Roadmap allgemein

Wer arbeitet woran?

- Installation, Start unter Linux und macOS, Datenbank, User Experience: Jörg (Code of Conduct)
- Klären der branch-bedingten Versionsverteilung im Code Repository

Roadmap Plattformunabhängikeit

Dies soll als Orientierung dienen, wie wir die Thera-Pi Software auch auf Linux und macOS verlässlich verfügbar machen können. Installation und Verwendbarkeit auf mobilen Geräten wie Smartphones und Tablets ist ein separates Thema.

Ziele

Die Ziele zur Herstellung der Plattformunabhängigkeit sind

- Compilierbarkeit für die drei Targets *windows*, *linux*, *macos* (inkl. Anpassung des Builders dafür)
- Erstellen von Installationspaketen (z. B. ZIP für Windows, DEB- und RPM-Pakete für Linux)
- einfache Installation auf den drei Plattformen (inkl. einfacher Anleitung üfr due Nutzer)

Hierbei geht es vorerst um die 32bit-Version der Software. Die Überführung in eine verlässlich compilierende und funktionierende Version wird hier nicht behandelt.

Mehrstufiges Vorgehen

Aus dem Forum, Chat und Gitlab Issues geht hervor, dass bereits mehrfach versucht wurde, die Plattformunabhängigkeit herzustellen, dies bisher aber noch nicht zu einem zufriedenstellenden Ergebnis gelangt ist. Da sich auch Linux und macOS in einigen Punkten unterscheiden und bereits mehrere Probleme beim Installieren und Ausführen der Software auf Linux zeigten, bietet sich eine mehrstufige Vorgehensweisen an, das Ziel zu erreichen.

Separates Vorgehen für Linux und macOS

Als unixoide Betriebssysteme sind sich Linux und macOS „unter der Haube“ recht ähnlich. Dennoch unterscheiden sie sich in mehreren Punkten, welche möglicherweise in der Konzeption der Software noch nicht erfasst sind. Da sich die Software viel einfacher und kostenfrei unter Linux (z.B. in VirtualBox VMs sowohl für 32bit als auch 64bit) testen lässt, als auf Apple Hard- und Software, bietet sich an, die Portabilitätsprobleme zuerst für Linux zu beheben. Damit sollten, wenn überhaupt, nur noch wenige macOS-spezifische Punkte verbleiben, die im Anschluss angegangen werden können.

Linux

Konfiguration

- Sämtliche im Quellcode hinterlegten Pfade sollten in Java-tauglicher, plattformunabhängiger Form hinterlegt sein. D. h. es sollten keine Backslashes verwendet werden.
  - a) via *File.separator* (etwas Overhead, schlechter lesbar, nicht wirklich notwendig)
  - **b)** durch konsistente Verwendung des „/“ in allen Pfadangaben (der gängige Weg, da dies von der JVM immer, auf allen Host-Plattformen, korrekt interpretiert wird, einfacher lesbar / pflegbar)
- Alle Pfadangaben sollten an zentraler, konfigurationsbezogener Stelle hinterlegt sein - nicht mehr hart-kodiert an der Stelle im Code, wo sie jeweils benötigt werden.

Bibliotheken

- Identifizieren aller DLLs (Windows-spezifische dynamisch gelinkte Bibliotheken)
- Bereitstellen der entsprechenden .so Libraries für Linux

Installationspakete (DEB/RPM)

Für eine einfache Installation der Software auf gängigen Linux_Distributionen sollten DEB- und RPM-Pakete im automatischen Build-Prozess erstellt werden.

macOS

- Ausprobieren einer Zip & Go Installation auf macOS
- ggf. Erstellen eines Installers oder macOS-spezifischen Software-Pakets
- Nach Erreichen der Plattformunabhängigkeit für Windows und Linux können noch auftretende Probleme auf macOS angegangen werden.

Roadmap Testing

Um für zukünftige Releases neuer Features oder Versionen der Software relativ zeitnah zu gewährleisten, dass bestehende grundlegende Funktionalität und Neuerungen vor Veröffentlichung getestet wurden, bietet sich folgendes Testkonzept an:

- Unit Tests
    - Evaluation der bisherigen Testabdeckung durch Unit Tests (JUnit) z. B. mittels GitLab Tools (wobei momentan eine ausreichende Unit-Test-Abdeckung durch die involvierten Entwicker nicht gewährleistbar scheint)
- Integrations-/End-to-End Tests
    - Use Cases
        - Sammeln der bestehenden Nutzerszenarien (Starten der Anwendung, Praxen anlegen etc.)
            - auch der negativen Varianten (was soll passieren, wenn etwas schief läuft, was soll geloggt werden, Fehlerausgaben für Nutzer)
        - Beschreibung neuer Features als Nutzerszenarien, wenn neue Entwicklungen anstehen
        - Konkrete Ausformulierung der Nutzerszenarien
            - Ausgangsbedingung (leeres System, vorausgesetzte Konfiguration und Daten)
            - Schritte für Durchlauf des Szenarios
            - Erwartung nach Durchlauf des Szenarios
        - Automatisierte Einbindung der Tests in GitLab CI/CD Pipeline
            - Erstellen eines Testsystems bei jedem größeren Build (neues Feature oder für neue Version)
            - automatisierte Testinfrastrukturen können in den GitLab Deployment-Ablauf eingebunden werden (und somit bei Merges und Releases als Qualitätsschwelle eingesetzt werden)
            - anhand der Nutzerszenarien können mit dem Keyword-basierten Testframework Robotframework und seiner Bibliothek für Tests von GUI-basierten Java-Anwendungen automatisierte menschenlesbare End-to-End Tests erstellt werden.