Adventure Game mit der Unity Game Engine

Ein Informatik-Projekt im Rahmen des Moduls BTI7301   
an der Berner Fachhochschule



Dokument eingereicht von:

Martin Käser

Fabian Schwab  
Marcel Tschanz

Betreuung durch:

Jürgen EckerleProjekt 1

Gruppe

Käser Martin

Schwab Fabian

Tschanz Marcel

Abgabe: 14.01.2015

Dokumentenversionen

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Erstelldatum | Autor | Änderungen |
| 1.0 | 17.09.2014 |  |  |
| 1.1 | xx |  |  |
| 1.2 | xx |  |  |
| 1.3 | xx |  |  |
| 1.4 | xx |  |  |
| 1.5 | 04.12.2014 |  |  |

**Inhaltsverzeichnis**

[Adventure Game mit der Unity Game Engine i](#_Toc405557270)

[Ein Informatik-Projekt im Rahmen des Moduls BTI7301 an der Berner Fachhochschule i](#_Toc405557271)

[1 Startanalyse 3](#_Toc405557272)

[1.1 Projektbeschrieb 3](#_Toc405557273)

[1.2 Projektziele 3](#_Toc405557274)

[1.3 Scoping 3](#_Toc405557275)

[1.4 Anforderungen 3](#_Toc405557276)

[2 Requirement Engineering 3](#_Toc405557277)

[2.1 Use Case Diagramm 3](#_Toc405557278)

[2.2 Use Cases 3](#_Toc405557279)

[2.2.1 Gameplay (Fäbu) 3](#_Toc405557280)

[2.2.2 Spielfigur (Marcel) 4](#_Toc405557281)

[2.2.3 Steuerung (Tinu) 4](#_Toc405557282)

[2.2.4 Items (Fäbu) 4](#_Toc405557283)

[2.2.5 Gegner (Tinu) 4](#_Toc405557284)

[2.2.6 Prototyperweiterungen 5](#_Toc405557285)

[2.2.7 Table of Content: Nice to have 5](#_Toc405557286)

[3 Design Phase 6](#_Toc405557287)

[3.1 Spielkonzept 6](#_Toc405557288)

[3.2 Spieleigenschaften 6](#_Toc405557289)

[3.3 Spielstory (bezogen auf Basislevel) 6](#_Toc405557290)

[3.4 Erstes Spielkonzept 8](#_Toc405557291)

[4 Technologien / Pattern 9](#_Toc405557292)

[4.1 Hierarchical State Machine 9](#_Toc405557293)

[4.2 Speichern und Laden in Unity 9](#_Toc405557294)

[Alternative: UnitySerializer 9](#_Toc405557295)

[4.2.1 Speicher- und Ladevorgänge in Rocket 10](#_Toc405557296)

[5 Realisierung 10](#_Toc405557297)

[5.1 Arbeitsweise 10](#_Toc405557298)

[5.1.1 Konfigurationen Windows / Mac 10](#_Toc405557299)

[5.1.2 Mergen 10](#_Toc405557300)

[5.2 Übersicht über Komponenten des Levels 12](#_Toc405557301)

[5.2.1 Levelplan 12](#_Toc405557302)

[5.3 Modellierung der Räume und deren Verhalten 13](#_Toc405557303)

[5.3.1 Situation beim Spielstart 13](#_Toc405557304)

[5.3.2 Status der Räume 13](#_Toc405557305)

[5.4 GameController 14](#_Toc405557306)

[5.5 Spielfigur 14](#_Toc405557307)

[5.5.1 Tastatur-Steuerung 14](#_Toc405557308)

[5.5.2 Maus-Navigation 14](#_Toc405557309)

[5.5.3 Inventar 14](#_Toc405557310)

[5.5.4 Interaktionen mit Objekten 14](#_Toc405557311)

[5.6 Computer 14](#_Toc405557312)

[5.6.1 Computer-Card 15](#_Toc405557313)

[5.7 Roboter 15](#_Toc405557314)

[6 Erfahrungen 15](#_Toc405557315)

[7 Projektabschluss 16](#_Toc405557316)

[7.1 Fazit 16](#_Toc405557317)

[7.2 Zusammenfassung 16](#_Toc405557318)

[7.3 Einfluss des Projekts 16](#_Toc405557319)

[7.4 Weiterführende Arbeiten 16](#_Toc405557320)

[8 Referenzen 16](#_Toc405557321)

[9 Appendix 16](#_Toc405557322)

[9.1 Spielideen 16](#_Toc405557323)

[9.2 Brainstroming - Mögliche Spielumgebungen 16](#_Toc405557324)

[9.3 Brainstroming – Spielideen 17](#_Toc405557325)

[9.4 Mögliche Spielinhalte 21](#_Toc405557326)

# Startanalyse

## Projektbeschrieb

[Anforderungsdokument](Anforderungsdokument.docx)

## Projektziele

[Anforderungsdokument](file:///C:\GitHub\rocket\Dokumente\Dokumentation\Anforderungsdokument.docx)

## Scoping

[Anforderungsdokument](file:///C:\GitHub\rocket\Dokumente\Dokumentation\Anforderungsdokument.docx)

## Anforderungen

[Anforderungsdokument](file:///C:\GitHub\rocket\Dokumente\Dokumentation\Anforderungsdokument.docx)

# Requirement Engineering

## Use Case Diagramm

[Diagramm](../Use%20Cases/UseCaseDiagramm.pdf)

## Use Cases

[Use Cases](../Use%20Cases)

### Gameplay (Fäbu)

1. Start des Spiels: Wie gelangt man ins Spiel, Informationsfenster, Start des Timers, Spieler ist bereit loszulaufen
   1. Extended Use Case: Anzeige der Uhr als Countdownvisualisierung
2. Beenden des Levels: Kriterien dazu, nach Abschluss des Levels erscheint eine Statistik, Items auswerten – Belohnung?; Game stoppt
3. Umgebungsgeräusche: Während dem Spiel werden Umgebungsgeräusche wahrgenommen. Diese stammen von:  
   - Gegnern  
   - Apparaturen

VORSCHLAG: Gegner verursachen auch ein leichtes leuchten, falls Audio überhört wird

### Spielfigur (Marcel)

+ Suche Assets für Level, wie erzeugen die anderen solche Levels?

1. Beim Gehen werden Schrittgeräusche erzeugt | Use Case hier: alles im Zusammenhang mit herumlaufen (Kollisionen)
2. Die Figur ändert die Gangart von gehen zu laufen inkl. allen Auswirkungen auf Spieler und ev. Gegner

### Steuerung (Tinu)

1. Gehen, rennen, ducken: Mit entsprechenden Tasten und Animation
2. Rundumsicht: In welcher Situation, wie: Tasten / Maus?  
   Sichtradius nach oben und unten?
3. Blick um Gegenstände auf Tastendruck um Gegner zu beobachten und / oder...? Das Sichtfeld entspricht kurz einer veränderten Position.
4. Möglichkeit zum „Benutzen“ von Objekten  
   In bestimmten Räumen sind Objekte vorhanden, die „benutzt“ werden können.  
   Aktionstaste, wie sind Objekte markiert...etc.  
   Was für Objekte könnten das sein?

### Items (Fäbu)

1. Aufnahme der Items durch die Spielfigur: Per Tastendruck der Aktionstaste, Item kommt in Inventar,   
   Verschiedene Items: Prio1-Item für den Abschluss --> Ideen dazu?  
   Unterschiedlich Wertvolle Items für Belohnung, besser bewacht?
2. Zentrales Artefakt

### Gegner (Tinu)

1. Sichtradien und Hörradien:  
   Die Gegner hören Schritte des Spielers und kommen in die Richtung aus der die Schritte hallten. Erst bei Sichtkontakt kommt es zur Verfolgung.  
   Verfolgungsmodus oder Patroullienmodus
2. Patroullie / Waypoints  
   Vordefinierte Spawnpunkte, Patroullienwege vordefiniert?  
   Die Wachen resp. deren Waypoints sind so intelligent, dass sie die Räume bestmöglich abdecken.
3. Unterschiedliches Verhalten der beiden Gegnertypen

-> Ausarbeitung der Typen mit Fähigkeiten / Geschwindigkeit / Seh- und Hörstärke etc.

### Prototyperweiterungen

Vorschläge, die direkt den Prototypen erweitern würden und somit vor den “Nice to have” – Feature implementiert weden sollen.

Im Gespräch mit Herr Eckerle wurde festgelegt, dass wir beim Weiterentwickeln und Priorisieren von neuen Implementationen den Fokus auf Spielelemente und neuen Interaktionen legen.

Somit wird unser Prototyp nur mit groben Strukturen und Grafiken auskommen müssen, dafür sollten die Möglichkeiten des Spielers und die Interaktionen des Roboters ausgefallener sein.

Featureliste:

* Der Spieler kann die feindlichen, aktiven Roboter abschalten.
* Wenn ein aktiver Roboter einen deaktiviereten Roboter sieht, kann er zu seiner Position hinfahren und dann den Roboter wieder aktivieren.
* Dabei läd der Roboter auch die Batterie um 10% auf.

### Table of Content: Nice to have

Das die Liste mit „Nice-to-have“-Features recht lange ist, werden dazu im Moment noch keine Use Cases erstellt. Sobald dann die Ziele der ersten Priorität abgearbeitet sind, kann diese Liste nochmals überdacht werden und eine paar Ideen besser ausgearbeitet werden. Im Appendix finden sich weitere

* Speicher- und Ladevorgänge
* Waffen für Gegner und Spieler
* Mit gefundenem Geld können nützliche Gegenstände gekauft werden
* Rangliste
* Minimap, welches nahe Gegner als Punkte darstellt
* Mehr Gegnercharaktere
* randomizing in verschiedene Strukturen bringen
* Leveldesign verschönern (Beleuchtung, Geräusche, Objekte, Konturen, Gegner, einfach den "Look" verbessern)
* Weitere neue Gegnernertypen designen (Verhalten, andere Routinen etc.)
* Munition generell oder verschiedene Munitionstypen zu Waffen einbauen
* Unterschiedliche Hintergrundmusik, der Szene angepasst, Siegessoundtrack
* Türen einbauen, Schiebetüre, Drehtüre etc.
* Fenster ( Sichtkegel erkennen den Spieler durch die Wand, ist aber nicht "begehbar")
* Healthsystem einbauen
* GUI für Spieler mit Leben / Rüstung und/oder aktueller Status und Bewaffnung
* Schadenkonzept für Gegner, Fallen und Spieler
* Gegener haben ebenfalls ein Healthbar
* Gegner betäuben oder töten können
* Verschiebbare Hindernisse, evtl. random platziert / vor dem Exit
* Fallen, siehe Spielideen
* Rangliste (für: Spielzeit, Anzahl Tode, gesammelte Punkte, Alarme ausgelöst)
* Explosive Fässer
* Pfeifen-Aktion für Spieler (Geräuschablenkung vom Gegner auf distanz zum Spieler hin)
* Gegenstand werfen (Geräuschablenkung vom Gegner auf distanz)
* "Aufmerksamkeits"-Level (Eine gewisse Zeit nachdem der Spieler erkannt wurde, sind die Gegner Aufmerksamer/ändern das verhalten )
* Inventar + Gegenstände
* Charaktereigenschaften verbessern (Leben, Geschwindigkeit)

# Design Phase

## Spielkonzept

**Leveldesign**

* Design des Levels – Eventuell vordefinierte Assets verwenden
  + Eine Ebene
  + Eintritt ins Level, Austritt / Ausgang (=Beenden des Levels)
  + Diverse Hindernisse, Räume, Dekoration
* Hintergrundmusik
* Passende Beleuchtung

## Spieleigenschaften

- Single Player

- First-Person

## Spielstory (bezogen auf Basislevel)

Nach einem kurzen Briefing durch ein Informationsfenster, welches über den groben Spielplan und die Regeln informiert, beginnt das Spiel mit dem Eintreten der Figur in das Level. Der Spieler steuert aus der First-Person-Perspektive eine menschenähnliche Figur durch ein abgeschlossenes Level, eines Bürokomplexes. Die Umgebung des Basislevels wird auf eine Ebene beschränkt bleiben.

Um den ersten Raum zu verlassen, muss ein passender Gegenstand gefunden werden, um die Türe für den nächsten Raum öffnen zu können. Der Spieler stellt fest, dass es im nächsten Raum kein Licht brennt. Nur die Notbeleuchtung ist aktiv. In der Nähe befindet sich ein Sicherungskasten mit Stromsymbol. Bei Interaktion mit dem Sicherungskasten wird der Spieler darauf hingewiesen, dass etwas kaputt ist. Bei einer Erneuten Interaktion findet der Spieler heraus, dass die Sicherung kaputt ist. Die Ersatzsicherung befindet sich immer noch im ersten Raum. Sobald der Spieler die Ersatzsicherung aufgenommen hat, ist diese im Inventar sichtbar. Um den Strom einzuschalten, muss der Spieler die Ersatzsicherung aufnehmen und erneut mit dem Sicherungskasten (-> sorgt für Levelbeleuchtung) interagieren. Das Licht im Level geht an, der Stromfluss ist wiederhergestellt. Ein Lift in der Nähe ist blockiert durch ein zerstörbares Hindernis. Beim Untersuchen des Hindernisses, bemerkt der Spieler, dass das Objekt durch eine Explosion zerstörbar sein könnte.

Neben dem Lift befindet sich ein Kontrollraum, der die Stromversorgung für die Räume einschaltet. Zudem befindet sich im Kontrollraum ein Button, um den über ein Fenster einzusehenden, anliegenden Raum zu öffnen. Anreiz dazu ist eine Leiche, die einen Batch bei sich trägt. Mit dem Batch ist ein Computer bedienbar, der u.A. den Lift aktivieren kann, den Schatzraum öffnen kann und den Code für die Bombe bereithält. Zudem ist im angrenzenden Raum ein Roboter sichtbar(oder nicht sichtbar). Dieser wurde mit dem Betätigen des Schalters ebenfalls aktiv. Der Roboter bemerkt den Spieler noch nicht, aber verlässt den Raum und gibt dabei ein Tonsignal ab.

In einem anderen Teil des Spiels befindet sich versteckt eine Bombe.

Um den Sprengsatz zu benutzen ist eine Kombination zur Aktivierung nötig.

(-> Computer)

Sobald das Hindernis weg ist, stellt der Spieler fest, dass er für den Lift einen Schlüssel benötigt, um von dieser Ebene wegzukommen. Das wird dem Spieler auch gesagt beim Untersuchen vom Lift. Der Schlüssel befindet sich in einem Raum, welcher noch nicht zwingend vom Spieler besucht wurde. Hat der Spieler den Schlüssel aufgenommen, kann er die Aufzugstüre öffnen und durch das Aktivieren des Lifts das Spiel gewinne.

## Erstes Spielkonzept

Nach einem kurzen Briefing durch ein Informationsfenster, welches über den groben Spielplan und die Regeln informiert beginnt das Spiel mit dem Eintreten der Figur in das Level. Der Spieler steuert aus der First-Person-Perspektive eine menschenähnliche Figur durch ein abgeschlossenes Level, ähnlich einer Halle, eines Museums oder eines Bürokomplex. Die Umgebung des Basislevels wird auf eine Ebene beschränkt bleiben.

Hindernisse und verwinkelte Gänge bieten innerhalb des Levels Sichtschutz vor den Gegnern. Um Hindernisse kann gespäht werden – entweder mit einer Bewegung oder einem Hilfmittel.

Die Fortbewegungsmöglichkeiten sind entsprechend der essentiellsten Fähigkeiten (gehen, laufen, ducken) eines Menschen nach zu bauen. Auch Geräusche werden wahrgenommen und helfen dem Spieler beim erfolgreichen Absolvieren des Levels.

Hauptziel für den Spieler ist das Suchen eines zentralen Items, welches am zu erreichenden Ausgangspunkt einzusetzen ist und womit das Level abgeschlossen und verlassen werden kann. Ein Nebenziel ist das Einsammeln von wertvollen Gegenständen, die beim Beenden des Levels zu Belohnungen in Form weiterer Fähigkeiten oder nützlichen Gegenständen führen. Beim Abschluss erscheint ein weiteres GUI, welches über die erreichte Punktzahl informiert.

Während dem durchqueren des Levels, wird man mit Gegnern und der Umgebung interagieren. Zu Beginn werden zwei Gegnertypen unterschieden. Diese Gegenspieler – modelliert als Automaten - reagieren automatisch auf das Verhalten des Spielers. Dabei werden ihre Zustände von Geräuschen und visuellen Kontakten beeinflusst. Auch die Gegner selbst verursachen Geräusche und ihre Nähe sind so für den einfacher Spieler einzuschätzen. Im Basislevel hat der Spieler keine Möglichkeit, sich gegen die Gegner aktiv zu verteidigen. Er versucht sich primär der Verfolgung durch die Gegner zu entziehen. Bei Feindberührung (Waffengewalt oder gegnerische Figur), oder beim Ablauf eines Countdowns (auf dem Bildschirm sichtbar als Uhr oder Ähnliches) gilt das Level als nicht bestanden.

# Technologien / Pattern

## Hierarchical State Machine

Das Verhalten der Roboter in Rocket wurde mit Zustandsmaschinen („State-Machines“) modelliert.

->Diesen Abschnitt komplettieren, wenn Märcu meine HSFM implementiert hat.

## Speichern und Laden in Unity

Das Speichern von Spielständen wird im Allgemeinen über den Ansatz der (De)-Serialisierung realisiert. Sich verändernde GameObjekte werden also exportiert und beim laden wieder eingelesen.

Bei der Serialisierung nach XML, JSON oder als Binary stehen einem aber die Unity-Komponenten im Weg, welche von MonoBehaviour erben -> GameObjects.   
Diese Unityklassen implementieren die notwendigen Interfaces nicht. Typische Fehlermeldungen, bei Exporten in XML und der Binary Serialisierung:

**XML:** *InvalidOperationException: To be XML serializable, types which inherit from IEnumerable must have an implementation of Add(System.Object) at all levels of their inheritance hierarchy. UnityEngine.Transform does not implement Add(System.Object)*

**C# Serialiazation (binary):**  
*SerializationException: Type UnityEngine.GameObject is not marked as Serializable*

### Alternative: UnitySerializer

Quelle: <http://whydoidoit.com/unityserializer/>

Seit Mitte 2013 nicht mehr Weiterentwickelt und weist Kompatibilitätsprobleme mit Unity 4.5 auf. Der Unityserializer basiert auf dem geschriebenen SilverlightSerializer, mit dem er Unitykomponenten serialisieren kann.

Die Daten werden in PlayerPrefs als String Key in die Registry geschrieben:  
(HKEY\_CURRENT\_USER\Software\DefaultCompany\SaveLoad)

Alle zu serialisierenden Gameobjekte müssen das Script *StoreInformation.cs* angehängt haben. Das Gameobjekt erhält so zum einen eine Unique ID und zum anderen werden damit die notwendigen Interfaces implementiert. Auch alle Kinder eines Objektes müssen dieses Script angehängt bekommen.

### Speicher- und Ladevorgänge in Rocket

Die Alpha-Version des Spiels Rocket wird voraussichtlich keine Speicher-und Ladefunktionen bieten. Falls vor Abgabetermin noch freie Zeit bleibt, könnte dies in einer nächsthöheren Releaseversion integriert werden.   
Erste Tests haben aber unbefriedigende Resultate beim Wiederherstellen von Objekten zur Laufzeit geliefert. Besonders die Re-Positionierung der Komponenten im Level scheiterte regelmässig.

# Realisierung

## Arbeitsweise

### Konfigurationen Windows / Mac

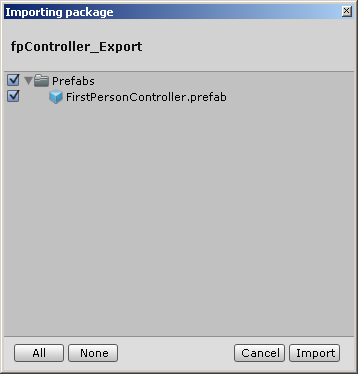
### Mergen

Anfangs Dezember begannen die Arbeitsgruppe damit, die einzeln entwickelten Komponenten zusammenzuführen und ein erstes Level zu gestalten, welches alle diese Einzelteile in sich vereinigt.

Diese Vorabversion wurde an alle Gruppenmitglieder verteilt und Änderungen wurden auf folgende Weise im Nachhinein implementiert:  
  
Bsp. **FirstPersonController Prefab verändert (exakt: Child Main Camera**)

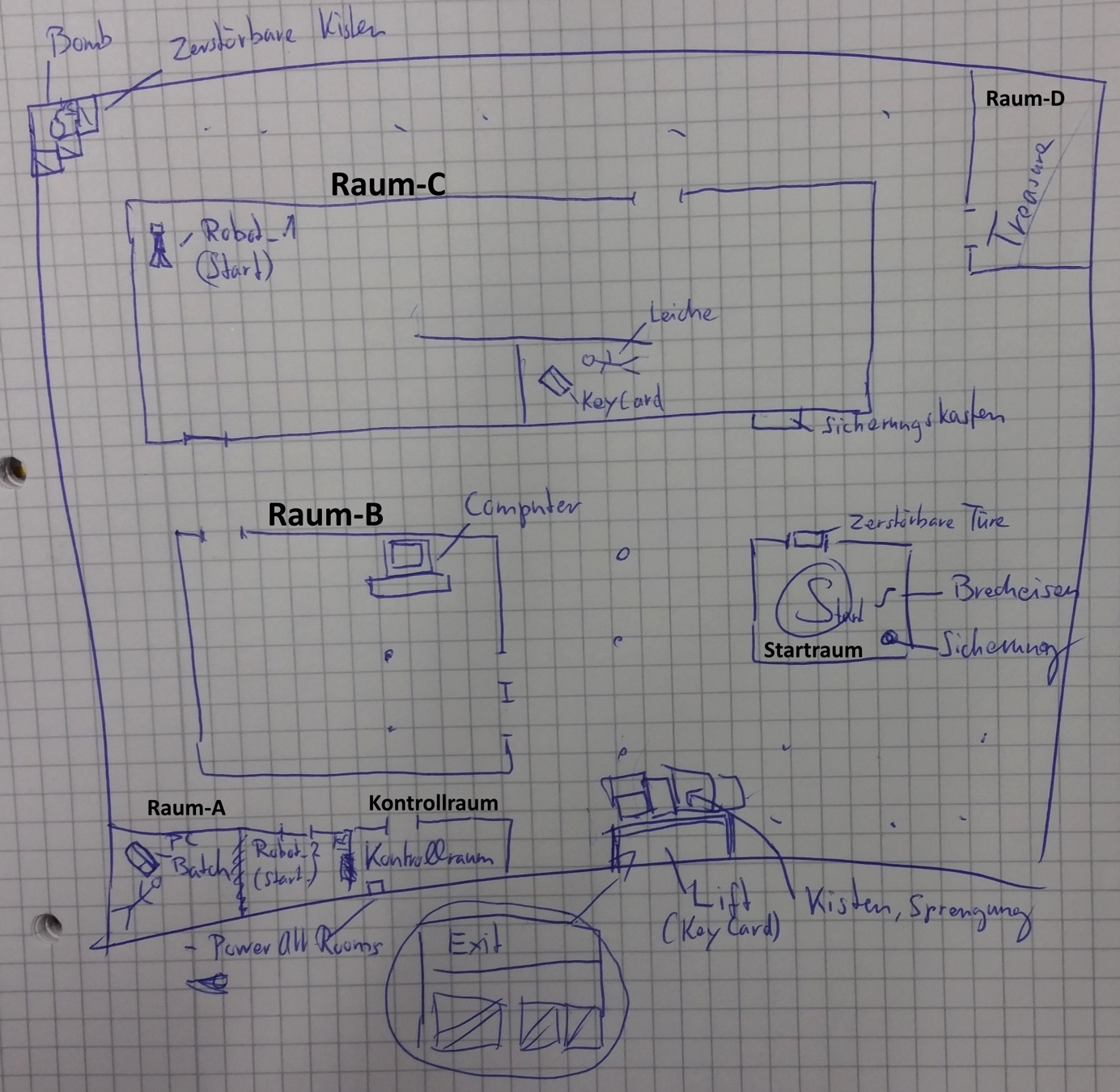
1. Erstellen der benötigten Scripts oder Materialien in den dafür vorgesehenen Ordnern.
2. Änderungen am Objekt in der Hierarchie vornehmen
3. Auf Parent des veränderten Objekts (wenn nicht selbst Parent) klicken und  
   im Inspector „apply“ wählen.
4. In der Project-View Rechtsklick auf das Prefab und „Export Package“ wählen
5. Wenn möglich alles abwählen, was sicherlich in keinem Zusammenhang mit den Änderungen steht. Auf jeden Fall das Prefab selbst und dann noch unbedingt benötigte Komponenten.
6. Änderungen kommunizieren, dass von Prefab X eine neue Version bereit steht

**Auf den anderen Clients**:

1. Rechtsklick auf Prefabs Folder und „Import Package“ wählen
2. Package auswählen und Frage nach Import bestätigen:  
     
    -> Import
3. Nun sollte auch die dazgehörige Instanz in der Hierarchie bereits aktualisiert worden sein. Falls dies nicht der Fall sein sollte, eventuell die „Rever“ Option auf dem fraglichen Prefab auswählen?

## Übersicht über Komponenten des Levels

### Levelplan



## Modellierung der Räume und deren Verhalten

Da es sich bei den Räumen um relativ statische, passive Elemente im Spiel handelt wurden die Aktionen und Zustände der Räume nicht durch eine State Machine modelliert sondern lassen sich mit einem Guard Diagram abbilden.

### Situation beim Spielstart

Nachdem der Spieler aus dem Startraum entkommen ist, findet er das Level in Dunkelheit vor. In unmittelbarer Nähe befindet sich der Sicherungskasten, in dem eine Sicherung eingelegt werden muss. Dies lässt die Notstromversorgung aktiv werden.

Nur eine durch eine schwache Beleuchtung gekennzeichnete „Strasse“ führt vom Startraum / Sicherungskasten hinüber zum Kontrollraum. Alle Räume sind zu dem Zeitpunkt dunkel und noch geschlossen.

Im Kontrollraum sind dann folgende Schalter zu finden:

* Stromversorgung der Gänge -> OFF, wenn eingeschaltet dann wird das Level hell
* Stromversorgung der Räume -> OFF, der Kontrollraum besitzt im Moment nur Notstrom. Wenn eingeschaltet, dann öffnen sich bis auf den Raum D (TreasureRaum) und den an den Kontrollraum angrenzenden Raum A die restlichen Räume und versorgen sie mit Strom. Licht brennt aber nach wie vor nicht in allen Räumen, sondern muss teilweise über die Lichtschalter manuell eingeschaltet werden.
* Raum A kann nur mit dem Don’t push me – Button geöffnet werden.

### Status der Räume

Die Hauptstromversorgung der Räume ist unterbrochen, da der Hauptschalter im Kontrollraum nicht auf ON ist. Dies bedeutet, dass zu Beginn des Spiels das Attribut powerState der einzelnen Environment Objekte, mit Ausnahme jenes im Kontrollraum auf false geschaltet sind. Siehe den Beschrieb der Initial Situation. Wird der Hautpschalter für Räume aktiviert, wird eine Message an die Gruppe Rooms geschickt, und die Räume reagieren auf den Empfang, in dem sie den powerState auf true schalten.

Es funktionieren somit nur jene elektronische Anlagen, die an der Notstromversorgung angeschlossen sind, was im Moment nur für den Kontrollraum zutrifft. Weitere Räume, die an der Notversorgung angeschlossen sind, sind nicht geplant. Falls diese doch realisiert werden sollten, käme am einfachsten ein weiteres Attribut: emergencyPower zur Environment Klasse hinzu.

## GameController

## Spielfigur

### Tastatur-Steuerung

Klassisch wird mit der WASD-Tastenbelegung gespielt. Daneben stehen dem Spieler folgende Steuermöglichkeiten zur Verfügung:

|  |  |  |
| --- | --- | --- |
| **Funktion** | **Tastenbelegung** | **Wirkung** |
| Laufen | Shit+W | Die Spielfigur bewegt sich schneller vorwärts |
| Lean-Look Links | Q | Der Sichtwinkel verändert sich, damit auf die linke Seite hin „um die Ecke“ geschaut werden kann. |
| Lean-Look Rechts | W | Der Sichtwinkel verändert sich, damit auf die rechte Seite hin „um die Ecke“ geschaut werden kann. |
| Pause |  |  |
| Inventar aufrufen |  |  |
| ?? |  |  |
| ?? |  |  |

### Maus-Navigation

### Inventar

### Interaktionen mit Objekten

## Computer

In Raum B befindet sich ein Computer, der vom Spieler unter der Bedingung des Besitzes der [Computercard](#_Computer-Card) benutzt werden kann.

Will man den Rechner verwenden, fordert er einem auf, die PC Batch Karte vorzulegen. Wenn man dies tut erscheint eine Menuliste, aus der man 2 Optionen wählen kann.

Optionsliste:

a) Enable Elevator  
Aktiviert den Lift. Nebst dem Beseitigen des Hindernisses und dem Auffinden der Lift-Karte ist dies die dritte Bedingung um ein Verlassen des Levels zu ermöglichen.

b) Code for arming of the bomb  
Um die Bombe zu verwenden und mit ihr das Hindernis vor dem Lift wegsprengen zu können, bedarf es eines Codes.

c) Unlock treasureroom

Öffnet die Türe zum Schatzraum und gibt die wertvollen Item frei.

d) Pause Robots for X Seconds   
Weil die Wahrscheinlichkeit, die Bombe zu platzieren und dabei vom Roboter nicht erwischt zu werden sehr klein ist, kann mit dieser Option der/die Roboter paralysiert werden.

+) EXIT

+) reset (schliesst den Schatzraum wieder, reaktiviert die Roboter, ändert Bomben-Code, sperrt den Lift)

Währenddem das Menu geöffnet ist, pausiert das Spiel und auch die Navigation mit Maus und Tastatur wird blockiert.

Computer fährt herunter (rsp. das Menu wird ausgegraut), nachdem man 2 Optionen ausgewählt hat / meldet, dass 2 Eingaben getätigt wurden. Frage, ob man

Durch dieses Schema ist es fast zwingend, folgendermassen vorzugehen (wenn man die Roboterkonfrontation vermeiden will):

1. Auswahl Treasure Room + disable Robots

-> Roboter danach manuell ausschalten

2. Reset

3. Lift und Bombencode

4. Level verlassen

### Computer-Card

In Raum C befindet sich neben dem Soldaten die Computerkarte, mit der die Verwendung des Computers möglich wird.

## Roboter

## Inventar und Items

Die Interaktion mit manchen Spiel-Objekten verlangt vom Spieler dass er ein bestimmtes Item in der Hand hält. Damit dies auch mit unterschiedlichen Items möglich und leicht erweiterbar bleibt, wurde ein Inventar implementiert.

### Generische Itemklasse

Die Itemklasse ermöglicht es schnell und unkompliziert ein neues Item der Spielewelt hinzuzufügen. Man fügt dabei einfach das Items.cs Script auf das gewünschte Game-Objekt und gibt Label so wie das Icon für die Darstellung im Inventar an. Beim interagieren wird dann automatisch erkannt dass es sich um ein auflesbares Game-Objekt handelt.

Abbildung 5‑1: Inventar

### Info-Boxen

Damit der Spieler genauer erkennen kann was er betrachtet, wird Ihm eine kleine Info-Box dargestellt mit dem Namen des Items. Diese Info-Boxen sind vom Typ “ProximityMessage” und werden von den Items bei deren Instanzierung automatisch generiert. Näheres dazu unter dem Abschnitt „Darstellung von Text-Meldungen“

### Auswählen des Items

Der Spieler hat die Möglichkeit das gewünschte Item durch drehen des Mausrades auszuwählen. Das aktive Item wird rot dargestellt und wird bei einer Interaktion mit einem anderen Game-Objekt mitgeschickt. Näheres dazu unter dem Abschnitt MessageDispatcher.

## Darstellung von Text-Meldungen

Für die Darstellung von Text-Meldungen wird die Klasse „TextDisplayer“ verwendet. Sie wählt unter berücksichtigung von Priorität und Anzeigedauer die aktuell passendste Nachricht aus und stellt sie dar. So hat zum Beispiel die „InteractionMessage“ eine höhere Priorität als die „ProximityMessage“ und wird daher bevorzugt bei der Darstellung.

In diesem Kapitel sind die Klassen vorgestellt welche den TextDisplayer verwenden.

### ProximityMessage

Damit der Spieler erkennen kann, was er im Moment betrachtet wurde die Klasse „ProximityMessage“ erstellt. Sie kann zusammen mit einer gewünschten Meldung einem Game-Objekt zugewiesen werden. Die ProximityMessage verwendet dann bei der Betrachtung den TextDisplayer für das darstellen der Meldung.

### Macintosh HD:Users:mkaeser:Desktop:asdf.pngInteractionMessage

Bei der Interaktion mit einem Game-Objekt kann eine InteractionMessage verwendet werden. Dies soll dem Spieler Hinweise zum verwenden des Game-Objekts geben. Zum Beispiel kann man über den TextDisplayer die Meldung ausgeben “Diese Türe versperrt mir den Weg!”, sobald der Spieler mit einer geschlossenen Türe interagieren will. Dabei kann je nach Item, welches der Spieler in der Hand hält, eine andere Meldung dargestellt werden.

Die „InteractionMessage“ ist generisch aufgebaut, somit kann man über das GUI die unterschiedlichen Rückmeldungen pro Item definieren.

Abbildung 5‑2: Konfiguration InteractionMessage

## Macintosh HD:Users:mkaeser:Desktop:Screen Shot 2014-12-17 at 21.06.49.pngMessageDispatcher & Telegramm

Die Klassen „MessageDispatcher“ und „Telegramm“ werden verwendet um ein generisches Interface für die Kommunikation zwischen den einzelnen Game-Objekten zu bieten.

Möchte man eine Nachricht von einem Game-Objekt ans nächste schicken so benutzt man den „MessageDispatcher“. Dieser nimmt die Nachricht entgegen und verpackt sie in ein Telegramm. Das Telegramm beinhält nebst der Nachricht eine Empfanger- und Sendeadresse so wie eine Liste mit zusätzlichen Daten.

Somit kann dann der Empfänger selber entscheiden wie er z.B. auf eine Interaktion mit dem Spieler reagieren soll.

Abbildung 5‑3: Aufbau Telegramm

Weiter kann man dem „MessageDispatcher“ mitteilen er soll die Nachricht mit einer Verzögerung verschicken. Dies wird zum Beispiel bei der Bombenplatzierung verwendet für den Countdown der Bombe.

### Spielerinteraktion mit Game-Objekten

Klickt der Spieler während des Spieles, so wird entschieden ob man mit dem Game-Objekt interagieren kann welches man gerad betrachtet.

Falls ja, wird diesem Game-Objekt ein Interaktions-Telegram verschickt. Dieses Interaktions-Telegram wird über den MessageDispatcher verschickt.

## EntityManager

Der Entitymanager bietet ein globales Interface für die adressierung von Game-Objekten.

Ein Game-Objekt kann sich bei dieser Klasse mit einer ID anmelden um direkt angesprochen zu werden. Ausserdem bietet der EntitiyManager die Möglichkeit, dass ein Game-Objekt sich auch mehreren Gruppen registrieren kann.

### Einsatz bei Telegramme

Wie das Kapitel „MessageDispatcher und Telegramm“ beschreibt, können Nachrichten zwischen Game-Objekten ausgetauscht werden indem man ein „Telegram“ verschickt.

In diesem Telegram werden Sender und Empfäger über die im EntityManager definierten IDs definiert. Es ist also zwingend notwendig, dass der Empfänger sich als Objekt im EntityManager registriert hat.

# Erfahrungen

# Projektabschluss

## Fazit

## Zusammenfassung

## Einfluss des Projekts

## Weiterführende Arbeiten

# Referenzen

[1] W. Strunk, E. B. White, R. Angell, *Elements of Style*, Fourth Edition, Longman Publishing, Ithaca, N.Y., 1999.

# Appendix

Appendices are for tables, listings, or code fragments that give much more detail than you want to include in the thesis itself but that are important to your thesis.

## Spielideen

## Brainstroming - Mögliche Spielumgebungen

aus dem Brainstroming hervorgegangen sind:

* Pyramide
* Höhle
* Vorgefertigte Models verwenden
* Altes Haus
* Keller
* Bunkeranlage
* Maschinenhalle Fabrik, alte oder neue mit Roboterfertigungsanlagen, die dem Spieler gefährlich sein können.
* Raumstation (Mögliches Aussenlevel mit Mondkrater, Schwerelosigkeit)
* Schule
* Bürogebäude
* Gefängnis
* Kloster
* Burg oder Schloss

## Brainstroming – Spielideen

Räuber

In einem Level stehen eine Anzahl Kisten – teils etwas versteckt.

Kisten sind umso schwerer zu öffnen, je wertvoller der Inhalt ist.  
Gegner bewachen Kisten und werden nach einer bestimmten Zeit aufmerksam – respektive haben einen bestimmten Sichtradius. Nähert sich ein Gegner, sollte man sich hinter der Kiste ducken.

Mit verschiedenen Werkzeugen hat man unterschiedlich lange, die Kiste zu öffnen.  
Langsame Werkzeuge machen zwar keinen Lärm, man riskiert aber von „patrouillierenden“ Gegnern entdeckt zu werden (ausser man versteckt sich). Schnelle Werkzeuge verursachen ihrerseits viel Lärm, was die Gegner auch aus Winkeln anlockt, die nicht im Sichtradius waren. Dafür hat der Spieler die Möglichkeit, schnell an den Schatz zu kommen.

Eine Progress-Bar zeigt den Fortschritt während des Öffnungsvorgangs auf. Kommt man an den Inhalt der Kiste und hat man eine bestimmte Anzahl (Gold?) oder was auch immer gesammelt, wechselt das Level.

Wenn man von den Gegnern attackiert wird, erscheint ein Optionenmenu und fragt nach, was man tun möchte:

- Gegner mit gesammeltem Gold bestechen

ODER

- Kämpfen (-> Wie?)

Anreiz um die schwierigen Kisten zu öffnen:  
- In schwierigen Kisten können auch Waffen / andere Werkzeuge sein

Portallevel mit Rätsel- und Knobelaufgaben

Ein Portal strategisch in der Halle platziert, ist aber durch einen Code gesichert

Um die Kombination zu kennen, müssen 1-2 Dinge erledigt werden, wie bsp. An einer Wand ein Rätsel lösen, irgendwelche Hinweise richtig deuten etc.

Kennt man die Kombination, schleicht man zur Kiste – ohne gesehen zu werden und kann so den Level verlassen.

Man kann auch auf „gut Glück“ versuchen, die Kombination einzugeben.  
Gibt man sie falsch ein, wird Alarm ausgelöst und die Gegner kommen auf einem zu.

Dann hat man noch 2, 3 Versuche, bis die Gegner einen erreichen.

-> Kampf oder sofortiger Tod.

Retten einer Geisel/Zivilperson

Der Spieler muss unentdeckt bis zur Geisel gelangen, diese befreien und dann zum Extraktionspunkt fliehen. Wird die Geisel oder der Spieler getötet, ist das spiel vorbei.

Der Spieler hat die Möglichkeit die Wachen abzulenken mit Geräuschen oder Gegenständen. Alternativ könnte der Spieler die Wachen betäuben oder sogar töten/zerstören. Das Töten sollte beschränkt werden, damit nicht alle Wachen im Level so neutralisiert werden können.

Sammeln von Wertgegenständen

Im Level sind direkt Gold oder Wertgegenstände von unterschiedlichem Goldwert verstreut. Der Spieler muss pro Level einen Mindestbetrag an Gold stehlen um damit den Raum wieder verlassen zu können. Fallen und Patrouillen versuchen dies zu verhindern. Das Entdecktwerden könnte tödlich sein, oder aber man verliert einen gewissen Goldbetrag und startet vom Start oder vom „Gefängins“ aus neu.

Alternativ könnten auch (zusätzlich) Zivilpersonen „bestohlen“ werden, was dynamischer wäre.

Arena

Man startet in einer Zufällig generierten Karte, nach dem man 1-n einfache Gegner getötet hat,, gewinnt man das Level. Jedes X-te (z.B. 3te) Level wird gegen den gleichen Bossgegner gekämpft. Jeder Bossgegner hat eigene Attacken, Buffs, Stärken und Schwächen. etc.

Der Spieler hat 3 Angriffsarten (Magie, Nahkampf, Fernkampf (Bogen), und unterschiedliche Gegner sind unterschiedlich anfällig dagegen (z.B. Nahkampf ist gut gegen Fernkampf, aber Magie ist gut gegen Nahkampf etc.)

Vor Angriffen der Boss sollte der Spieler visuell oder akustisch gewarnt werden und ausweichen können.

Jump & Run mit Hinternissen und Fallen

Mit Verschieben von Blöcken müssen Wege freigeräumt werden, Gegenstände wie Dynamit oder Seile lassen unmöglich erscheinende Hindernisse überwinden. Diese Gegenstände findet man im bereits zugänglichen Level oder hat diese zum Start im Inventar.

Tiere und Fallen versuchen, den Spieler daran zu hindern. Direkt mit Schaden auf den Spieler oder indirekt mit Verlangsamen der Spielfigur oder Blockieren eines Weges.

Stealth

Der Spieler muss unentdeckt verschiedene Gegenstände sammeln um das Level verlassen zu können.

Gegner versuchen mit Sehen und Hören den Spieler zu ertappen. Könnte Sofort tödlich sein, sonst müsste der Spieler den Angreifern ausweichen oder betäuben können.

Lauf Forest Lauf-Prinzip

Der Spieler kann sich ein Level im „Spectator“ Modus anschauen. Er soll sich Startpunkt, Ziel punkt und hindernisse dabei merken. Nach dem er bereit ist und/oder nach Ablauf eines Timers startet hinter dem Startpunkt ein tödliches Event/Gegner (Lava, Wasser, Killerroboter) und der Spieler muss es lebend bis zum Ausgang schaffen.

Doppelagent -Assassin

Einer der Wachen im Level ist ein Doppelagent. Der Spieler sollte mit „observieren“ der Wachen herausfinden, welche der Wachen einzigartig ist. Der Spieler versucht anschliessend, diese Wache unbemerkt zu beseitigen.

Kann „erschwert“ werden durch:

mit einem Timer, der abläuft und das Spiel beendet beim Ablauf.

Die Wachen sehen immer unterschiedlich aus, der Doppelagent ist immer einzigartig. Nicht immer gleiches Merkmal sollte den Doppelagenten verraten.

Nach einem Alarm, Sichtung des Spielers sind alle Wachen wachsamer und/oder die Patrouille erweitern sich.

## Mögliche Spielinhalte

Hier werden nur einige Ideen-„Schnippsel“ gegeben, die man in eine Vielzahl von Spiele oder Levels einbauen könnte.

Neutrale Fallen

KI-Lose Objekte, die mit einem Mechanismus ausgestattet sind, um allen Personen zu schaden. Das könnten Laserstrahlen sein, die mit Springen, Kriechen oder mit Platzieren von Objekten umgangen werden können.

Statt Laser können es auch Ketten, Sägeblätter, Pfeile oder giftige Tiere oder Stacheln sein, je nach Spielumgebung.

Unterscheidet nicht von Freund und Feind

**Beispiele**: Feuerspucker, Bodenplatten Laser, Rollender Felsbrocken, Stacheln, Lose Hochspannungsleitung.

Anti-Player Fallen

Diese Fallen sind wie in „Neutrale Fallen“ anzusehen, reagieren aber in der Regel nur auf den Spieler. Mit Hacken könnte man diese zum Beispiel zu Neutralen Fallen machen.

**Beispiele**: Autoschuss Kamera, Roboter mit Waffe

Gravitation

Puzzle oder Situationen schaffen, die nur mit einer bestimmten Gravitation gelöst werden können.

Dabei wird dem Spieler selber eine Art „Energie“ gegeben, und er kann die Gravitation selber beeinflussen oder mit hilfe von Game-Objekte kann er zwischen 2-3 Zuständen hin und her schalten. (anderer Bewegungsgeschwindigkeit, maximale Sprunghöhe, Gewichte heben)

Lichtlevel

Die Umgebung wird unterschiedlich beleuchtet. Aufgrund vom Lichtlevel des Spielers (Mit einer Anzeige sichtbar machen) könnten Wachen oder Fallen den Spieler nicht mehr bemerken.

Oder man hat ein Event, Wie zum Beispiel ein „Geisterpulse“, der dem Spieler Schaden zufügt, wenn er sich zum Zeitpunkt des Pulses im Schatten befindet.

Geräusche

Spieler macht beim Gehen/rennen/kriechen in Kombination mit der Bodenbeschaffenheit unterschiedlich viel Lärm. Könnte Gegner anlocken oder eben an Gegner vorbeischleichen.

Weiter könnten so „Geräuschverursacher“ wie Lagerfeuer oder Radio die Spielergeräusche übertönen und so ebenfalls ein Vorbeischleichen ermöglichen.

Inventar

Objekte oder Geräte, die vom Spieler gesammelt werden können, werden im Inventar abgelegt.

Können zu einem Späteren Zeitpunkt gebraucht werden.

**Beispiele**: Heiltränke, Stärkungstränke, Seile, Steine , Munition, Waffen

Charakter

Der Charakter kann 1-n Gegenstände aufnehmen, die ihn stärker oder anders „machen“. Waffen oder Rüstungen, oder Amulette oder Booster die die Heilung oder den Ausgeteilten Schaden beeinflussen.

Oder aber die Geschwindigkeit oder Verwundbarkeit des Spielers verändern.

**Beispiel**: Ritterrüstung: nur noch 20% des Schadens wird erlitten, aber die Geschwindigkeit wird halbiert. Damit könnte der Spieler eine zuvor tödliche Pfeilschuss-Falle überleben.

Buffs

Eignet sich für Puzzle level: Bestimmte Objekte (Brunnen, Automaten, Schreine) könnten spielereigenschaften (Speed, Angriff, Leben, Sprungkraft, Stärke) so verändern, dass zuvor unmögliche Spielabschnitte oder Hindernisse überwunden werden können.

Die Kleinere Variante wie Stärkungstränke/Booster könnten den Kampf erleichtern.

Geld

Erfolgreiches Abschliessen von Spielzielen kann mit Geld belohnt werden. Dieses kann für Ausrüstung oder Informationen oder Buffs ausgegeben werden und machen die nächsten Levels einfacher

Erfahrung

Erfolgreiches Abschliessen von Spielzielen kann mit Geld belohnt werden. Dieses kann für Ausrüstung oder Informationen oder Buffs ausgegeben werden und machen die nächsten Levels einfacher

Vorteil: Braucht kein Inventar sondern kann direkt den Charakter verstärken.

Leaderboard

Aufzeichen von Spieler via Spielername, für fast alle Eigenschaften: (Kills, Geld, Punkte, Zeit) Könnte eine Rangliste erstellt werden.