Aufgabe 4:

**Kommentar-Konventionen**

**Kommentare bei einer Prozedur:**

Bei einer Prozedur bzw. Methode soll die Funktion der Methode kurz beschrieben werden, um bessere Nachvollziehbarkeit für andere Programmierer zu sichern.

**Kommentare bei Variablen:**

Kurze Beschreibung des Variablentyps und der Funktion der Variable. Die Änderung der Variable wird ebenfalls kommentiert. Kommt eine Änderung oft, auf gleiche Weise vor(Bsp.: „increase counter by 1“), wird nur einmal kommentiert.

**Kommentare bei Chaptern:**

Der Beginn und das Ende eines Chapters wird als solches durch entsprechende Kommentare gekennzeichnet.

**Kommentare bei Räumen:**

Räume, die für die Story wichtig sind und/oder wichtige Gegenstände beinhalten(Spind, Heilungskabine etc.) werden kommentiert. Ebenso Räume mit Kontaminierten.

**Kommentare bei Gegenständen:**

Gegenstände werden mit ihrem Ort und ihrer Funktion kommentiert.

**Kommentare zur Story:**

Bei einem Characterwechsel wird dieser kommentiert. Ebenso an wichtigen Ereignissen der Story bspw. Der Sauerstoffabfall, die Dekontamination, .

**Programmierkonventionen**

**Namen:**

- Namenskonventionen für Objekte und Räume

- existieren mehrere gleiche Dinge: Durchnummerierung

<Name>\_<Nummer>

- Namen aus Spezifikation/Lastenheft

- keine Namen vorhanden 🡪 sinnvolle Namen (ähnlich wie andere)

- Leerzeichen als ‘\_‘

- printed name mit Leerzeichen

- eindeutige Namen für Variablen (Bsp.: Zähler für Schritte: Schrittzähler)

**Kommentare:**

- Kommentieren an bestimmten Stellen

- siehe Kommentierkonvention

- Segmente (Definitionen, Kontaminierte, …) mit [\*\*\*\*\*<Name>\*\*\*\*\*] kennzeichnen

(- darunter einrücken: gilt für Mehrfach-Definitionen: Definitionen, Aktionen, …)

- Block mit [<Name>] kennzeichnen

**Code:**

- Code-Priorisierung (wenn möglich auch 2.)

- 1.funktionierend und übersichtlich

- 2.so kurz wie möglich

- Groß- und Kleinschreibung beachten (Satzanfang groß, Objekte und Eigennamen groß)

- Code möglichst so schreiben, dass keine Kommentare notwendig sind

- Einrücken von zusammengehörenden Dingen und Blöcken

- zusammengehörende Dinge zusammenschreiben

- zusammengehörende Blöcke und Quelltext

- Definitionen, Aktionen, … zusammenschreiben

- Platz lassen zwischen Code-Segmenten und -Blöcken

- zwischen Blöcken 1-2 Zeilen

- zwischen Segmenten 3-4 Zeilen

- ähnliche Code-Blöcke sollen gleich aussehen

- orientieren an existierenden Blöcken

- keine doppelten Dinge 🡪 Wiederverwertung

- nicht mehrere Anweisungen (=Sätze) in einer Zeile

- Ausnahme: Objektdefinition und printed name

- globale Variablen zuordnen:

- nur von einem Segment genutzt: zu Segment

- dürfen von anderen Segmenten nicht genutzt werden

- von mehreren Segmenten: nach ganz oben

- vor Abgabe des Codes: Testen

Blöcke: eine große Anweisung

Schleife, mehrere Bedingungen, every turn, before …

Segmente: Code und ein oder mehrere Blöcke im sinnlichen Zusammenhang

Bsp.: Programmierung Kontaminierter

**Versionsverwaltung:**

Die Software wird in einem GIT-Repository auf der Onlineplattform GitHub gespeichert. Dort wird sie dann auch verwaltet, dies hat den Vorteil das dort mehrere Teammitglieder simultan an unterschiedlichen Aufgaben arbeiten können. Die einzelnen Kapitel/Szenen haben ihre eigenen Branches die unabhängig voneinander programmiert werden. Dieser Code in jedem Branch wird von einem anderen Teammitglied jedes Wochenende auf die Konventionen überprüft. Erst am Ende, wenn der Code komplett überprüft wurde und alles fertig ist werden die einzelnen Kapitel zusammengeführt.

**Allgemeines für den Projektplan:**

**DoD:**

Damit ein Punkt als fertig angesehen werden kann muss er folgende Kriterien erfüllen:

* der Inform7 Code muss einwandfrei generiert werden
* es muss alles aus der Spezifikation für den jeweiligen Gegenstand/ Ereignis / Raum / Tür übernommen sein
* es den obigen Konventionen entspricht
* es bei der wöchentlichen Codeüberprüfung abgenommen wurde

**Worst Case Szenarien:**

* Mitglied verhindert (Krankheit, Termine, Notfälle)
  + Wahrscheinlichkeit: 40%
  + Verhindern:
    - / nicht möglich da meistens nicht vorhersehbar
  + Lösungsansätze:
    - Kommunikation
    - Puffer in der Projektplanung
    - Umverteilung der Aufgabe
* Konflikte (innerteamlich)
  + Wahrscheinlichkeit: 40%
  + Verhindern:
    - Kommunikation (Treffen und über die Probleme reden)
  + Lösungsansätze:
    - Unabhängige Person zum Lösen dazu holen
* Technische Probleme (Programme funktionieren nicht, Daten gehen verloren)
  + Wahrscheinlichkeit: 60%
  + Verhindern:
    - Sich mit den Programmen vertraut machen
    - Sicherheitskopien
    - Online Repository
  + Lösungsansätze:
    - Die gespeicherten Daten aus dem Repository holen
    - Auf eine funktionierende Version zurücksetzen
* Deadline wird nicht eingehalten
  + Wahrscheinlichkeit: 60%
  + Verhindern:
    - Realistische Zeitrahmen in der Projektplanung
    - Kommunikation mit Teammitgliedern falls es zu knapp/ viel wird
      * Unterstützung des Teammitglieds
  + Lösungsansätze:
    - Puffer in der Projektplanung
    - Unterstützung des Teammitglieds
* Qualität entspricht nicht den Vorgaben/ Vorstellungen
  + Wahrscheinlichkeit: 50%
  + Verhindern:
    - Regelmäßige Überprüfung des Codes
    - Kommunikation mit Teammitgliedern falls es zu schwierig wird
      * Unterstützung des Teammitglieds
  + Lösungsansätze:
    - Puffer in der Projektplanung für eine Überarbeitung