

Anforderungen

- jedes Level startet mit angefangenem Parcour und Gewinn-Bedingung
- Spieler beendet Parcour, indem er Objekte auf Bildschirm platziert
- Kettenreaktionen erfüllt Gewinn-Bedingung → Level abgeschlossen
- ! Level Editor und Physics Engine

→ Core Features:

- ▶ Switching simulation- and editor view with play/reset button.
- ▶ Feature static, dynamic, and special physics objects¹
- ▶ In editor view can placing/removing certain objects
- ▶ Level loading using human-readable JSON files
- ▶ Puzzle mode (as seen in Crazy Machines) with level progression
- ▶ Simple graphics using basic shapes and colors, real-time display
- ▶ Levels feature win conditions, constraint zones, and inventory

Low-fidelity Clone von Crazy Machines (2005). Funktionalität des original Spiels: siehe Video [\[2\]](#).

Physiksimulation und Editiermodus: "Der zentrale Spielablauf wechselt zwischen dem Platzieren von Objekten und dem Klicken auf 'Play', um die Simulation zu starten."

- Statische und dynamische Physikobjekte und Spezial Objekte (siehe Kick-off Folien)
- Button zum Starten/Abbrechen der Simulation & Zurücksetzen in den Editiermodus
- Editor Modus mit simplen Platzieren/Entfernen von Objekten

Level-system: "Levels sollten per Datei modifizierbar sein."

- Unterstützung für Level-Dateien
- Dateien sollten auch menschenlesbar sein
- Level Progression nach Abschließen des Levels im Puzzle Modus

Einfache Grafik: "Programmierer-Grafik" tut seinen Zweck."

- Objekte als einfache Formen
- Farben hinzufügen
- Anzeige in Echtzeit

Puzzle-Modus: "Mach daraus ein Spiel, indem ihr den Puzzle-Modus einrichtet."

- Siegebedingungen
- Einschränkungszonen
- Inventarsystem

Wahlfeatures sind Vertiefungen. Wählt 3 aus. Sterne markieren Herausforderungen, versucht eine davon selbstständig Umzusetzen.

I. Richtiges Menü: "Das Schlimmste ist ein Game-Prototyp, den man nur mit Alt+F4 beenden kann..."

- Hauptmenü als Zentrale hinzufügen
- Levelauswahlmenü mit Thumbnails und Level Infos

II. Mehr Objekte: "Schwingen, in der Luft hängen, ziehen, befestigen – alles erfordert Seile und Möglichkeiten zur Befestigung."

- Seil, statischer Anker, Hebel hinzufügen (siehe Video)
- Ballon und Eimer um Befestigungspunkte für Seile erweitern

III. Maschinen-Interface: "Zum Trainieren/Testen von KI-Agenten möchten wir das Spiel auch ohne grafische Oberfläche auf Servern betreiben können."

- Zustandprotokoll (Text/Console) hinzufügen
- Headless Mode / Kommandozeilen-Interface hinzufügen

IV. Level-Editor+: "Geh über den einfachen Level-Editor hinaus."

- Drag & Drop
- Bereits platzierte Objekte verschieben/rotieren
- Undo/Redo-Funktion

V. Komplexere Grafiken: "Gib dem Spiel eine visuelle Identität. Und verwirr CNNs."

- Objekte als Texturen/Bilder rendern
- Möglichkeit zum austauschen von Sprite-sheets/texture-packs
- Hintergrund

↳ Zusätzliche Features nach Release 2, eins umsetzen

Welche drei wählen wir? (mind.)

→ Elective Features:

1. **Menus:** Add a main menu and a level selection screen with thumbnails and level info, mid simulation pause screen
2. **More Objects:** Extend the object set with ropes, anchors, levers
3. **Machine Interface:** Support a headless mode (no graphics), command-line interface, and game state logging for AI testing.
4. **Level Editor+:** Enable drag & drop, object rotation/movement, undo/redo
5. **Improved Graphics:** Render objects with textures/images, add background visuals, collision sounds

static, dynamic und special Objekte

→ Puzzle Mode:

- Restriction Zone
- Inventory
- Win-Condition
- Play-Button

Wichtig!

- Code Quality

- Test Coverage

- Documentation

↳ Story Cards

↳ use-case diagrams

↳ UML (Architecture)

↳ javadoc

Anforderungsdokumentation

Die Anforderungen des Produkts sollen übersichtlich in einem **Anwendungsfalldiagramm** dargestellt werden. Die einzelnen Anforderungen sollen detailliert auf **Story-Cards** festgehalten werden, auf die während des Praktikums zurückgegriffen werden soll.

Bedienungsanleitung

In diesem Dokument sollen alle **Funktionen des Systems** kurz vorgestellt werden und erklärt sein, wie diese angesteuert werden können.

Generell gilt: Sind Features nicht korrekt in der Bedienungsanleitung beschrieben und werden Sie bei der Bewertung nicht entdeckt, werden sie als nicht vorhanden bewertet. Analoges gilt, wenn sich die Bedienung der Features nicht unmissverständlich aus der Bedienungsanleitung ergibt.

Architekturendokumentation

In diesem Dokument sollen die grundsätzlichen Ideen hinter der gewählten **Architektur** dargelegt werden. Es sollte erklärt werden, in welche Komponenten das System eingeteilt wurde, welche Klassen zur Umsetzung der einzelnen Komponenten dienen und welche Kommunikation zwischen den einzelnen Klassen stattfindet. Außerdem sollte klar werden, inwiefern die gewählte Architektur eine Erweiterbarkeit und einfache Wartbarkeit des Programms gewährleistet.

Für die Dokumentation kann die Verwendung von **UML-Diagrammen** sinnvoll sein.

2 Javadoc-Dokumentation

Alle öffentlichen Klassen, Methoden (abgesehen von Getters und Setters) und Attribute sollten Javadoc-konform dokumentiert sein.

→ Nicht-Funktionsale Anforderungen:

Codequalität

- Einhaltung der Metriken wie im Abschnitt Codingstyle, Metriken, Testabdeckung beschrieben
- Einhaltung Style-Convention wie im Abschnitt Codingstyle, Metriken, Testabdeckung beschrieben
- Kommentierung des Codes, wo notwendig

Testabdeckung

- Abdeckung von 80 % des Codes (ohne GUI Klassen) durch JUnit-Tests

Dokumentation

- Stets aktualisierte Anforderungsdokumentation via Story-Cards, Anwendungsfalldiagrammen
- Stets aktualisierte Dokumentation der Architektur unter Zuhilfenahme von UML-Diagrammen (Klassen-, Objekt-, Sequenz-, Zustands-), wo sinnvoll.
- Stets aktualisierte Javadoc-Dokumentation des Programms
- Stets aktualisierte Dokumentation der Programmverwendung (Bedienungsanleitung)

Codingstyle, Metriken, Testabdeckung

Das Template verwendet die Plugins PMD [\[2\]](#) und JaCoCo [\[2\]](#) zur Generierung von Reports über Metriken, Codestyle und Testabdeckung. Die Verwendung der Plugins und der resultierenden Reports wird im **Tutorial** demonstriert.

Es wird eine Testabdeckung des gesamten Codes von 80% Instruction Coverage erwartet. (Dabei dürfen GUI-Klassen ausgeschlossen werden. Weitere Details dazu finden sich im **Tutorial**.)

Im Template wird unser PMD-Regelsatz [\[2\]](#) eingebunden. Diese Regeln sind für den Code und die Testfälle einzuhalten.

Bewertung

Produktbezogene Qualitätsaspekte (50%)

- 1. Funktionalität und Zuverlässigkeit (30%)
 - 1.1. Implementierung aller Pflichtfeatures (14%)
 - 1.1.2. Implementierung ausgewählter Wahlfeatures (6%)
 - 1.1.3. Fehlerfreiheit der implementierten Funktionen (10%)
- 1.2. Gebrauchstauglichkeit (15%)
 - 1.2.1. Verständlichkeit, Erlernbarkeit, Bedienbarkeit und Attraktivität des Produkts
 - 1.2.2. Abfangen von Bedienfehlern
- 1.3. Effizienz/Performanz (5%)
 - 1.3.1. Keine unverhältnismäßig lange Rechen-/Antwortzeit
 - 1.3.2. Angemessener Ressourcenverbrauch (Arbeitsspeicher etc.)

Entwicklungsbezogene Qualitätsaspekte (30%)

- 2.1. Stabile und modifizierbare Architektur (10%)
- 2.2. Testing (8%)
 - 2.2.1. Testqualität
 - 2.2.2. Prozentuale Testabdeckung
- 2.3: Einhaltung Metriken (4%)
 - 2.4. Codingstyle (4%)
 - 2.4.1. Einhaltung Codierungskonventionen
 - 2.4.2. Verständlicher Code (sprechende Variablen-/Methodennamen, Kommentare wo nötig)
 - 2.4.3. Einfacher Code (wo möglich)
 - 2.5. Versionsverwaltung (4%)
 - 2.5.1. Sprechende Commit-Nachrichten
 - 2.5.2. Sinnvolle Commitgrößen

Dokumentation (20%)

- 3.1. Lückenlose Dokumentation der Anforderungen (Usecase-Diagramm, Storycards) (3%)
- 3.2. Qualitativ hochwertige Dokumentation der Architektur inkl. Begründung (7%)
 - 3.3. Bedienungsanleitung für das Produkt (2%)
 - 3.4. Javadoc-Dokumentation (4%)
 - 3.5. Abschlusspräsentation (4%)

Story Cards: User stellt Spieler ein