainingMovementY -= moveY;
179
180           //Bewegen
181           RealX += Velocity.X*moveX;
182           RealY += Velocity.Y*moveY;
183
184           //Ende der Bewegung?
185           if (RemainingMovementX <= 0 && RemainingMovementY <= 0)
186             Velocity = Direction.None;
187
188           ResetIdle();
189         }
190         else
191         {
192           //Idlezeit akkumulieren
193           Idle(elapsed);
194         }
195       }
196
197       public override void Render(Texture t, Point offset, int elapsed)
198       {
199         if (Sprite != null)
200         {
201           //Rendercode ueberschreiben, um genaueres Rendern zu ermoeglichen
202           int x = offset.X + (int) RealX;
203           int y = offset.Y + (int) RealY;
204           Sprite.Render(t, elapsed, new Point(x, y));
205         }
206       }
207
208       public override bool CanActivate()
209       {
210         return true;
211       }
212     }
213 }
```

Game/World/Abstract/AbstractLiving.cs

```
1  using System;
2  using BDGame.World.Interfaces;
3
4  namespace BDGame.World.Abstract
5  {
6    /// <summary>
7    /// Lebendes Objekt
8    /// </summary>
9    internal abstract class AbstractLiving : AbstractMoveable, ICrackable
10   {
11     protected AbstractLiving(int x, int y) : base(x, y)
12     {
13     }
14
15     #region ICrackable Members
16
17     public void Crack()
18     {
19       //Lebewesen sterben, wenn ihnen 'n Stein auf den Kopf prallt
20       Remove(); //ruft .Die() auf
21     }
22
23     #endregion
24
25     public override void Tick(int elapsed)
26     {
27       Think();
28
29       base.Tick(elapsed);
30     }
31
32     /// <summary>
33     /// "Denk"-Prozedur
34     /// </summary>
35     public abstract void Think();
36
37     public override void Idle(int elapsed)
38     {
39     } //Cant sleep
40
41     /// <summary>
42     /// "Stirb!" - Aufgerufen, nachdem das Objekt entfernt wurde
43     /// </summary>
44     public virtual void Die()
45     {
46     }
47
48     public override void Remove()
49     {
50       base.Remove();
51
52       //Call .Die() after removal to prevent recursive Remove() calls, because
53       //the object would still be in the world at the time of death
54       Die();
55     }
56   }
57 }
```

Game/World/Abstract/AbstractGravitable.cs

```
1  using System;
2  using BDGame.Helpers;
```

```
3
4  namespace BDGame.World.Abstract
5  {
6    /// <summary>
7    /// Object, welches unter Schwerkraft "leidet"
8    /// ("Today we are creative in matters of class names...")
9    /// </summary>
10   internal abstract class AbstractGravitable : AbstractMoveable
11   {
12     /// <summary>
13     /// Gravity Constant
14     /// Nirgendswo genutzt...
15     /// ...Aber warum eigentlich nicht..?
16     /// </summary>
17     public const float GRAVITY_CONST = 9.81f;
18
19     protected AbstractGravitable(int x, int y) : base(x, y)
20     {
21     }
22
23     /// <summary>
24     /// Wieviele Tiles ist der Stein bereits gefallen?
25     /// </summary>
26     protected int m_TilesFallen { get; private set; }
27
28     public override void Tick(int elapsed)
29     {
30       if (!Moving)
31       {
32         if (Move(Direction.Down)) //Versuchen, abwärts zu bewegen
33           m_TilesFallen++;
34         else m_TilesFallen = 0;
35       }
36
37       base.Tick(elapsed);
38     }
39   }
40 }
```

Game/World/Abstract/AbstractStone.cs

```
1  using System;
2  using System.Drawing;
3  using BDGame.Helpers;
4  using BDGame.World.Interfaces;
5
6  namespace BDGame.World.Abstract
7  {
8    /// <summary>
9    /// Abstract Stone
10   /// </summary>
11   internal abstract class AbstractStone : AbstractGravitable, IPushable
12   {
13     protected AbstractStone(int x, int y) : base(x, y)
14     {
15     }
16
17     protected override void OnCollision(GameObject o, Point vel)
18     {
19       base.OnCollision(o, vel);
20
21       //Fallbewegung
22       if (o != null && vel == Direction.Down)
23       {
24         //Crush & Crack
25         if (m_TilesFallen > 0) Crush(o);
26
27           //Slip (Abrutschen)
28         else Slip(o);
29       }
30     }
31
32     /// <summary>
33     /// Ausgelöst, wenn ein Stein auf ein anderes Objekt aufschlägt
34     /// </summary>
35     /// <param name="o"></param>
36     protected virtual void Crush(GameObject o)
37     {
38       if (o is ICrackable)
39         ((ICrackable) o).Crack();
40     }
41
42     /// <summary>
43     /// Abrutschen?
44     /// </summary>
45     /// <param name="o"></param>
46     protected virtual void Slip(GameObject o)
47     {
48       bool slippy = false;
49
50       if (o is AbstractStone) //Generell: Stones sind Slippy
51       {
52         //Wenn sich das untere Objekt noch bewegt,
53         //dann warten/nicht.
54         slippy = !((AbstractMoveable) o).Moving;
55       }
56
57       if (slippy)
58       {
59         //Prüfen, ob das darunter liegende Feld frei ist
60         //Erst Rechts, dann Links, da wir ein reproduzierbares
61         //Verhalten erwünschen
62         //Gegenläufig zum World Tick
63         GameObject o2;
64
```

```
65          Point dir = Direction.Right;
66          if (!CheckCollision(new Point(dir.X, dir.Y + 1), out o2)) //rechts unten
67            Move(dir);
68          dir = Direction.Left;
69          if (!Moving && !CheckCollision(new Point(dir.X, dir.Y + 1), out o2)) //links
70            Move(dir);
71        }
72      }
73    }
74  }
```

Game/World/Livings/Player.cs

```
1  using System;
2  using System.Drawing;
3  using System.Windows.Forms;
4  using BDCore;
5  using BDCore.Helpers;
6  using BDGame.Helpers;
7  using BDGame.World.Abstract;
8  using BDGame.World.Interfaces;
9
10 namespace BDGame.World.Livings
11 {
12   /// <summary>
13   /// Spielfigur
14   /// </summary>
15   internal class Player : AbstractLiving
16   {
17     private bool m_KeyDown;
18     private bool m_KeyLeft;
19     private bool m_KeyRight;
20     private bool m_KeyUp;
21
22     public Player(int x, int y) : base(x, y)
23     {
24     }
25
26     public override void Activate()
27     {
28       if (Active)
29         return;
30
31       base.Activate();
32       Keyboard.KeyPress += KeyPress;
33       Keyboard.KeyUp += KeyUp;
34       Game.Controller = this;
35     }
36
37     public override void Load()
38     {
39       Sprite = Tileset.Get("players", 1, 1);
40     }
41
42     private void KeyPress(object sender, PreviewKeyDownEventArgs e)
43     {
44       switch (e.KeyCode)
45       {
46         case Keys.Down:
47           m_KeyDown = true;
48           break;
49         case Keys.Up:
50           m_KeyUp = true;
51           break;
52         case Keys.Left:
53           m_KeyLeft = true;
54           break;
55         case Keys.Right:
56           m_KeyRight = true;
57           break;
58       }
59     }
60
61     private void KeyUp(object sender, KeyEventArgs e)
62     {
63       switch (e.KeyCode)
64       {
65         case Keys.Down:
66           m_KeyDown = false;
67           break;
68         case Keys.Up:
69           m_KeyUp = false;
70           break;
71         case Keys.Left:
72           m_KeyLeft = false;
73           break;
74         case Keys.Right:
75           m_KeyRight = false;
76           break;
77       }
78     }
79
80     public override void Deactivate()
81     {
82       if (!Active)
83         return;
84
85       Keyboard.KeyUp -= KeyUp;
86       Keyboard.KeyPress -= KeyPress;
87       base.Deactivate();
88     }
89
90     public override void Think()
91     {
92       if (!Moving)
```

```
93        {
94          //Sobald ein Move erfolgreich war, gehen die darauf folgenden Aufrufe
95          //ins "leere"
96
97          //Links und rechts haben Priorität (ausweichen)
98
99          //Beim Bewegen nach Oben können am meisten Kollisionen
100         //Mit Steinen passieren, so dass wir dieses als letztes Prüfen
101         if (m_KeyLeft)
102           Move(Direction.Left);
103         else if (m_KeyRight)
104           Move(Direction.Right);
105         else if (m_KeyDown)
106           Move(Direction.Down);
107         else if (m_KeyUp)
108           Move(Direction.Up);
109       }
110     }
111
112     protected override void OnCollision(GameObject o, Point vel)
113     {
114       base.OnCollision(o, vel);
115
116       //Lücke füllen? (nach Entfernen eines Gegendstandes)
117       bool fillGap = false;
118
119       //Sand ausgraben
120       if (o is IDigable)
121       {
122         ((IDigable) o).Dig();
123         fillGap = true;
124       }
125         //Items? (Diamanten, ..)
126       else if (o is IPickupable)
127       {
128         ((IPickupable) o).Pickup();
129         (o).Remove();
130
131         //In das nun leere Feld bewegen
132         fillGap = true;
133
134         //Siegbedingung
135         if (o is ICollectable)
136           CheckWin();
137       }
138         //Steine schieben?
139       else if (o is IPushable && o is AbstractMoveable && (vel == Direction.Left || vel ==
              Direction.Right))
140       {
141         AbstractMoveable m = (AbstractMoveable) o;
142         m.Tick(0); //Prüft, ob der Stein fallen kann (Sonst würde man ihn durch die Luft verschieben
              können)
143         if (!m.Moving)
144         {
145           m.Move(vel);
146           fillGap = true;
147         }
148       }
149
150       if (fillGap)
151       {
152         if (!CheckCollision(vel, out o))
153           Move(vel); //Lücke schließen
154       }
155     }
156
157     public override void Die()
158     {
159       MessageBox.Show("You just died!", "Boulderdash", MessageBoxButtons.OK, MessageBoxIcon.Error);
160       base.Die();
161
162       Game.DoResetLevel();
163     }
164
165     /// <summary>
166     /// Check Winning Condition
167     /// </summary>
168     public void CheckWin()
169     {
170       if (World.DiamondsRemaining <= 0)
171       {
172         MessageBox.Show("You did it!\r\nYou found enough Diamonds!", "Boulderdash",
              MessageBoxButtons.OK,
173                         MessageBoxIcon.Exclamation);
174         Game.DoRandomLevel();
175       }
176     }
177   }
178 }
```

Game/World/Solids/Sand.cs

```
1  using System;
2  using BDCore.Helpers;
3  using BDGame.World.Interfaces;
4
5  namespace BDGame.World.Solids
6  {
7    /// <summary>
8    /// Sand
9    /// </summary>
10   internal class Sand : GameObject, IDigable
11   {
12     /// <summary>
13     /// Init
```

```
14      /// </summary>
15      /// <param name="x"></param>
16      /// <param name="y"></param>
17      public Sand(int x, int y) : base(x, y)
18      {
19      }
20
21      #region IDigable Members
22
23      public void Dig()
24      {
25        Remove();
26      }
27
28      #endregion
29
30      public override void Tick(int elapsed)
31      {
32      }
33
34      public override bool CanActivate()
35      {
36        return false;
37      }
38
39      public override void Load()
40      {
41        //Textur variieren
42        Sprite = Tileset.Get("sands", Utils.Random(1, 3), Utils.Random(1, 3));
43      }
44    }
45  }
```

Game/World/Solids/Wall.cs

```
1   using BDCore.Helpers;
2
3   namespace BDGame.World.Solids
4   {
5     /// <summary>
6     /// Wall
7     /// </summary>
8     internal class Wall : GameObject
9     {
10      public Wall(int x, int y) : base(x, y)
11      {
12      }
13
14      public override void Load()
15      {
16        Sprite = Tileset.Get("walls", 1, 1);
17      }
18    }
19  }
```

Game/World/Stones/Stone.cs

```
1   using System;
2   using BDCore.Helpers;
3   using BDGame.World.Abstract;
4
5   namespace BDGame.World.Stones
6   {
7     /// <summary>
8     /// Stone
9     /// </summary>
10    internal class Stone : AbstractStone
11    {
12      public Stone(int x, int y) : base(x, y)
13      {
14      }
15
16      public override void Load()
17      {
18        Sprite = Tileset.Get("stones", 1, 1);
19      }
20    }
21  }
```

Game/World/Stones/Diamond.cs

```
1   using System;
2   using BDCore.Helpers;
3   using BDGame.World.Abstract;
4   using BDGame.World.Interfaces;
5   using BDGame.World.Livings;
6
7   namespace BDGame.World.Stones
8   {
9     /// <summary>
10    /// Einsammelbarer Diamant
11    /// </summary>
12    internal class Diamond : AbstractStone, IPickupable, ICollectable
13    {
14      public Diamond(int x, int y) : base(x, y)
15      {
16      }
17
18      #region IPickupable Members
```

```
19
20      public void Pickup()
21      {
22        World.DiamondsRemaining--;
23      }
24
25      #endregion
26
27      public override void Load()
28      {
29        Sprite = Tileset.Get("diamonds", 1, 1);
30      }
31    }
32  }
```

Game/Helpers/Direction.cs

```
1   using System;
2   using System.Drawing;
3   using BDCore.Helpers;
4
5   namespace BDGame.Helpers
6   {
7     /// <summary>
8     /// Richtungskontainerklasse. Bietet Richtungsvektoren
9     /// </summary>
10    internal static class Direction
11    {
12      public static readonly Point Down = new Point(0, 1);
13      public static readonly Point Left = new Point(-1, 0);
14      public static readonly Point None = new Point(0, 0);
15      public static readonly Point Right = new Point(1, 0);
16      public static readonly Point Up = new Point(0, -1);
17
18      /// <summary>
19      /// Gibt die "nächste" Richtung wieder (Uhrzeigersinn)
20      /// </summary>
21      /// <param name="old"></param>
22      /// <param name="increment"></param>
23      /// <returns></returns>
24      public static Point GetDirection(Point old, int increment)
25      {
26        increment = increment%4; //4 mögliche Richtungen
27
28        if (increment < 0) //Nur positive Winkel
29          increment = 4 - Math.Abs(increment);
30
31        while (increment-- > 0)
32        {
33          if (old == Right || old == None)
34            old = Down;
35          else if (old == Down)
36            old = Left;
37          else if (old == Left)
38            old = Up;
39          else if (old == Up)
40            old = Right;
41        }
42
43        return old;
44      }
45
46      /// <summary>
47      /// Gibt eine zufällige Bewegungsrichtung wieder
48      /// </summary>
49      /// <returns></returns>
50      public static Point GetRandom()
51      {
52        switch (Utils.Random(1, 4))
53        {
54          case 1:
55            return Up;
56          case 2:
57            return Down;
58          case 3:
59            return Left;
60          case 4:
61            return Right;
62        }
63        return None;
64      }
65    }
66  }
```

Quellcode

- Boulderdash.exe -

Boulderdash/Program.cs

```
1  using System;
2  using System.IO;
3  using System.Threading;
4  using System.Windows.Forms;
5  using BDCore;
6  using BDGame;
7
8  namespace Boulderdash
9  {
10   /// <summary>
11   /// Hauptprogrammklasse
12   /// </summary>
13   internal static class Program
14   {
15     /// <summary>
16     /// Haupteinstiegspunkt (PSVM)
17     /// </summary>
18     /// <param name="args">Command Line Args</param>
19     [MTAThread]
20     public static void Main(params string[] args)
21     {
22       //Fehler Behandlung (Background Threads)
23       Application.ThreadException += ((sender, e) => OnException("Thread Exception", e.Exception));
24       AppDomain.CurrentDomain.UnhandledException += ((sender, e) => OnException("Unhandled Exception
             ", e.ExceptionObject));
25
26       //Startverzeichnis
27       Environment.CurrentDirectory = Path.GetDirectoryName(Application.ExecutablePath);
28
29       //Fehler Behandlung (Hauptthread)
30       try
31       {
32         //Standard Einstellungen für Windows XP vornehmen
33         Application.EnableVisualStyles();
34         Application.SetCompatibleTextRenderingDefault(false);
35
36         //Kern initialisieren
37         Core.Init(new MainForm().GameControl);
38
39         //Spiel starten
40         Game.Init();
41
42         //Hauptthread in einen "Warte-Zustand" versetzen
43         while (Core.Running && Game.Running && Core.FormShown) Thread.Sleep(1000);
44
45         //Spiel denitialisieren
46         Game.Deinit();
47
48         //Kern deinitialisieren
49         Core.Deinit();
50       }
51       catch (Exception e)
52       {
53         OnException("Fatal Exception", e);
54       }
55     }
56
57     /// <summary>
58     /// On Exception
59     /// </summary>
60     /// <param name="caption"></param>
61     /// <param name="ex"></param>
62     public static void OnException(string caption, object ex)
63     {
64       try
65       {
66         //Wir wollen nicht eine Exception, welche eine neue Exception auslöst...
67         MessageBox.Show((ex ?? "(null)").ToString(), caption, MessageBoxButtons.OK, MessageBoxIcon.
               Error);
68       }
69       finally
70       {
71         Environment.Exit(caption.GetHashCode());
72       }
73     }
74   }
75 }
```

Boulderdash/MainForm.cs

```
1  using System;
2  using System.Windows.Forms;
3  using BDCore;
4  using BDGame;
5
6  namespace Boulderdash
7  {
8    /// <summary>
9    /// Mainform
10   /// Erstellt mit dem Formdesigner
11   /// </summary>
12   internal partial class MainForm : Form
13   {
14     public MainForm()
15     {
16       InitializeComponent();
17     }
18
19     /// <summary>
20     /// Game Control
21     /// </summary>
22     public GameControl GameControl
23     {
```

```
24        get { return gameControl; }
25      }
26
27      private void MainForm_Shown(object sender, EventArgs e)
28      {
29        BringToFront();
30        Focus();
31      }
32
33      private void pauseResumeToolStripMenuItem_Click(object sender, EventArgs e)
34      {
35        Game.DoPause();
36      }
37
38      private void quitToolStripMenuItem_Click(object sender, EventArgs e)
39      {
40        Game.DoQuit();
41      }
42
43      private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
44      {
45        new AboutBD().ShowDialog();
46      }
47
48      private void randomLevelToolStripMenuItem_Click(object sender, EventArgs e)
49      {
50        Game.DoRandomLevel();
51      }
52
53      private void resetLevelToolStripMenuItem_Click(object sender, EventArgs e)
54      {
55        Game.DoResetLevel();
56      }
57    }
58  }
```

Boulderdash/MainForm.Designer.cs

```
1   using BDCore;
2
3   namespace Boulderdash
4   {
5     partial class MainForm
6     {
7       /// <summary>
8       /// Required designer variable.
9       /// </summary>
10      private System.ComponentModel.IContainer components = null;
11
12      /// <summary>
13      /// Clean up any resources being used.
14      /// </summary>
15      /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</
            param>
16      protected override void Dispose(bool disposing)
17      {
18        if (disposing && (components != null))
19        {
20          components.Dispose();
21        }
22        base.Dispose(disposing);
23      }
24
25      #region Windows Form Designer generated code
26
27      /// <summary>
28      /// Required method for Designer support - do not modify
29      /// the contents of this method with the code editor.
30      /// </summary>
31      private void InitializeComponent()
32      {
33        System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.
              ComponentResourceManager(typeof(MainForm));
34        this.mainMenu = new System.Windows.Forms.MenuStrip();
35        this.gameToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
36        this.randomLevelToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
37        this.toolStripMenuItem1 = new System.Windows.Forms.ToolStripSeparator();
38        this.pauseResumeToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
39        this.toolStripMenuItem2 = new System.Windows.Forms.ToolStripSeparator();
40        this.quitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
41        this.miscToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
42        this.aboutToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
43        this.gameControl = new BDCore.GameControl();
44        this.resetLevelToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
45        this.mainMenu.SuspendLayout();
46        this.SuspendLayout();
47        //
48        // mainMenu
49        //
50        this.mainMenu.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
51              this.gameToolStripMenuItem,
52              this.miscToolStripMenuItem});
53        this.mainMenu.Location = new System.Drawing.Point(0, 0);
54        this.mainMenu.Name = "mainMenu";
55        this.mainMenu.RenderMode = System.Windows.Forms.ToolStripRenderMode.Professional;
56        this.mainMenu.Size = new System.Drawing.Size(802, 24);
57        this.mainMenu.TabIndex = 1;
58        this.mainMenu.Text = "mainMenu";
59        //
60        // gameToolStripMenuItem
61        //
62        this.gameToolStripMenuItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Text
              ;
63        this.gameToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
64              this.randomLevelToolStripMenuItem,
```

```
65              this.toolStripMenuItem1,
66              this.pauseResumeToolStripMenuItem,
67              this.resetLevelToolStripMenuItem,
68              this.toolStripMenuItem2,
69              this.quitToolStripMenuItem});
70          this.gameToolStripMenuItem.Name = "gameToolStripMenuItem";
71          this.gameToolStripMenuItem.Size = new System.Drawing.Size(46, 20);
72          this.gameToolStripMenuItem.Text = "&Game";
73          //
74          // randomLevelToolStripMenuItem
75          //
76          this.randomLevelToolStripMenuItem.Name = "randomLevelToolStripMenuItem";
77          this.randomLevelToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
78          this.randomLevelToolStripMenuItem.Text = "&Random Level";
79          this.randomLevelToolStripMenuItem.Click += new System.EventHandler(this.
                randomLevelToolStripMenuItem_Click);
80          //
81          // toolStripMenuItem1
82          //
83          this.toolStripMenuItem1.Name = "toolStripMenuItem1";
84          this.toolStripMenuItem1.Size = new System.Drawing.Size(149, 6);
85          //
86          // pauseResumeToolStripMenuItem
87          //
88          this.pauseResumeToolStripMenuItem.Name = "pauseResumeToolStripMenuItem";
89          this.pauseResumeToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
90          this.pauseResumeToolStripMenuItem.Text = "&Pause";
91          this.pauseResumeToolStripMenuItem.Click += new System.EventHandler(this.
                pauseResumeToolStripMenuItem_Click);
92          //
93          // toolStripMenuItem2
94          //
95          this.toolStripMenuItem2.Name = "toolStripMenuItem2";
96          this.toolStripMenuItem2.Size = new System.Drawing.Size(149, 6);
97          //
98          // quitToolStripMenuItem
99          //
100         this.quitToolStripMenuItem.Name = "quitToolStripMenuItem";
101         this.quitToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
102         this.quitToolStripMenuItem.Text = "&Quit";
103         this.quitToolStripMenuItem.Click += new System.EventHandler(this.quitToolStripMenuItem_Click
                );
104         //
105         // miscToolStripMenuItem
106         //
107         this.miscToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
108             this.aboutToolStripMenuItem});
109         this.miscToolStripMenuItem.Name = "miscToolStripMenuItem";
110         this.miscToolStripMenuItem.Size = new System.Drawing.Size(39, 20);
111         this.miscToolStripMenuItem.Text = "&Misc";
112         //
113         // aboutToolStripMenuItem
114         //
115         this.aboutToolStripMenuItem.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.
                Text;
116         this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
117         this.aboutToolStripMenuItem.Size = new System.Drawing.Size(114, 22);
118         this.aboutToolStripMenuItem.Text = "&About";
119         this.aboutToolStripMenuItem.Click += new System.EventHandler(this.
                aboutToolStripMenuItem_Click);
120         //
121         // gameControl
122         //
123         this.gameControl.BackColor = System.Drawing.Color.DimGray;
124         this.gameControl.Dock = System.Windows.Forms.DockStyle.Fill;
125         this.gameControl.Location = new System.Drawing.Point(0, 24);
126         this.gameControl.MinimumSize = new System.Drawing.Size(800, 600);
127         this.gameControl.Name = "gameControl";
128         this.gameControl.Size = new System.Drawing.Size(802, 600);
129         this.gameControl.TabIndex = 0;
130         this.gameControl.Text = "gameControl";
131         //
132         // resetLevelToolStripMenuItem
133         //
134         this.resetLevelToolStripMenuItem.Name = "resetLevelToolStripMenuItem";
135         this.resetLevelToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
136         this.resetLevelToolStripMenuItem.Text = "Reset &Level";
137         this.resetLevelToolStripMenuItem.Click += new System.EventHandler(this.
                resetLevelToolStripMenuItem_Click);
138         //
139         // MainForm
140         //
141         this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
142         this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
143         this.ClientSize = new System.Drawing.Size(802, 624);
144         this.Controls.Add(this.gameControl);
145         this.Controls.Add(this.mainMenu);
146         this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
147         this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
148         this.MainMenuStrip = this.mainMenu;
149         this.MaximizeBox = false;
150         this.MinimumSize = new System.Drawing.Size(808, 655);
151         this.Name = "MainForm";
152         this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
153         this.Text = "Boulderdash";
154         this.Shown += new System.EventHandler(this.MainForm_Shown);
155         this.mainMenu.ResumeLayout(false);
156         this.mainMenu.PerformLayout();
157         this.ResumeLayout(false);
158         this.PerformLayout();
159
160     }
161
162     #endregion
163
164         private System.Windows.Forms.MenuStrip mainMenu;
165         private System.Windows.Forms.ToolStripMenuItem gameToolStripMenuItem;
```

```
166         private System.Windows.Forms.ToolStripSeparator toolStripMenuItem1;
167         private System.Windows.Forms.ToolStripMenuItem quitToolStripMenuItem;
168         private System.Windows.Forms.ToolStripMenuItem miscToolStripMenuItem;
169         private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
170         private System.Windows.Forms.ToolStripMenuItem pauseResumeToolStripMenuItem;
171         private System.Windows.Forms.ToolStripSeparator toolStripMenuItem2;
172         private GameControl gameControl;
173         private System.Windows.Forms.ToolStripMenuItem randomLevelToolStripMenuItem;
174         private System.Windows.Forms.ToolStripMenuItem resetLevelToolStripMenuItem;
175
176     }
177 }
```

Boulderdash/AboutBD.cs

```
1  using System;
2  using System.IO;
3  using System.Reflection;
4  using System.Windows.Forms;
5
6  namespace Boulderdash
7  {
8    /// <summary>
9    /// About Box
10   /// Vom Designer generiert
11   /// </summary>
12   internal partial class AboutBD : Form
13   {
14     public AboutBD()
15     {
16       InitializeComponent();
17       Text = String.Format("About␣{0}", AssemblyTitle);
18       labelProductName.Text = AssemblyProduct;
19       labelVersion.Text = String.Format("Version␣{0}", AssemblyVersion);
20       labelCopyright.Text = AssemblyCopyright;
21       labelCompanyName.Text = AssemblyCompany;
22       textBoxDescription.Text = AssemblyDescription;
23     }
24
25     #region Assembly Attribute Accessors
26
27     public string AssemblyTitle
28     {
29       get
30       {
31         object[] attributes = Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(
               AssemblyTitleAttribute), false);
32         if (attributes.Length > 0)
33         {
34           AssemblyTitleAttribute titleAttribute = (AssemblyTitleAttribute) attributes[0];
35           if (titleAttribute.Title != "")
36             return titleAttribute.Title;
37         }
38         return Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().CodeBase);
39       }
40     }
41
42     public string AssemblyVersion
43     {
44       get { return Assembly.GetExecutingAssembly().GetName().Version.ToString(); }
45     }
46
47     public string AssemblyDescription
48     {
49       get
50       {
51         object[] attributes = Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(
               AssemblyDescriptionAttribute),
52                                                                                   false);
53         if (attributes.Length == 0)
54           return "";
55         return ((AssemblyDescriptionAttribute) attributes[0]).Description;
56       }
57     }
58
59     public string AssemblyProduct
60     {
61       get
62       {
63         object[] attributes = Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(
               AssemblyProductAttribute), false);
64         if (attributes.Length == 0)
65           return "";
66         return ((AssemblyProductAttribute) attributes[0]).Product;
67       }
68     }
69
70     public string AssemblyCopyright
71     {
72       get
73       {
74         object[] attributes = Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(
               AssemblyCopyrightAttribute), false);
75         if (attributes.Length == 0)
76           return "";
77         return ((AssemblyCopyrightAttribute) attributes[0]).Copyright;
78       }
79     }
80
81     public string AssemblyCompany
82     {
83       get
84       {
85         object[] attributes = Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(
               AssemblyCompanyAttribute), false);
```

```
86          if (attributes.Length == 0)
87            return "";
88          return ((AssemblyCompanyAttribute) attributes[0]).Company;
89        }
90      }
91
92      #endregion
93    }
94  }
```

Boulderdash/AboutBD.Designer.cs

```
1  namespace Boulderdash
2  {
3      partial class AboutBD
4      {
5          /// <summary>
6          /// Required designer variable.
7          /// </summary>
8          private System.ComponentModel.IContainer components = null;
9
10         /// <summary>
11         /// Clean up any resources being used.
12         /// </summary>
13         protected override void Dispose(bool disposing)
14         {
15             if (disposing && (components != null))
16             {
17                 components.Dispose();
18             }
19             base.Dispose(disposing);
20         }
21
22         #region Windows Form Designer generated code
23
24         /// <summary>
25         /// Required method for Designer support - do not modify
26         /// the contents of this method with the code editor.
27         /// </summary>
28         private void InitializeComponent()
29         {
30             System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.
                   ComponentResourceManager(typeof(AboutBD));
31             this.tableLayoutPanel = new System.Windows.Forms.TableLayoutPanel();
32             this.logoPictureBox = new System.Windows.Forms.PictureBox();
33             this.labelProductName = new System.Windows.Forms.Label();
34             this.labelVersion = new System.Windows.Forms.Label();
35             this.labelCopyright = new System.Windows.Forms.Label();
36             this.labelCompanyName = new System.Windows.Forms.Label();
37             this.textBoxDescription = new System.Windows.Forms.TextBox();
38             this.okButton = new System.Windows.Forms.Button();
39             this.tableLayoutPanel.SuspendLayout();
40             ((System.ComponentModel.ISupportInitialize)(this.logoPictureBox)).BeginInit();
41             this.SuspendLayout();
42             //
43             // tableLayoutPanel
44             //
45             this.tableLayoutPanel.ColumnCount = 2;
46             this.tableLayoutPanel.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.
                   Windows.Forms.SizeType.Percent, 33F));
47             this.tableLayoutPanel.ColumnStyles.Add(new System.Windows.Forms.ColumnStyle(System.
                   Windows.Forms.SizeType.Percent, 67F));
48             this.tableLayoutPanel.Controls.Add(this.logoPictureBox, 0, 0);
49             this.tableLayoutPanel.Controls.Add(this.labelProductName, 1, 0);
50             this.tableLayoutPanel.Controls.Add(this.labelVersion, 1, 1);
51             this.tableLayoutPanel.Controls.Add(this.labelCopyright, 1, 2);
52             this.tableLayoutPanel.Controls.Add(this.labelCompanyName, 1, 3);
53             this.tableLayoutPanel.Controls.Add(this.textBoxDescription, 1, 4);
54             this.tableLayoutPanel.Controls.Add(this.okButton, 1, 5);
55             this.tableLayoutPanel.Dock = System.Windows.Forms.DockStyle.Fill;
56             this.tableLayoutPanel.Location = new System.Drawing.Point(9, 9);
57             this.tableLayoutPanel.Name = "tableLayoutPanel";
58             this.tableLayoutPanel.RowCount = 6;
59             this.tableLayoutPanel.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.
                   Forms.SizeType.Percent, 10F));
60             this.tableLayoutPanel.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.
                   Forms.SizeType.Percent, 10F));
61             this.tableLayoutPanel.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.
                   Forms.SizeType.Percent, 10F));
62             this.tableLayoutPanel.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.
                   Forms.SizeType.Percent, 10F));
63             this.tableLayoutPanel.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.
                   Forms.SizeType.Percent, 50F));
64             this.tableLayoutPanel.RowStyles.Add(new System.Windows.Forms.RowStyle(System.Windows.
                   Forms.SizeType.Percent, 10F));
65             this.tableLayoutPanel.Size = new System.Drawing.Size(417, 265);
66             this.tableLayoutPanel.TabIndex = 0;
67             //
68             // logoPictureBox
69             //
70             this.logoPictureBox.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
71             this.logoPictureBox.Dock = System.Windows.Forms.DockStyle.Fill;
72             this.logoPictureBox.Image = ((System.Drawing.Image)(resources.GetObject("logoPictureBox
                   .Image")));
73             this.logoPictureBox.Location = new System.Drawing.Point(3, 3);
74             this.logoPictureBox.Name = "logoPictureBox";
75             this.tableLayoutPanel.SetRowSpan(this.logoPictureBox, 6);
76             this.logoPictureBox.Size = new System.Drawing.Size(131, 259);
77             this.logoPictureBox.TabIndex = 12;
78             this.logoPictureBox.TabStop = false;
79             //
80             // labelProductName
81             //
82             this.labelProductName.Dock = System.Windows.Forms.DockStyle.Fill;
```

```
83              this.labelProductName.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F,
                    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
84              this.labelProductName.Location = new System.Drawing.Point(143, 0);
85              this.labelProductName.Margin = new System.Windows.Forms.Padding(6, 0, 3, 0);
86              this.labelProductName.MaximumSize = new System.Drawing.Size(0, 17);
87              this.labelProductName.Name = "labelProductName";
88              this.labelProductName.Size = new System.Drawing.Size(271, 17);
89              this.labelProductName.TabIndex = 19;
90              this.labelProductName.Text = "Product Name";
91              this.labelProductName.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
92              //
93              // labelVersion
94              //
95              this.labelVersion.Dock = System.Windows.Forms.DockStyle.Fill;
96              this.labelVersion.Font = new System.Drawing.Font("Arial", 8.25F, ((System.Drawing.
                    FontStyle)((System.Drawing.FontStyle.Bold | System.Drawing.FontStyle.Italic))),
                    System.Drawing.GraphicsUnit.Point, ((byte)(0)));
97              this.labelVersion.Location = new System.Drawing.Point(143, 26);
98              this.labelVersion.Margin = new System.Windows.Forms.Padding(6, 0, 3, 0);
99              this.labelVersion.MaximumSize = new System.Drawing.Size(0, 17);
100             this.labelVersion.Name = "labelVersion";
101             this.labelVersion.Size = new System.Drawing.Size(271, 17);
102             this.labelVersion.TabIndex = 0;
103             this.labelVersion.Text = "Version";
104             this.labelVersion.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
105             //
106             // labelCopyright
107             //
108             this.labelCopyright.Dock = System.Windows.Forms.DockStyle.Fill;
109             this.labelCopyright.Font = new System.Drawing.Font("Arial", 8.25F, System.Drawing.
                    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
110             this.labelCopyright.Location = new System.Drawing.Point(143, 52);
111             this.labelCopyright.Margin = new System.Windows.Forms.Padding(6, 0, 3, 0);
112             this.labelCopyright.MaximumSize = new System.Drawing.Size(0, 17);
113             this.labelCopyright.Name = "labelCopyright";
114             this.labelCopyright.Size = new System.Drawing.Size(271, 17);
115             this.labelCopyright.TabIndex = 21;
116             this.labelCopyright.Text = "Copyright";
117             this.labelCopyright.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
118             //
119             // labelCompanyName
120             //
121             this.labelCompanyName.Dock = System.Windows.Forms.DockStyle.Fill;
122             this.labelCompanyName.Font = new System.Drawing.Font("Arial", 8.25F, System.Drawing.
                    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
123             this.labelCompanyName.Location = new System.Drawing.Point(143, 78);
124             this.labelCompanyName.Margin = new System.Windows.Forms.Padding(6, 0, 3, 0);
125             this.labelCompanyName.MaximumSize = new System.Drawing.Size(0, 17);
126             this.labelCompanyName.Name = "labelCompanyName";
127             this.labelCompanyName.Size = new System.Drawing.Size(271, 17);
128             this.labelCompanyName.TabIndex = 22;
129             this.labelCompanyName.Text = "Company Name";
130             this.labelCompanyName.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
131             //
132             // textBoxDescription
133             //
134             this.textBoxDescription.Dock = System.Windows.Forms.DockStyle.Fill;
135             this.textBoxDescription.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
                    System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
136             this.textBoxDescription.Location = new System.Drawing.Point(143, 107);
137             this.textBoxDescription.Margin = new System.Windows.Forms.Padding(6, 3, 3, 3);
138             this.textBoxDescription.Multiline = true;
139             this.textBoxDescription.Name = "textBoxDescription";
140             this.textBoxDescription.ReadOnly = true;
141             this.textBoxDescription.ScrollBars = System.Windows.Forms.ScrollBars.Both;
142             this.textBoxDescription.Size = new System.Drawing.Size(271, 126);
143             this.textBoxDescription.TabIndex = 23;
144             this.textBoxDescription.TabStop = false;
145             this.textBoxDescription.Text = "Description";
146             //
147             // okButton
148             //
149             this.okButton.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.
                    AnchorStyles.Bottom | System.Windows.Forms.AnchorStyles.Right)));
150             this.okButton.DialogResult = System.Windows.Forms.DialogResult.Cancel;
151             this.okButton.Location = new System.Drawing.Point(339, 239);
152             this.okButton.Name = "okButton";
153             this.okButton.Size = new System.Drawing.Size(75, 23);
154             this.okButton.TabIndex = 24;
155             this.okButton.Text = "&OK";
156             //
157             // AboutBD
158             //
159             this.AcceptButton = this.okButton;
160             this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
161             this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
162             this.ClientSize = new System.Drawing.Size(435, 283);
163             this.Controls.Add(this.tableLayoutPanel);
164             this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
165             this.MaximizeBox = false;
166             this.MinimizeBox = false;
167             this.Name = "AboutBD";
168             this.Padding = new System.Windows.Forms.Padding(9);
169             this.ShowIcon = false;
170             this.ShowInTaskbar = false;
171             this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
172             this.Text = "AboutBD";
173             this.tableLayoutPanel.ResumeLayout(false);
174             this.tableLayoutPanel.PerformLayout();
175             ((System.ComponentModel.ISupportInitialize)(this.logoPictureBox)).EndInit();
176             this.ResumeLayout(false);
177
178         }
179
180         #endregion
181
182         private System.Windows.Forms.TableLayoutPanel tableLayoutPanel;
```

```
183             private System.Windows.Forms.PictureBox logoPictureBox;
184             private System.Windows.Forms.Label labelProductName;
185             private System.Windows.Forms.Label labelVersion;
186             private System.Windows.Forms.Label labelCopyright;
187             private System.Windows.Forms.Label labelCompanyName;
188             private System.Windows.Forms.TextBox textBoxDescription;
189             private System.Windows.Forms.Button okButton;
190         }
191     }
```

Dateien der Facharbeit (CD)

/Facharbeit/.. - Tex-Dokumente der Facharbeit und Diagramme
/Stufe I/.. - Erste Implementation (Alpha)
/Stufe II/.. - Erweiterung (Beta)
/Stufe III/.. - Endgültige Version
/Stufe IV/.. - Verkürzte Facharbeitsversion
/Quellen/.. - Quellen als gespeicherte Websites
/Dokumentation (..)/.. - Generierte Dokumentation der Klassen
/Gedanken/.. - Skizzen für die Spiellogik
/Resourcen/.. - Grafiken für das Spiel

Hinweise: In den jeweiligen „Stufe ..“-Verzeichnissen befindet sich in dem Ordner „/bin/Release/“ die ausführbare Datei. Die Projektmappen sind direkt in den Stufen-Verzeichnissen und können mithilfe von Visual Studio geöffnet werden.

Erklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

_____