 Template

Mai 2022

**Über arc42**

arc42, das Template zur Dokumentation von Software- und Systemarchitekturen.

Template Version 8.1 DE. (basiert auf AsciiDoc Version), Mai 2022

Created, maintained and © by Dr. Peter Hruschka, Dr. Gernot Starke and contributors. Siehe <https://arc42.org>.

Diese Version des Templates enthält Hilfen und Erläuterungen. Sie dient der Einarbeitung in arc42 sowie dem Verständnis der Konzepte. Für die Dokumentation eigener System verwenden Sie besser die *plain* Version.

# Einführung und Ziele

## Aufgabenstellung

**Inhalt**

Dieses Dokument beschreibt eine Advanced Client – Server Spielvariante von Tron. Tron ist ein action rennspiel, bei dem Motorräder einen Schatten hinterlassen. Fahren andere Spieler gegen den Schatten, sind sie tot, sollten Spieler „crashen“ oder auch gegen die Wand fahren, sind diese auch tot und haben verloren. Der letzte Überlebende des Spiels, ist der Gewinner.

Das Spiel soll folgende Anforderungen erfüllen:

### Use Cases

**Nummer:** UC-1

**Titel:** Spieleranzahl festlegen

**Akteur:** Spieler

**Ziel:** Die maximale Anzahl der Spieler für die nächste Spielrunde soll festgelegt werden.

**Auslöser:** Intention eines Spielers, eine vom Standardwert abweichende Anzahl der Mitspieler festzulegen.

**Vorbedingung:**

* Die Anwendung wurde erfolgreich gestartet.
* Es wird Bildschirm 1 angezeigt.

**Nachbedingung:**

* Das Eingabefeld enthält einen ganzzahligen Wert von 2 bis 6 für die Spieleranzahl
* Die anwendungsinterne Spieleranzahl wurde auf den eingegebenen Wert geändert, sodass dieser beim nächsten Spielstart verwendet wird.

**Erfolgsszenario:**

1. System erzeugt GUI Fuer Bildschirm 1 und zeigt diese an
2. Das System zeigt ein Eingabefeld für die Spieleranzahl an, das zunächst den konfigurierten Standardwert enthält.
3. Der Spieler wählt per Mausklick das Eingabefeld an.
4. Der Spieler ersetzt den vorherigen Inhalt des Eingabefeldes mit einem ganzzahligen Wert von 2 bis 6.
5. Beim Mausklick außerhalb des Eingabefeldes wird der gültige Eingabewert für den nächsten Spielstart übernommen.

**Fehlerfälle:**

1. a) Der Spielers versucht, einen nicht-numerischen Wert einzugeben.
2. a) 1. Die Eingabe des Spielers erscheint nicht im Eingabefeld.
3. a) 2. Weiter bei 3.

3. b) Der Spieler macht eine Eingabe, die keiner Ganzzahl von 2 bis 6 entspricht.

1. b) 1. Das Eingabefeld wird rot umrandet und es wird ein Warnhinweis über die ungültige Eingabe angezeigt. Der Start-Button wird ausgegraut und deaktiviert.
2. b) 2. Weiter bei 3.

**Erweiterungsfälle:**

1. a) Nach Abschluss des Erfolgsszenarios ist immer noch ein Warnhinweis vorhanden.
2. a) 1. Der Warnhinweis und die rote Umrandung des Eingabefeldes werden ausgeblendet.

Der Startbutton wird wieder freigegeben.

**Nummer**: UC-2

**Titel**: Starten eines Spiels

**Akteur**: Der Spieler

**Ziel**: Der Spieler möchte ein Spiel spielen

**Auslöser**: Der Spieler hat das Spiel gestartet, weil er ein Spiel spielen will

**Vorbedingungen**:

* Spiel(Programm) wurde gestartet

**Nachbedingungen:**

* Wartebildschirm mit allen Spielern wird angezeigt

**Erfolgsszenario**

1. Das Spiel zeigt den Startbildschirm an
2. Der Spieler wählt das Feld für die Eingabe der maximalen Spieleranzahl aus
3. Der Spieler gibt eine Spieleranzahl an
4. Das Spiel überprüft die eingegebene Spieleranzahl
5. Der Spieler klickt auf den Button “Spiel starten”
6. Das. System erstellt die GUI fuer das Wartezeit Fenster
7. Das Spiel zeigt das Wartezeit Fenster an

**Fehlerfall**

1. Das System zeigt an, dass die angegebene Spieleranzahl kein korrekter Wert ist

**Zugrundeliegende Anforderungen:**

* UC-1 Spieleranzahl festlegen

**Nummer:** UC-3

**Titel:** Spiel Beitreten

**Aktuer:** Spieler

**Ziel:** Einem Spiel beizutreten.

**Auslöser:** Spieler hat die intention einem Spiel beizutreten

**Vorbedingung:**

* Bildschirm 2 wurde angezeigt
* Der Spieler hat die Anzahl der Spieler festgelegt
* Der Spieler hat ein Spiel gestartet

**Nachbedingung:**

* Der Benutzer ist dem Spiel beigetreten

**Erfolgszenario:**

1. Das System zeigt den Wartescreen an
2. Das System startet einen Countdown
3. Der jeweilige Spieler klickt auf eine Taste in einer der vordefinierten steuerungsbereichen
4. Das System merkt sich, dass ein neuer Spieler auf dem gerade gedrückten steuerungsbereich beigetreten ist
5. Sobald der Countdown zu ende ist, wird das Spiel gestartet

**Erweiterungsfälle:**

5.a. Es sind nicht alle Spieler beigetreten:

5.a.1 Das spiel wird nach einem bestimmten Countdown gestartet

**Fehlerfälle:**

1. b. Es sind nicht genug spieler eingetreten (weniger als 2):

5.b.1 Das Spiel wird abgebrochen und das System kehrt zurück zum start bildschirm

**Häufigkeit:**

* jedes mal wenn ein Spiel gestartet wird, müssen Spieler beitreten

**Zugrundeliegende Anforderungen**

* UC-1: Spieleranzahl festlegen
* UC-2: Spiel Starten

**Nummer**: UC-4

**Titel**: Bewegung eines spielers

**Akteur**: Der Spieler

**Ziel**: Der Spieler möchte, dass sich seine Spielfigur in der nächsten Bewegung in seine gewünschte Richtung bewegt

**Auslöser**: Spieler drückt eine Richtungstaste Vorbedingung:

**Vorbedingungen:**

* Der Spieler befindet sich in einem laufendem Spiel
* Der Countdown nach dem Start des Spiels ist abgelaufen Nachbedingung
* Die Spielefigur hat sich in die gewünschte Richtung bewegt

**Erfolgsszenario**

1. Das Spiel zeigt das Spiel an
2. Das Spiel zeigt den Spieler in der richtigen Richtung an

**Nummer:** UC-5

**Titel:** Spieler Sterben / verlieren

**Aktuer:** Spieler

**Ziel:** Ein Spieler stirbt im Spiel bzw. verliert das Spiel

**Auslöser:**

* Ein Spieler fährt gegen eine Wand
* Ein Spieler fährt gegein den Schatten eines anderen Spielers oder gegen seinen eigenen Schatten
* Ein Spieler fährt gegen ein anderes Motorrad
* Beide Spieler sterben wenn sie einen Frontalen zusammenstoß haben

**Vorbedingungen:**

* Der Spieler besitzt die Tron Applikation
* Der Spieler hat die Spieleranzahl wurde festgelegt
* Der Spieler hat das Spiel gestartet
* Spieler sind dem Spiel beigetreten

**Nachbedingungen:**

* Der Spieler kann nicht mehr weiter am Spiel teilnehmen
* Der Schatten des Spielers welcher gestorben ist, verschwindet aus dem Spiel

**Erfolgszenario:**

1. Ein Spieler spielt das Spiel und löst dabei einen der oben genannten Auslöser aus
2. Der Spieler verschwindet aus dem Spielfeld & sein Schatten verschwindet aus dem Spielfeld
3. Das Spiel wird weiter gespielt, ohne den gerade gestorbenen Spieler

**Erweiterungsfälle:**

1.a Wenn mehrere Spieler gleichzeitig sterben (durch gleiche oder verschiedene Auslöser)

1.a.1 Alle Spieler, welche gleichzeitig sterben, scheiden gleichzeitig aus dem Spiel aus & deren Schatten verschwindet aus dem Spiel

**Häufigkeit:**

* In jedem Spiel, welches Gespiel wird, werden Spieler sterben / verlieren.

**Zugrundeliegende Anforderungen:**

* UC-4 Bewegung eines spielers

**Nummer:** UC-6

**Titel:** Countdown zum Spielstart wird angezeigt

**Akteur:** System

**Ziel:** Die Spieler werden visuell auf den bevorstehenden Start des Spiels hingewiesen.

**Auslöser:** Abschluss des UC-3. (Spiel wurde gestartet)

**Vorbedingung:**

* Bildschirm 3 wird angezeigt.

**Nachbedingung:**

* Spiel ist gestartet.

**Erfolgsszenario:**

1. Das System erstellt die GUI fuer das Spielfeld
2. Bildschrim 3 (Spielfeld) wird angezeigt.
3. Es erscheinen in großen Buchstaben der Reihenfolge nach die Ausgaben “3”, “2”, “1”, “Go!” im Abstand von jeweils einer Sekunde.
4. Das Spiel startet.

**Häufigkeit**

* Bei jedem Spielstart.

**Zugrundeliegende Anforderungen**

* UC-3 Spieler treten dem Spiel bei

**Nummer**: UC-7

**Titel**: Startpositionen der Spieler

**Akteur**: Das System

**Ziel**: Alle Spieler besitzen Startpositionen, welche ein faires Spiel gewährleisten

**Auslöser**: Ein Spiel wurde gestartet

**Vorbedingungen**

* Ein Spiel wurde gestartet
* Das Spiel wurde geladen

**Nachbedingungen**: Spieler befinden sich auf fairen Startpositionen

**Erfolgsszenario**

1. Das System zeigt das Spiel an
2. Das System berechnet faire Startpositionen
3. Das System setzt die Spieler bei den Startpositionen ein

**Zugrundeliegende Anforderungen:**

* UC-2 starten eines Spiels

**Nummer:** UC-8

**Titel:** Spieler hinterlassen Schatten

**Aktuer:** Spieler

**Ziel:** Der Spieler bzw. das Motorrad des Spielers hinterlässt einen Schatten

**Vorbedingung:**

* Der Spieler besitzt die Applikation
* Der Spieler hat das Spiel gestartet
* Die Spieler sind dem Spiel beigetreten
* Die Spieler können sich auf dem Spielfeld bewegen

**Nachbedingungen:**

* Nach jeder Bewegung tuacht an der vorherigen Position des Spielers ein Schatten auf

**Erfolgszenario:**

1. Der Spieler bewegt sich auf dem Spielfeld
2. Bei jeder Bewegung des Spielers wird ein stück schatten mehr generiert und taucht hinter dem Spieler auf in der Farbe des Spielers

**Erweiterungsfälle:**

**Fehlerfälle:**

**Häufigkeit:**

* In jedem Spiel, welches gespielt wird, wird bei jedem Spieler ein Schatten hinterlassen

**Zugrundeliegende Anforderungen**

* UC-4 Bewegung eines spielers

**Nummer:** UC-9

**Titel:** Game over Screen

**Aktuer:** System

**Ziel:** Der Game over screen wird angezeigt mit den jeweiligen Gewinnern oder Unentschieden

**Auslöser:** Ein Spiel wurde beendet. bzw. in einem Spiel verbleiben weniger als 2 Spieler

**Vorbedingung:**

* Ein Spiel wurde gestartet und von mindestens 2 Spielern gespielt
* Das Spiel wurde beendet indem im Spiel weniger als 2 Spieler verblieben sind.

**Nachbedingungen:**

* Der Startbildschirm wird nach dem “Game over” screen angezeigt

**Erfolgszenario:**

1. Das Spiel wird beendet
2. Das System erstellt die GUI fuer den Game Over screen
3. Das System wechselt vom “Spielfeld” screen zum “Game over” screen
4. Das System startet einen 10 sek. countdown.
5. Auf dem Game Over Screen wird der Spieler angezeigt, welcher gewonnen hat. Dies wird durch “Spieler X” angezeigt, in der jeweiligen Spieler farbe
6. Sobald der 10 sek. countdown abgelaufen ist, wird auf den Startbildschirm der Applikation gewechsel.

**Erweiterungsfälle:**

4.a Wenn das Spiel mit einem Unentschieden beendet wurden ist

4.a.1: Anstatt “Spieler X” wird der Schriftzug “Unentschieden” angezeigt

**Fehlerfälle:**

**Häufigkeit:**

* Jedes mal, wenn ein Spiel gespielt wurden ist, wird dieser Screen angezeigt.

**Zugrundeliegende Anforderungen:**

* UC-5 Spieler sterben / verlieren

## Motivation

Die wesentliche Motivation für uns, dieses Spiel zu implementieren ist es, die PVL zu erhalten. Weitere Motivationspunkte wären aber auch, neues zu lernen und unser bisheriges Wissen zu vertiefen.

## Qualitätsziele

|  |
| --- |
| **Qualitätsziele:** |
| Gut definierte Schnittstellen |
| Kompatibilität zu einer anderen Gruppe (Mindestens zwei Teams müssen miteinander Spielen können) |
| Fehlertoleranz (Wenn ein Spieler abstürzt, egal welcher Spieler, dann geht das Spiel trotzdem weiter) -> Stabilität |
| Das spiel soll gleich schnell laufen für alle (keine Jitter-abhängigkeit) |
| Ein Spiel mit 6 Lueten, soll einmal komplett ohne Fehler durchlaufen. |

## Stakeholder

**Inhalt**

Unsere Stakeholder sind die Entwickler (Studenten), der Kunde (Professor) und die Spieler (Studenten).

Die Stakeholder mit deren Kontakt werden in der unteren Tabelle aufgelistet:

| Rolle | Kontakt | Erwartungshaltung |
| --- | --- | --- |
| *Entwickler* | [Dominik.martin@haw-hamburg.de](mailto:Dominik.martin@haw-hamburg.de) | *Ein gutes Spiel zu programmieren & die PVL zu erhalten* |
| *Entwickler* | [*Can.heintze@haw-hamburg.de*](mailto:Can.heintze@haw-hamburg.de) | *Ein gutes Spiel zu programmieren & die PVL zu erhalten* |
| *Entwickler* | [*Dominik.mueller@haw-hamburg.de*](mailto:Dominik.mueller@haw-hamburg.de) | *Ein gutes Spiel zu programmieren & die PVL zu erhalten* |
| *Kunde* | [*Martin.becke@haw-hamburg.de*](mailto:Martin.becke@haw-hamburg.de) | *Ein Lauffaehiges Spiel, bei dem 6 Spieler gleichzeitig ein komplettes Spiel ohne fehler durchspielen koennen.* |
| *Spieler* | *n/a* | *Ein funktionierendes Spiel spielen und dabei Spaß haben* |

# ------

# Randbedingungen

**Inhalt**

Randbedingungen und Vorgaben, die ihre Freiheiten bezüglich Entwurf, Implementierung oder Ihres Entwicklungsprozesses einschränken. Diese Randbedingungen gelten manchmal organisations- oder firmenweit über die Grenzen einzelner Systeme hinweg.

**Motivation**

Für eine tragfähige Architektur sollten Sie genau wissen, wo Ihre Freiheitsgrade bezüglich der Entwurfsentscheidungen liegen und wo Sie Randbedingungen beachten müssen. Sie können Randbedingungen vielleicht noch verhandeln, zunächst sind sie aber da.

**Form**

Einfache Tabellen der Randbedingungen mit Erläuterungen. Bei Bedarf unterscheiden Sie technische, organisatorische und politische Randbedingungen oder übergreifende Konventionen (beispielsweise Programmier- oder Versionierungsrichtlinien, Dokumentations- oder Namenskonvention).

Siehe [Randbedingungen](https://docs.arc42.org/section-2/) in der online-Dokumentation (auf Englisch!).

# Kontextabgrenzung

## A picture containing shape Description automatically generatedFachlicher Kontext

## Technischer Kontext

Graphical user interface

Description automatically generated

# Lösungsstrategie

### Funktionale Anforderungen

|  |  |  |  |
| --- | --- | --- | --- |
| **Methoden Signatur** | **Beschreibung** | **Instanz** | **UseCase** |
| void showStartScreen () | * Zeigt bildschirm 1 an: * Zeigt den Start Button an * Ermoeglicht dem Spieler auszuwaehlen, wie viele Spieler mitspielen sollen | ScreenHandler | UC-1 |
| void showLobbyScreens(int, int) | * Zeigt Bildschirm 2 und / oder 3 an. Also den Wartebildschirm fuer den Spieler beitritt oder den Endbildschirm, mit der Spieler Nummer vom Gewinner | ScreenHandler | UC-3 / UC-9 |
| Void showGameScreen(Map<Integer, int[][]> | * Zeigt den Game Bildschirm an * Zeichnet die jeweiligen Motorraeder * Bekommt eine Map, der key steht fuer den jeweiligen Spieler und der Value sind dann die jeweiligen Koordinaten fuer diesen Spieler | ScreenHandler | UC-4 |
| Void init(Stage) | * Initialisiert die View | ScreenHandler | UseCase Uebergreifend |
| Boolean checkInputString(String) | * Prueft, ob der Input des Spielers valide ist. (also ob die korrekten pfeiltasten gedrueckt wurden sind etc.) | InputHandler | UC-4 / UC-3 |
| Void forwardInput (String) | * Gibt den Spieler input an den Controller weiter | InputHandler | UC-4 / UC-3 |
| Void handle(KeyEvent) | * Nimmt den Spieler input entgegen und ruft forwardInput & checkInputString auf | InputHandler | UC-4 / UC-3 |
| Void endgame() | * Das Spiel wird beendet, indem es den 4 Bildschirm anzeigt mit dem jeweiligen Gewinner etc. * Ruft createWinnerScreen auf | Lobby | UC-9 / UC-5 |
| Void createWinnerScreen() | * Ueberprueft den Gewinner Status aus der gamelogic und zeigt den 4ten Bildschirm an | Lobby | UC-9 / UC-5 |
| Lobby initLobby() | * Zeigt Bildschirm 2 an (wartescreen) | Lobby | UC-3 |
| Void playerJoin(int) | * Erstellt die jeweiligen spieler, wenn diese dem Spiel aus dem 2ten Bildschirm beitreten * Die view wird dabei aktualisiert um die beigetretenen Spieler anzuzeigen | Lobby | UC-3 |
| Void setCurrentPlayerCount(int) | * Setzt den CurrentPlayerCount | Lobby | Use Case Uebergreifend |
| Void updateView(int) | * Ruft showStartScreen(int) auf | Lobby | Use Case Uebergreifend |
| Map<Integer, List<String>> getPlayerMappings() | * Gibt die Playermappings zurueck * Die Playermappings sind die jeweiligen tastenbelegungen fuer die jeweiligen Spieler | Config | Use Case Uebergreifend |
| Int getLobbyTimerDuration() | * Gibt die sekunden anzahl fuer die Dauer der Lobby (wartebildschirm) an | Config | UC-3 |
| Int getPlayerCount() | * Gibt die Maximal moegliche Spieleranzahl zurueck (6) | Config | UC-3 / Use Case Uebergreifend |
| Void setPlayerCount() | * Setzt den playerCount | Config | Use Case Uebergreifend |
| Int[] getBoardSize() | * Gibt die groeße des Boards zurueck | Config | UC-7 |
| Int getCellSize() | * Gibt die groeße der jeweiligen Zellen zurueck | Config | Use Case Uebergreifend |
| Void loadConfigFile(String) | * Laded die Config Datei und setzt die jeweiligen Werte | Config | Use Case Uebergreifend |
| Void startLobbyTimer(int, Runnable) | * Startet den Timer fuer den zweiten Bildschirm (Wartebildschirm) | ThreadedTimerImpl | UC-3 |
| Void startGameTimer(int, Runnable) | * Starten den Timer fuer das Spiel * Startet einen Thread, welcher fuer die Aktualisierung der Ticks im Spiel sind | ThreadedTimerImpl | UC-4 / UC-8 |
| StopGameTimer | * Stoppt den Spiel Timer | ThreadedTimerImpl | UC-4 / UC-8 |
| Void setPlayerMove(int,int) | * Ruft setIntendedDirection * Setzt die Bewegung fuer einen Spieler | PlayerManager | UC-4/UC-8 |
| Void setIntendedDirection(int, int) | * Setzt die Bewegung fuer einen Spieler | PlayerManager | UC-4/UC-8 |
| List<Players> getLivinPlayers() | * Gibt eine Liste an Spieler zurueck, welche noch leben | PlayerManager | Use Case Uebergreifend |
| Void notifyCollisions(int[][]) | * Setzt den lebensstatus auf False, wenn eine Kolision vorhanden ist | PlayerManager | UC-5 |
| Void commitPlayerMoves() | * Ruft commitMove auf die jeweiligen Spieler auf und setzt die Richtung fuer den Spieler | PlayerManager | UC-4 |
| Void createPlayer(List<String>, int) | * Erzeugt spieler | PlayerManager | UC-3 |
| List<int[]> getPlayerPositions(int) | * Gibt eine Liste an positionen des jeweiligen Spieler zurueck | PlayerManager | Use Case Uebergreifend |
| Void killPlayer(int) | * Setzt den Lebensstatus eines Spielers auf False | PlayerManager | UC-5 |
| Plaer getPlayerById(int) | * Gibt den jeweiligen Spieler einer Spieler ID zurueck | PlayerManager | Use Case Uebergreifend |
| Map<Integer, int[][]> checkPlayerCollision(List<int[]>) | * ????? | PlayerManager | UC-5/UC-4 |
| Void killPlayers() | * Loescht alle Spieler | PlayerManager | UC-5 / UC-9 |
| List<Player> getPlayers() | * Gibt eine Liste an Spielern zurueck | PlayerManager | Use Case Uebergreifend |
| Void commitMove(int) | * Setzt die jeweilige Richtung fuer einen Spieler, dabei wird die Richtung als int der Methode uebergeben. * Es gibt noch eine ueberladung mit void commitMove() | Player | UC-4 |
| Void changePosition(int,int) | * Aendert die Position auf die neuen x,y Koordinaten eines Spielers | Player | UC-4 |
| Bool checkBackwards(int[]) | * Prueft ob ein Spieler versucht, einen Schritt zurueck zu machen. * Wenn ja, dann wird das Motorrad trotzdem ein Schritt in die bisherige Richtung gesetzt | Player | UC-4 |
| Void resetNextId() | * Reseted die nextId variable. * Dies wird genutzt, damit, wenn ein neues Spiel startet, die spieler, wieder mit ID’s 1-6 erstellt werden | Player | Use Case Uebergreifend |
| Void startGame() | * Das Spiel wird gestartet * Die Startpositionen der Spieler wird berechnet indem calculateStartPositions aufgerufen wird | GameLogic | UC-2 |
| Bool updateTick() | * Dies aktualisiert den Spiel Status mit den aktuellen Werten fuer das Board (Positionen etc.), nach jedem Tick | GameLogic | Use Case Uebergreifend |
| List<int[]> checkCollision | * Prueft auf eine Kollision und gibt eine Liste mit Koordinaten, an denen die Kollision stattgefunden hat | GameLogic | UC-5 |
| Void calculateStartPositions() | * Berechnet die Startpositionen fuer die jeweiligen Spieler am Anfang des Spiels | GameLogic | UC-7 |
| Int[] getWinnerStatus() | * Gibt den Ausgangsstatus des Spiels zurueck | GameLogic | Use Case Uebergreifend |
| Void removePlayers() | * Ruft killPlayers auf den PlayerManager auf | GameLogic | UC-5 / UC-9 |
| Void updateBoard() | * Aktualisiert das Board mit den jeweils neuen Positionen der Spieler und Schatten | GameLogic | UC-4 / UC-8 |
| Void setStartPosition(List<Player>, int) | * Setzt die Start Positionen der Spieler nach der Berechnung der Start Positionen | GameLogic | UC-2 / UC-4 |
| Void remainingTime(int) | * Zeigt den Wartebildschirm an | Board | UC-3 |
| Void updateBoard(List<int[]>) | * Aktulisiert das Spiel Feld mit den jeweiligen positionen der Motorraeder und Schatten indem es diese einer „Obstacles“ Liste anfuegt. | Board | UC-4 / UC-8 |
| Board initBoard() | * Initialisiert das Spiel Feld | Board | Use Case Uebergreifend |
| Void updateView(int, Map<Integer, int[][]> | * Ruft showGameScreen auf den ScreenHandler auf und zeichnet somit die Motorraeder | Board | UC-4 / UC-8 |
| Void updateVIew(int, int) | * Ruft showLobbyScreens auf * Wird genutzt, wenn das Spiel beendet ist und auf den Endscreen gewechselt werden soll | Board | UC-9 |
| List<int[]> getObstacles() | * Gibt eine Liste mit Koordinaten zurueck, an denen sich die „Obstacles“ auf dem Spielfeld befinden | Board | Use Case Uebergreifend |
| Int[] getBoardSize() | * Gibt die Groeße des Spiel Felds zurueck | Board | Use Case Uebergreifend |
| Void clearBoard() | * Loescht alles aus der Liste der Obstacles raus und rauemt somit das Spiel Feld leer | Board | UC-5 / UC-9 |
| Void removeObstacles(List<int[]>) | * Loescht die Obstacles an einer bestimmten Position | Board | UC-5 |
| Void playerJoin(int) | * Wird aufgerufen, wenn ein Spieler beitritt | GameManagerImpl | UC-3 |
| Void loadLobby() | * Ruft initLobby auf die Lobby auf und zeigt somit den Wartebildschirm an | GameManagerImpl | UC-3 |
| Bool isReadyToPlay() | * Prueft ob die derzeitige Spieler anzahl groeßer 1 ist und kleiner gleich die Maximal Spieler Anzahl | GameManagerImpl | UC-3 |
| Void startGame() | * Initialisiert das Spiel Feld | GameManagerImpl | UC-2 / UC-3 |
| Void updateView(int) | * Ruft updateView auf der Lobby auf | GameManagerImpl | Use Case Uebergreifend |
| Void removePlayers() | * Setzt den currentPlayerCount auf 0 und entfernt die Spieler | GameManagerImpl | Use Case Uebergreifend |
| Void endgame() | * Sauebert das Spiel Feld und laesst die Spieler entfernen | GameManagerImpl | UC-9 |
| Void onKeyPress(String) | * Hier werden Aktionen ausgefuehrt, sobald ein Spieler auf eine Taste klickt. * Je nach situation, bewegt der Spieler damit sein Motorrad oder er tritt dem Spiel bei | PlayerInputManagerImpl | UC-3 / UC-4 |
| List<String> getValidKeys() | * Gibt zugelassene Tasten zurueck | PlayerInputManagerImpl | Use Case Uebergreifend |
| Void switchPlayerController() | * Wechselt den PlayerController, so, dass es im Wartebildschirm den PlayerController fuer das Spiel Beitreten nutzt und im Spiel dann den PlayerController fuer das Bewegen der Spieler nutzt | PlayerInputManagerImpl | UC-3/UC-4 |
| IPlayerController createPlayerController(String) | * Erstellt die jeweiligen PlayerController je nach Bedarf | PlayerCOntrollerFactory | Use Case Uebergreifend |
| Void loadMappings() | * Laded und setzt die Tastenbelegungen der jeweiligen Spieler | IPlayerController | Use Case Uebergreifend |
| Int gerDirectionForKey(String) | * Gibt fuer die jeweilige Taste die dazugehoerige Richtung zurueck | PlayerControllerMovementImpl | UC-4 |
| Int getPlayerForKey(String) | * Gibt die PlayerID fuer die jeweilige Taste zurueck | PlayerControllerMovementImpl | UC-4 |
| Int getPlayerForKey(String) | * Gibt die PlayerID fuer die jeweilige Taste zurueck * Wird beim Spiel Beitritt genutzt | PlayerControllerOnbaordingImpl | UC-3 |
| Int changeToNextScene() | * Wechselt zur naechsten Szene, also zum naechsten Bildschirm | SceneChangerImpl | Use Case Uebergreifend |
| Int commitNadChangeToNextScene(int) | * Setzt den PlayerCount, welcher vom Spieler im Start Bildschirm angegeben wurde und wechselt zur naechsten Szene | SceneChangerImpl | UC-1 |
| Int changeToPreviousScene() | * Wechselt zur vorherigen Szene, also zum vorherigen Bildschirm | SceneChangerImpl | Use Case Uebergreifend |
| Void registerNextSceneCallback(Runnable) | * Registriert den Callback fuer die naechste Szene | SceneChangerImpl | Use Case Uebergreifend |
| Void registerPreviousSceneCallback  (Runnable) | * Registriert den Callback fuer die vorherige Szene | SceneChangerImpl | Use Case Uebergreifend |
| Void init() | * Initialisiert die jeweiligen Szenen | SceneChangerImpl | Use Case Uebergreifend |
| Void updateTick() | * Stopt den Game Timer wenn das Spiel zu ende sit | SceneChangerImpl | UC- 5 |

# Bausteinsicht

## Komponten Diagramm:

Dies ist das Komponenten Diagramm, welches die unsere Komponenten auf Hoechster Ebene zeigt.

## Diagram Description automatically generated

Hier folgen nun die Komponenten Diagramme auf zweiter Ebene:

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram, engineering drawing

Description automatically generated

Nun Folgen unsere Klassediagramme:

View

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram, schematic

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

# Laufzeitsicht

In diesem kapitel beschreiben wir, was in der Tron Standalone Aplikation waehrend der Luafzeit passiert. Dies visualisieren wir mittels Sequenzdiagramme.

Die Sequenzdiagramme zeigen die Methodenaufrufe zwischen den Komponenten auf Zwei Ebenen auf.

Sequenzdiagramme der ersten Ebene:

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Sequenzdiagramme der zweiten Ebene:

Diagram

Description automatically generated

Diagram

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram, schematic

Description automatically generated

Diagram

Description automatically generated

Graphical user interface, diagram, text

Description automatically generated

# Verteilungssicht

**Inhalt**

Die Verteilungssicht beschreibt:

1. die technische Infrastruktur, auf der Ihr System ausgeführt wird, mit Infrastrukturelementen wie Standorten, Umgebungen, Rechnern, Prozessoren, Kanälen und Netztopologien sowie sonstigen Bestandteilen, und
2. die Abbildung von (Software-)Bausteinen auf diese Infrastruktur.

Häufig laufen Systeme in unterschiedlichen Umgebungen, beispielsweise Entwicklung-/Test- oder Produktionsumgebungen. In solchen Fällen sollten Sie alle relevanten Umgebungen aufzeigen.

Nutzen Sie die Verteilungssicht insbesondere dann, wenn Ihre Software auf mehr als einem Rechner, Prozessor, Server oder Container abläuft oder Sie Ihre Hardware sogar selbst konstruieren.

Aus Softwaresicht genügt es, auf die Aspekte zu achten, die für die Softwareverteilung relevant sind. Insbesondere bei der Hardwareentwicklung kann es notwendig sein, die Infrastruktur mit beliebigen Details zu beschreiben.

**Motivation**

Software läuft nicht ohne Infrastruktur. Diese zugrundeliegende Infrastruktur beeinflusst Ihr System und/oder querschnittliche Lösungskonzepte, daher müssen Sie diese Infrastruktur kennen.

**Form**

Das oberste Verteilungsdiagramm könnte bereits in Ihrem technischen Kontext enthalten sein, mit Ihrer Infrastruktur als EINE Blackbox. Jetzt zoomen Sie in diese Infrastruktur mit weiteren Verteilungsdiagrammen hinein:

* Die UML stellt mit Verteilungsdiagrammen (Deployment diagrams) eine Diagrammart zur Verfügung, um diese Sicht auszudrücken. Nutzen Sie diese, evtl. auch geschachtelt, wenn Ihre Verteilungsstruktur es verlangt.
* Falls Ihre Infrastruktur-Stakeholder andere Diagrammarten bevorzugen, die beispielsweise Prozessoren und Kanäle zeigen, sind diese hier ebenfalls einsetzbar.

Siehe [Verteilungssicht](https://docs.arc42.org/section-7/) in der online-Dokumentation (auf Englisch!).

## Infrastruktur Ebene 1

An dieser Stelle beschreiben Sie (als Kombination von Diagrammen mit Tabellen oder Texten):

* die Verteilung des Gesamtsystems auf mehrere Standorte, Umgebungen, Rechner, Prozessoren o. Ä., sowie die physischen Verbindungskanäle zwischen diesen,
* wichtige Begründungen für diese Verteilungsstruktur,
* Qualitäts- und/oder Leistungsmerkmale dieser Infrastruktur,
* Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur

Für mehrere Umgebungen oder alternative Deployments kopieren Sie diesen Teil von arc42 für alle wichtigen Umgebungen/Varianten.

***<Übersichtsdiagramm>***

Begründung

*<Erläuternder Text>*

Qualitäts- und/oder Leistungsmerkmale

*<Erläuternder Text>*

Zuordnung von Bausteinen zu Infrastruktur

*<Beschreibung der Zuordnung>*

## Infrastruktur Ebene 2

An dieser Stelle können Sie den inneren Aufbau (einiger) Infrastrukturelemente aus Ebene 1 beschreiben.

Für jedes Infrastrukturelement kopieren Sie die Struktur aus Ebene 1.

### *<Infrastrukturelement 1>*

*<Diagramm + Erläuterungen>*

### *<Infrastrukturelement 2>*

*<Diagramm + Erläuterungen>*

…

### *<Infrastrukturelement n>*

*<Diagramm + Erläuterungen>*

# Querschnittliche Konzepte

**Inhalt**

Dieser Abschnitt beschreibt übergreifende, prinzipielle Regelungen und Lösungsansätze, die an mehreren Stellen (=*querschnittlich*) relevant sind.

Solche Konzepte betreffen oft mehrere Bausteine. Dazu können vielerlei Themen gehören, beispielsweise:

* Modelle, insbesondere fachliche Modelle
* Architektur- oder Entwurfsmuster
* Regeln für den konkreten Einsatz von Technologien
* prinzipielle — meist technische — Festlegungen übergreifender Art
* Implementierungsregeln

**Motivation**

Konzepte bilden die Grundlage für *konzeptionelle Integrität* (Konsistenz, Homogenität) der Architektur und damit eine wesentliche Grundlage für die innere Qualität Ihrer Systeme.

Manche dieser Themen lassen sich nur schwer als Baustein in der Architektur unterbringen (z.B. das Thema „Sicherheit“).

**Form**

Kann vielfältig sein:

* Konzeptpapiere mit beliebiger Gliederung,
* übergreifende Modelle/Szenarien mit Notationen, die Sie auch in den Architektursichten nutzen,
* beispielhafte Implementierung speziell für technische Konzepte,
* Verweise auf „übliche“ Nutzung von Standard-Frameworks (beispielsweise die Nutzung von Hibernate als Object/Relational Mapper).

**Struktur**

Eine mögliche (nicht aber notwendige!) Untergliederung dieses Abschnittes könnte wie folgt aussehen (wobei die Zuordnung von Themen zu den Gruppen nicht immer eindeutig ist):

* Fachliche Konzepte
* User Experience (UX)
* Sicherheitskonzepte (Safety und Security)
* Architektur- und Entwurfsmuster
* Unter-der-Haube
* Entwicklungskonzepte
* Betriebskonzepte



Siehe [Querschnittliche Konzepte](https://docs.arc42.org/section-8/) in der online-Dokumentation (auf Englisch).

## *<Konzept 1>*

*<Erklärung>*

## *<Konzept 2>*

*<Erklärung>*

…

## *<Konzept n>*

*<Erklärung>*

# Architekturentscheidungen

**Inhalt**

Wichtige, teure, große oder riskante Architektur- oder Entwurfsentscheidungen inklusive der jeweiligen Begründungen. Mit "Entscheidungen" meinen wir hier die Auswahl einer von mehreren Alternativen unter vorgegebenen Kriterien.

Wägen Sie ab, inwiefern Sie Entscheidungen hier zentral beschreiben, oder wo eine lokale Beschreibung (z.B. in der Whitebox-Sicht von Bausteinen) sinnvoller ist. Vermeiden Sie Redundanz. Verweisen Sie evtl. auf Abschnitt 4, wo schon grundlegende strategische Entscheidungen beschrieben wurden.

**Motivation**

Stakeholder des Systems sollten wichtige Entscheidungen verstehen und nachvollziehen können.

**Form**

Verschiedene Möglichkeiten:

* ADR ([Documenting Architecture Decisions](https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions)) für jede wichtige Entscheidung
* Liste oder Tabelle, nach Wichtigkeit und Tragweite der Entscheidungen geordnet
* ausführlicher in Form einzelner Unterkapitel je Entscheidung

Siehe [Architekturentscheidungen](https://docs.arc42.org/section-9/) in der arc42 Dokumentation (auf Englisch!). Dort finden Sie Links und Beispiele zum Thema ADR.

# Qualitätsanforderungen

**Inhalt**

Dieser Abschnitt enthält möglichst alle Qualitätsanforderungen als Qualitätsbaum mit Szenarien. Die wichtigsten davon haben Sie bereits in Abschnitt 1.2 (Qualitätsziele) hervorgehoben.

Nehmen Sie hier auch Qualitätsanforderungen geringerer Priorität auf, deren Nichteinhaltung oder -erreichung geringe Risiken birgt.

**Motivation**

Weil Qualitätsanforderungen die Architekturentscheidungen oft maßgeblich beeinflussen, sollten Sie die für Ihre Stakeholder relevanten Qualitätsanforderungen kennen, möglichst konkret und operationalisiert.

**Weiterführende Informationen**

Siehe [Qualitätsanforderungen](https://docs.arc42.org/section-10/) in der online-Dokumentation (auf Englisch!).

## Qualitätsbaum

**Inhalt**

Der Qualitätsbaum (à la ATAM) mit Qualitätsszenarien an den Blättern.

**Motivation**

Die mit Prioritäten versehene Baumstruktur gibt Überblick über die — oftmals zahlreichen — Qualitätsanforderungen.

* Baumartige Verfeinerung des Begriffes „Qualität“, mit „Qualität“ oder „Nützlichkeit“ als Wurzel.
* Mindmap mit Qualitätsoberbegriffen als Hauptzweige

In jedem Fall sollten Sie hier Verweise auf die Qualitätsszenarien des folgenden Abschnittes aufnehmen.

## Qualitätsszenarien

**Inhalt**

Konkretisierung der (in der Praxis oftmals vagen oder impliziten) Qualitätsanforderungen durch (Qualitäts-)Szenarien.

Diese Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht.

Wesentlich sind zwei Arten von Szenarien:

* Nutzungsszenarien (auch bekannt als Anwendungs- oder Anwendungsfallszenarien) beschreiben, wie das System zur Laufzeit auf einen bestimmten Auslöser reagieren soll. Hierunter fallen auch Szenarien zur Beschreibung von Effizienz oder Performance. Beispiel: Das System beantwortet eine Benutzeranfrage innerhalb einer Sekunde.
* Änderungsszenarien beschreiben eine Modifikation des Systems oder seiner unmittelbaren Umgebung. Beispiel: Eine zusätzliche Funktionalität wird implementiert oder die Anforderung an ein Qualitätsmerkmal ändert sich.

**Motivation**

Szenarien operationalisieren Qualitätsanforderungen und machen deren Erfüllung mess- oder entscheidbar.

Insbesondere wenn Sie die Qualität Ihrer Architektur mit Methoden wie ATAM überprüfen wollen, bedürfen die in Abschnitt 1.2 genannten Qualitätsziele einer weiteren Präzisierung bis auf die Ebene von diskutierbaren und nachprüfbaren Szenarien.

**Form**

Entweder tabellarisch oder als Freitext.

# Risiken und technische Schulden

**Inhalt**

Eine nach Prioritäten geordnete Liste der erkannten Architekturrisiken und/oder technischen Schulden.

Risikomanagement ist Projektmanagement für Erwachsene.

— Tim Lister Atlantic Systems Guild

Unter diesem Motto sollten Sie Architekturrisiken und/oder technische Schulden gezielt ermitteln, bewerten und Ihren Management-Stakeholdern (z.B. Projektleitung, Product-Owner) transparent machen.

**Form**

Liste oder Tabelle von Risiken und/oder technischen Schulden, eventuell mit vorgeschlagenen Maßnahmen zur Risikovermeidung, Risikominimierung oder dem Abbau der technischen Schulden.

Siehe [Risiken und technische Schulden](https://docs.arc42.org/section-11/) in der online-Dokumentation (auf Englisch!).

# Glossar

**Inhalt**

Die wesentlichen fachlichen und technischen Begriffe, die Stakeholder im Zusammenhang mit dem System verwenden.

Nutzen Sie das Glossar ebenfalls als Übersetzungsreferenz, falls Sie in mehrsprachigen Teams arbeiten.

**Motivation**

Sie sollten relevante Begriffe klar definieren, so dass alle Beteiligten

* diese Begriffe identisch verstehen, und
* vermeiden, mehrere Begriffe für die gleiche Sache zu haben.
* Zweispaltige Tabelle mit <Begriff> und <Definition>
* Eventuell weitere Spalten mit Übersetzungen, falls notwendig.

Siehe [Glossar](https://docs.arc42.org/section-12/) in der online-Dokumentation (auf Englisch!).

| Begriff | Definition |
| --- | --- |
| *<Begriff-1>* | *<Definition-1>* |
| *<Begriff-2* | *<Definition-2>* |