Semesterprojektentwurf

OG Retro Racer

Autoren: Timo Schrage, Jan Schröder, Pascal Rene Daniel, Philip Kiesler-Großpietsch, Pascal Schwartz, Simon Gartzke, Oliver Elias Adler

---

Anforderungen

Für das Semesterprojekt werden wir alle Anforderungen und Erweiterungen über Tickets in Github abbilden. Das ermöglicht es uns die Priorität der verschiedenen Anforderungen zu bestimmen. Außerdem bietet und das integrierte KanbanBoard eine gute Übersicht über die anstehenden Aufgaben und den aktuellen Projektfortschritt.

Das Softwareprojekt basiert auf einer Client-Server-Architektur. Dies bedeutet, dass ein zentraler Server mit verschiedenen Clients kommuniziert und Daten austauscht. Die Client-Serverkommunikation wird über Socket.io hergestellt.

Rollenverteilung und Verantwortlichkeiten

Bei der Rollenverteilung haben wir uns vorerst auf folgendes geeinigt:

Philip ist verantwortlich für das Design. Alle neuen Erweiterungen und Anforderungen sollen mit Ihm abgesprochen werden. Zusätzlich wird er auch an der Implementierung des Pyhtonclients arbeiten. Oliver ist ebenfalls für den Pythonclient mit verantwortlich. Währenddessen werden sich Timo, Simon, Pascal und Jan eher auf den Javaclient konzentrieren. Zum Schluss wird Pascal Schwartz die Hauptverantwortung für den Python Server übernehmen. Wir werden weitere Frameworks, ganz besonders für die GUI mit das Projekt einbinden. Restliche Rollen beziehungsweise Verantwortlichkeiten haben wir bisher nicht definiert. Diese sollen sich im Laufe des Projektes entwickeln und festigen.

Risiken

Während des Projektes gibt es verschiedene Risiken, die wir berücksichtigen müssen. Ganz besonders der sogenannte "Bottle-Neck", der durch eine schlechte Aufgabenverteilung entstehen kann. Wenn die Aufgaben nicht gleichmäßig auf die Teammitglieder verteilt werden, können einige von Ihnen überlastet werden. Das kann zu Verzögerungen im Projekt führen, da die betroffenen Mitglieder nicht in der

Lage sind, ihre Aufgaben rechtzeitig abzuschließen. Hinzu kommen unvorhergesehene Ausfälle, sei es durch Krankheit oder andere unerwartete Abwesenheiten. Diese können den Fortschritt des Projekts stark beeinträchtigen. Auch die Kommunikation ist nicht zu unterschätzen und spielt einen wichtigen Faktor. Mangelnde Kommunikation kann nämlich zu Missverständnissen und Fehlern führen, die das Projekt ins Stocken bringen können.

Meetings und Kommunikation

Wir planen, regelmäßige Meetings durchzuführen. Hierzu gehört das Replenishing Meeting, in dem Feedback gegeben wird, der aktuelle Status besprochen und neue Aufgaben verteilt werden. Darüber hinaus sind spontane Zwischenmeetings geplant, die je nach Bedarf einberufen werden können, um eventuelle Risiken entgegenwirken zu können. Wir haben uns darauf geeinigt uns mindestens einmal pro Woche zu treffen. Diese Treffen werden über „Discord“ abgehalten. Sollten jedoch Probleme auftauchen oder ein erhöhter Abstimmungsbedarf bestehen, kann die Frequenz der Meetings erhöht werden. In Bezug auf die Priorisierung der Funktionalitäten haben wir uns darauf geeinigt, zuerst den MVP auszuliefern. Dies stellt sicher, dass wir die grundlegendsten und wesentlichen Funktionen des Spiels so früh wie möglich bereitstellen. Um den Fortschritt unseres Projekts kontinuierlich im Auge zu behalten und zu steuern, setzen wir auf verschiedene Tools und Methoden wie Kanbantickets, Github und regelmäßige Meetings.

Meilensteine und Fortschritt

Im Rahmen unseres Projekts haben wir bestimmte Meilensteine festgelegt, die es zu beachten gilt. Einer unserer Hauptziele in den Anfangsphasen ist die Erstellung von MVPs (Minimum Viable Products). Dies bedeutet, dass unser Server in der Lage sein sollte, erste Anfragen zu bearbeiten und wir die ersten GUI-Elemente fertiggestellt haben sollten.

Was die endgültige Auslieferung betrifft, so liegt unser Fokus darauf, einen stabil laufenden Server sowie funktionsfähige Clients bereitzustellen. Ein weiteres wichtiges Ziel für uns ist es, eine schöne (GUI) zu entwickeln.

Da unser Projekt einen laufenden Server benötigt, ist die Hardware in Form eines eigenen Rootservers erforderlich.

Tools und Werkzeuge

Schließlich legen wir großen Wert auf die Dokumentation. Der gesamte Quellcode sollte ordnungsgemäß kommentiert werden, eventuell unter Verwendung von Javadoc. Eine Anleitung für das Spiel wird erstellt, um Benutzern den Einstieg zu erleichtern. Zudem werden wir die Funktionalitäten des Spiels ausführlich erklären und auch Probleme oder Misserfolge im Laufe des Projekts dokumentieren.

Für das Projekt haben wir uns für die Programmiersprachen Python und Java entschieden. Der Austausch und die Versionierung erfolgen hauptsächlich über GitHub.

Unsere Projekttagebücher, Meeting Protokolle und die gesamte Projektdokumentation sind für alle Teammitglieder zugänglich und werden im zentralen Repository unter:

```
https://github.com/Only-OGs
```

gespeichert.

Um unser Repository sauber zu halten und um Schwierigkeiten zu vermeiden, werden wir den GitFlow-Ansatz anwenden. Dadurch gibt es klare Richtlinien für Feature-Entwicklungen, Hotfixes und Releases und das Masterbranch bleibt funktionsfähig. Wenn Jemand etwas Änderungen pushen möchte, werden diese in Form von Pull Requests zur Überprüfung eingereicht. Bei diesen Code-Reviews wird nicht nur überprüft, ob der Code-Standard (siehe unten) eingehalten wurde, sondern auch, ob die Funktionalität korrekt implementiert ist. Grundsätzlich kann jedes Teammitglied ein Code-Review durchführen, jedoch muss jeder Pull Request von mindestens zwei Personen überprüft werden.

Für die Durchführung unserer Tests werden wir JUnit für Java und UnitTest für Python nutzen. Sobald die Software fertiggestellt ist, planen wir, sie auf einem eigenen RootServer zu deployen. Für die Kommunikation zwischen unseren Softwaremodulen setzen wir auf die Bibliothek Socket.io und verwenden das TCP-Protokoll zur Datenübertragung.

Als weitere Tools werden wir PyCharm, IntelliJ und VSCode nutzen. Wir haben uns für diese entschieden, da wir bereits mit diesen vertraut sind und auch regelmäßig mit diesen arbeiten.

Bei der Implementierung werden auf folgende wesentliche Aspekte geachtet, um schlechtes Programmieren zu vermeiden:

- Variablennamen im CamelCase
- Maximal 50 Zeilen pro Methode
- Nicht mehr als 5 verschachtelte If-Anweisungen
- Höchstens 6 Parameter pro Methode
- Maximal 300 Zeilen in einer Klasse
- Generelles Vermeiden von "Bad Smells"

- Dokumentation in deutscher Sprache, während der eigentliche Code in Englisch geschrieben wird.

Im Anschluss folgen auch bereits erste Ansätze und Ideen was die GUI betreffen und auf welches einheitliche Aussehen wir uns geeinigt haben:

In einem sogenannten Style Guide definieren wir die zu verwendenden Farben und Schriftarten: