Projektleitdokument

Dungeon Arena

Betreut durch Prof. Dr. Gerhard Wanner
Hochschule für Technik Stuttgart

2016-03-31

1 Projektdefinition

Jasmin Hellebronth

1.1 Hintergrund des Projektes

Der Bachelor-Studiengang Informatik enthält im Hauptstudium das Modul *Informatik-Projekt 2*. Es hat innerhalb der Bachelor-Prüfung ein Gewicht von 7/110 (SPO2013). Gegenstand des Moduls ist die Realisierung einer umfangreichen Anwendung unter Einhalt eines agilen Vorgehensmodells und unter Einsatz moderner Techniken.

1.2 Gewünschte Ziele und Ergebnisse

Angestrebte Ergebnisse sind

- eine vollständige Dokumentation,
- eine lauffähige Anwendung: ein Multiplayer Game für Apple iPhone und iPad,
- eine Präsentation,
- ein Beitrag für die Hochschulzeitung Stallgeflüster sowie
- ein Webauftritt.

1.3 Projektumfang und Ausschlüsse

Das Projekt umfasst

- die Planung,
- den Entwurf,
- die Programmierung,
- die Validierung und Verifikation sowie
- das Management von Qualität, Konfigurationen und Risiken.

Das Projekt umfasst nicht

- das Marketing,
- Public Relations und Social Media
- den Support oder
- die Außerbetriebnahme.

1.4 Einschränkungen, Randbedingungen und Ausnahmen

Die Randbedingungen sind

- die Entwicklung unter Mac OS X,
- die Anwendung unter iOS sowie
- die Einhaltung der Vorgaben vom Apple Game Center.

1.5 Benutzer und alle anderen bekannten Interessengruppen

Mögliche Nutzer sind alle Nutzer des Apple App-Store.

2 Projektlösungsansatz

Eren Sonal

In Gruppenarbeit haben wir diskutiert welches Spiel wir entwickeln können.
Es gab verschiedene Spieleideen, wobei wir uns für Dungeon Arena entschieden haben.
Es folgten Skizzen und weitere Überlegungen für das Spiel.
Nach weiteren Treffen haben wir besprochen welche Features wir in das Spiel einbauen könnten und die Aufgaben wurden verteilt.
Des Weiteren wurde besprochen welches Framework und welche Programmiersprache wir benutzen werden.

3 Business Case

Eren Sonal

3.1 Managementzusammenfassung

Es wird ein multiplayer 2D-Spiel entwickelt, welches später auf den Markt kommt und Gewinn einbringen soll.

3.2 Gründe

Software-Projekt 2 an der Hochschule für Technik.

3.3 Optionen

- Null-Option: das Projekt wird nicht durchgeführt
- Minimum-Option: ein spielbares Spiel
- Minimum-Plus-Option: spielbares Spiel mit Features

3.4 Erwarteter Nutzen und Nutzentoleranz

- Spielbares Multiplayer-Spiel
- Bestehen von SP2
- (Gewinneinbringend)

3.5 Zeitrahmen

April bis Juni 2016.

3.6 Kosten

Bisherige Kosten:

- Aufsetzen eines Projektmanagement- und Versionsverwaltungssystem
- zwei verschiedene Schätzverfahren

Entstehende Kosten:

- siehe Tabelle (Zeiterfassung)

4 Projektmanagementteamstruktur

Vladyslav Trutniev

Beschreibung der Hierarchie des Projektmanagementteams einschließlich Lenkungsausschuss, Projektmanager, aller Teammanager sowie aller Rollen in Projektsicherung und Projektunterstützung.

- Ansprechpartner auf Seite des Kunden: Prof. Dr. Gerhard Wanner
- Product Owner: Ridvan Esin, Jasmin Hellebronth
- Master: Matthias Wenzel, Nahla Thameur
- Entwicklungsteam: Ridvan Esin, Dennis Geiger, Julian Haspel, Jasmin Hellebronth, Volkan Kutlar, Marcel Mayer, Eren Sonal, Nahla Thameur, Vladyslav Trutniev, Matthias Wenzel

5 Qualitätsmanagementstrategie

Volkan Kutlar

5.1 Einführung inklusive Zweck, Ziele und Umfang

Für das gesamte Projekt wird das Testframework **NUnit** verwendet. Dieses Framework dient dazu nötige Unit Tests für die Software zu entwickeln. Jedes Teammitglied muss dafür, je nach zugeteilter Anforderung, selbst Unit Tests schreiben und diese bei der Implementierung der Features berücksichtigen bzw. anpassen.
Zweck dessen ist es Fehler schon von Entwicklungsbeginn an auszuschließen. Das Ziel ist es dabei Fehler so zu vermeiden, dass sie im späteren Entwicklungsprozess zu keinen Komplikationen führen können. Wie bereits erwähnt muss jedes Mitglied seine eigenen Testklassen schreiben. Dies führt dazu, dass das gesamte System im gesamten Umfang regelmäßig getestet wird.
Weiter werden manuelle Tests von Teammitgliedern durchgeführt, die ein bestimmtes Szenario im Spiel vorgegeben bekommen und dieses auf Fehler testen müssen.

5.2 Qualitätsmanagementverfahren

5.2.1 Qualitätsplanung

Um Qualitätsanforderungen an das Produkt in überprüfbarer Form festzulegen, müssen die einzelnen Anforderungen bzw. Features des Spiels einzeln definiert vorliegen und dem Entwicklungsteam bekannt sein. Jedes Mitglied muss dann, je nach Aufgabenbereich, seine Aufgabe so erledigen, dass diese Anforderungen erfüllt werden. Erachtet das jeweilige Mitglied es für Sinnvoll Unit Tests für die Anforderung zu schreiben, so müssen diese so umgesetzt werden, dass die jeweiligen Anforderungen in den Ergebnissen wiedergespiegelt werden. Jedes Mitglied ist dabei für seine Feature Umsetzung und seine Testvorbereitung selbst verantwortlich.

5.2.2 Qualitätskontrolle

Um die Qualität zu kontrollieren, müssen alle Unit Tests von allen Mitgliedern des Teams regelmäßig durchgeführt werden. Fallen Tests dabei durch, müssen die dafür verantwortlichen Mitglieder diese Tests anpassen bzw. Korrigieren.

Um eine angemessene Testabdeckung der Software zu gewährleisten, wird ein **Code Coverage** Tool verwendet. Hiermit kann geprüft werden, wie weit der geschrieben Code der Entwickler tatsächlich durch Tests abgedeckt ist. Als Code Coverage Tool nutzen wir die eingebaute Software der Entwicklungsumgebung **MonoDevelop**. MonoDevelop wird mit der Engine automatisch mitgeliefert.

5.2.3 Qualitätssicherung

Zur Sicherstellung einer guten Qualität des Produktes muss jedes Mitglied des Entwicklerteams in regelmäßigen, kurzen Zeitabständen alle momentan verfügbaren Unit Tests des Systems ausführen. Um dies zu vereinfachen werden diese Tests in Blöcke zusammengefasst, die eine

gewisse Automatisierung ermöglichen. Die Zusammenfassung der Blöcke erfolgt automatisch durch das Asset **Unity Test Tools**, welches in Unity mithilfe des **AssetStores** importiert werden kann. Fallen Tests durch, müssen die Verantwortlichen benachrichtigt werden. Diese kümmern sich dann selbständig um das Problem und müssen somit ihre Tests entweder anpassen oder umschreiben.

5.3 Tools und Techniken

Um die Qualität des Produkts sicher zu stellen wird bei der Entwicklung das Testframework **NUnit** genutzt. Zur Prüfung einer angemessenen Testabdeckung wird ein **Code Coverage** Tool verwendet. Zusätzlich werden Testszenarien vorgegeben, die beschreiben wie im laufenden Spiel bestimmte Aktionen durchzuführen sind um Funktionalitäten zu testen.

5.4 Dokumentation

Zur Dokumentation ist ein **Qualitätsregister** im Wiki des Projekts vorhanden. Dieser beschreibt welche Unit Tests geschrieben wurden und welche Klassen sie Testen. Jedes Mitglied muss in diesem Qualitätsregister seine Klassen und Unit-Tests selber eintragen. Ebenso muss der Zustand des Testcoverages mit eingetragen werden.

5.5 Berichterstattung

Zu jedem Sprintende wird jedes Mitglied aus dem Entwicklerteam das Ergebnis der laufenden Unit Tests im Qualitätsregister notieren. Dieser Bericht ist dann im Wiki zu finden. Der Bericht enthält Daten wie gelaufene Unit Tests, getestete Klassen, die gesamte Erfolgsquote und die momentane Testabdeckung. Zusätzlich müssen zu jedem fehlgeschlagenem Test die dafür verantwortlichen Mitglieder erwähnt werden, damit dieses sich um die Probleme direkt kümmern können.

5.6 Zeitplanung der Qualitätsmanagementaktivitäten

Für das Unit Test schreiben und deren Testanpassungen wird insgesamt etwa ein Drittel der gesamten Entwicklungsdauer angerechnet. Weiter wird für jede Person eine Zeit von insgesamt 10 Stunden (2 Stunden * 5 Sprints) Arbeitszeit dazu gerechnet.

5.7 Rollen und Verantwortlichkeiten

Jedes Mitglied ist für seinen geschriebenen Code selbst verantwortlich. Das bedeutet, dass jedes Mitglied dafür zu sorgen hat, dass sein Code fehlerfrei ist, logisch Sinn ergibt, je nach Bedarf durch Unit Tests abgedeckt ist und einen angemessenen Code Coverage besitzt.

6 Konfigurationsmanagementstrategie

Matthias Wenzel

Die Konfigurationsmanagementstrategie legt fest, wie, wann und durch wen die Produkte kontrolliert und geschützt werden.

6.1 Einführung

Im Rahmen des Konfigurationsmanagements wird eine große Anzahl projektspezifischer Aufgaben adressiert, welche jeweils durch mindestens einen Aufgabenträger des Projektteams bearbeitet werden sollten. Es werden dabei neben den Rollen bzw. Aufgabenbereichen der Teammitglieder sowohl Kategorien festgelegt, welche den Projektfortschritt gesamt, sowie den Fortschritt der jeweils
aktuellen Aufgabe Repräsentieren, als auch Meilensteine, zu welchen die einzelnen Aufgaben zuzuordnen sind. Weitere wichtige Themen zum Konfigurationsmanagement ist die Haltung und Pflege einer Informationsplattform (Wiki), sowie die Verwaltung der Ressourcen (z.B. Programmcode) und deren Versionsinformationen.
Zur zentralen Konfigurations- und Versionsverwaltung findet in diesem Projekt das kostenlose Tool GitHub Verwendung, welches durch das Browser-Plugin ZenHub um die Funktionalität der Projekt- und Aufgabenverwaltung ergänzt wird. Beide Anwendungen sind nahtlos ineinander integriert.

6.2 Verfahren für das Konfigurationsmanagement

Planung und Identifikation der Aufgaben (Issues):

In der Menüansicht Boards wird die Planung der projektrelevanten Aufgaben vorgenommen. Diese bietet eine Übersicht über initial sechs Kategorien, welchen die einzelnen Issues zugeordnet werden können. Hierbei lassen sich diese durch Drag- und Drop zwischen den einzelnen Kategorien verschieben. Weiter können Aufgabenträger zugeordnet werden, sowie einzelne Issues durch Labels markiert werden, sodass ersichtlich wird, ob, und falls ja, welche Besonderheiten in der Bearbeitung
der einzelnen Aufgaben vorliegen.

Steuerung und Projektstatusverwaltung:

Der Fortschritt des Projektes wird während der Projektlaufzeit in mehrere Sprints eingeteilt. Diese Sprints beinhalten jeweils alle Relevanten Aufgaben, welche im Rahmen des Projektes innerhalb eines im Team festgelegten Zeitintervalls adressiert werden müssen. Sie liegen in GitHub anhand mehrerer Meilensteine vor, welche neben der Aufgabenzuordnung mit einer Fertigstellungszeit(Deadline) einher gehen. Einem Meilenstein können mehrere Issues und damit mehrere verantwortliche Adressaten zugeordnet werden.

Verifikation, Statuskontrolle und Qualitätssicherung

Die für das Projekt erforderlichen Ressourcen, darunter Grafiken, Quellcode und Verzeichnisse, können direkt per Fileupload auf GitHub hochgeladen werden. Dabei existiert ein System zur

Authentifizierung und Qualitätssicherung.
Die Konfiguration des Projektes ist in so genannte Branches eingeteilt. Während die Entwicklung des Kernprojektes und der damit verbundenen Infrastruktur auf der initialen Master-Branch stattfindet, ist das Projekt derart angelegt, dass weitere Punkte, welche unterschiedliche Bereiche adressieren, in unterschiedlichen Branches angelegt werden können. Dies ist ein wichtiger Punkt für die Versionsverwaltung und der Nachvollziehbarkeit sämtlicher Änderungen im Projekt. Branches, deren Verwaltung und Erstellung können im dafür vorgesehenen Auswahlmenü im GitHub-Reiter Code adressiert werden.
Der phasenorientierte Commit in GitHub kann dahingehend konfiguriert werden, dass zur endgültigen Aktualisierung der verwalteten Ressourcen auf GitHub eine Überprüfung einer oder mehreren, weiteren Personen erforderlich ist. Weitere Details dazu im Qualitätsmanagement.

6.3 Prozesse zur Steuerung der offenen Punkte und Anforderungen

Die Erfassung und Steuerung projektspezifischer Prozesse wird anhand von Meilensteinen bzw. Sprints, sowie den damit verbundenen Issues im GitHub-Reiter Boards vorgenommen.
Es werden diesbezüglich regelmäßige Team-Meetings stattfinden, in denen etwaige Änderungen und Anforderungen in diesen Bereichen besprochen werden.
Zur Anforderungsverwaltung können die folgenden Punkte adressiert werden:

Bewertung des Aufwands einer Aufgabe:

Der Aufwand einer Anforderung kann bei GitHub anhand einer Zahl von 1 bis 40 im jeweiligen Issue
geschätzt werden. Der daraus resultierende Grad wird entsprechend der Ansicht im Aufgabenboard
angezeigt und ist für alle Entwickler sichtbar.

Vorschlag von Maßnahmen anhand einer Aufgabe:

Durch das Erstellen und Festlegen eines Labels können Probleme in der Ausführung einer Aufgabe, sowie Besonderheiten dergleichen für das gesamte Team öffentlich gekennzeichnet werden.
Ein solches Label kann in der Editierung bzw. Erstellung einer Aufgabe im rechten Auswahlmenü eingesetzt werden. Darüber hinaus, können in der Board-Übersicht bei Bedarf weitere Kategorien (Spalten) erstellt werden, welche Aufgaben der gewünschten Kategorisierung aufnehmen können.

Umsetzung projektspezifischer Änderungen:

Die Adressierung von Änderungen in den Aufgaben ist öffentlich und kann in GitHub anhand eines Burndown-Charts nachvollzogen werden. Das Burndown-Chart vermittelt dabei Informationen über
den direkten Adressaten der Aufgabe, sowie auch über den Status des aktuellen Sprints und der Übersicht des Projektfortschritts. Der Fortschritt eines Sprints wird dabei im aktuellen Milestone in der Boards-Übersicht festgehalten und laufend aktualisiert.

6.4 Techniken und Methoden

Projektverwaltung und Konfigurationsmanagement:
Die Zentrale Projektverwaltung, sowie das Konfiguratiosmanagement erfolgt durch GitHub.

Für das Kommunikationsmanagement wird die Software Slack verwendet. Diese ist mit der Projektumgebung von ZenHub verknüpft.

Versionsverwaltung:
Dennis Geiger, Matthias Wenzel

Die Wahl für das Versionsverwaltungssystem ist auf Git gefallen. Git ist ein viel verwendetes, kostenloses Versionsverwaltungssystem, welches sich bereits in zahlreichen Projekten bewährt hat
und kontinuierlich weiterentwickelt wird.
Da für die Konfigurationsverwaltung das Tool GitHub bereits genutzt wird und Git als Versionsverwaltung aktiv nutzt, bietet es sich auch daher an Git zu verwenden. Dadurch kann das Projekt direkt in GitHub angezeigt werden, womit das Team einen einfachen Einblick in den aktuellen und ehemaligen Zustand des Projektes erhält.
Durch die Verwendung von Git besitzt außerdem jedes Teammitglied eine Vollständige Kopie des Repositories auf seinem Rechner, wodurch das Arbeiten an dem Projekt ohne Internetanbindung oder vorherige Kommunikation mit dem Team abgewickelt werden kann. Anschließend werden Änderungen an ein Master-Repository übertragen und auftretende Unstimmigkeiten zwischen den Repositories automatisch oder manuell bereinigt werden.
Die Entwicklungsumgebung der Engine Unity bietet ebenfalls Unterstützung zur Nutzung von Git und
erlaubt eine einfache Übertragung der Versionen an das Repository.

6.5 Dokumentation

Komponenten und Funktionen des zu entwickelnden Produktes können im integrierten Wiki festgehalten und dokumentiert werden. Wichtige Informationen bezüglich der Tools GitHub und ZenHub werden nach Bedarf unter dem Register Konfigurationsmanagement erreichbar sein.
Ziel hierbei ist die Dokumentation der wichtigsten Funktionen, sowie das Zusammenspiel der Software für die projektbeteiligten zu erläutern.

6.6 Berichterstattung

Sämtliche offene Punkte, welche zur geplanten Entwicklung des Projektes für kommende Sprints erforderlich sind, oder später als optionale Features Relevanz tragen, werden im Wiki-Register Features aufgeführt. Informationen über den aktuellen Projektfortschritt, sowie über die für den aktuellen- und unmittelbar darauf folgenden Sprint relevanten Aufgaben werden unter dem Register
Projektfortschritt eingetragen. Die Eintragung erfolgt direkt nach Bekanntwerden aller erforderlichen
Aufgaben für den jeweils nächsten Sprint. Diese ist i.d.R. deckungsgleich mit den im Reiter Boards aufgelisteten Issues und ergänzt diese mit optionalen Aufgaben, welche dort entsprechend gekennzeichnet sind.

6.7 Zeitplanung

Die Zeitplanung für Aktivitäten des Konfigurationsmanagements erfolgt durch die Kombination zweier damit verbundener Aspekte:

Zeiterfassung je Aufgabe bzw. Team-Mitglied:

Erfolgt derzeit elektronisch durch die Eintragung in einer Google-Tabelle (Siehe Verweis im GitHub-Wiki)

Zeitplanung des Projektfortschritts:

Erfolgt elektronisch durch die Zuordnung von Aufgaben zu Milestones bzw. Sprints in GitHub. Jeder in GitHub festgelegter Meilenstein repräsentiert ein Sprint im Projekt und kann mit einer Zeitinformation(Deadline) versehen werden.
Änderungen durch einzelne Nutzer innerhalb der jeweiligen Branches und Projektbereichen werden von GitHub automatisch erfasst.

Zeitaufwand für den Prozess des Konfigurationsmanagements:

Konfigurationsmanagement an sich ist ein laufender Prozess, da Änderung des Programmcodes, sowie auch jede Aktualisierung der projektspezifischen Anforderungen protokolliert werden müssen,
ist die Konfigurationsverwaltung durch GitHub und ZenHub unbeschränkt während der gesamten Projektlaufzeit aktiv.

6.8 Rollen und Verantwortlichkeiten

Im Rahmen der Projektzuteilung sind für jedes Teammitglied unterschiedliche Rollen vorgesehen. Dabei wird pro Teammitglied mindestens eine Rolle in der Produktentwicklung eingenommen. Die Rollenverteilung ergibt sich aus der Kombination unseres im Projekt verwendeten, agilen Vorgehensmodells Scrum, kombiniert mit verschiedenen entwicklungsspezifischen Rollen, welche nach Bedarf und Detailgrad der Umsetzung nachträglich entstehen werden.

Die folgenden Rollen sind aktuell vorgesehen:

- Ansprechpartner auf Seite des Kunden: Prof. Dr. Gerhard Wanner
- Product Owner: Ridvan Esin, Jasmin Hellebronth
- Scrum-Master: Matthias Wenzel, Nahla Thameur
- Entwicklungsteam: Ridvan Esin, Dennis Geiger, Julian Haspel, Jasmin Hellebronth, Volkan Kutlar, Marcel Mayer, Eren Sonal, Nahla Thameur, Vladyslav Trutniev, Matthias Wenzel

Je Aufgabenzuordnung werden für die Entwicklung unterschiedlicher Teilkomponenten eine oder mehrere Personen konsultiert. Diese Sub-Teams bestehen wiederrum aus mindestens einem Issue-Verantwortlichen(Aufgabenträger), sowie einem Co-Verantwortlichen(Ansprechpartner). Dieses System sollte sicherstellen, dass keine Missverständnisse oder unnötige Verzögerungen aufgrund von Nacharbeiten entstehen können, indem jeder Aufgabenträger einen direkten Ansprechpartner für etwaige Probleme hat, welcher in der jeweiligen Aufgabe oder in einer damit verbundenen, übergeordneten Aufgabe Kenntnis besitzt.

In GitHub werden Aufgabenträger für ein beliebiges Issue in der jeweiligen Detailansicht unter Assignee festgelegt. Der Aufgabenträger, sowie der Co-Verantwortliche werden zusätzlich in der Beschreibung eines Issues(erste Zeile) für alle Teammitglieder sichtbar,
gelistet.

Des Weiteren sind zur primären Entwicklung des Hauptprojektes die folgenden Rollen bekannt, deren Verantwortlichkeiten nach Aufgaben spezifiziert während der jeweiligen Sprints festgelegt werden:

- Programmentwickler
- Produkt- und Grafikdesigner
- Datenbankadministrator

6.9 Bewertungsskala für Priorität

Die Priorisierung von Aufgaben kann durch die Verwendung von Labels je Issue in GitHub und der dafür vorgesehenen Prioritäten visualisiert werden. Diese ist in der Detailansicht je
Issue zu finden. Es ist ebenso möglich, benutzerdefinierte Labels für eine Bewertung festzulegen. Des weiteren können Aufgaben im Rahmen der Zeitverwaltung im Zeiterfassungsblatt (Google-Tabelle) priorisiert werden. Es existieren derzeit die Bewertungskategorien “verpflichtend”
und “optional”.

6.10 Bewertungsskala für den Schweregrad

Der Schweregrad einer Aufgabe wird anhand deren Aufwandsschätzung ermittelt. Die Einteilung ist
in der Detailansicht der jeweiligen Aufgabe im Git-Reiter Boards unter der Bezeichnung Estimation zu
entnehmen. Dort kann anhand einer Punkteskala die Komplexität eines Issues geschätzt und festgelegt werden.

7 Risiko-Managementstrategie

Marcel Mayer

7.1 Einführung

Zur Risiko-Managementstrategie gehören die Identifikation der Risiken, deren Bewertung hinsichtlich der Eintrittswahrscheinlichkeit und der Auswirkungen sowie die Analyse von Verfahren zur Risikovermeidung und Risikobehandlung.

Ziel der Risiko-Managementstrategie ist das frühzeitige Identifizieren der Risiken und die daraus folgende Minimierung der Eintrittswahrscheinlichkeit und des zu erwarteten Schadens. Somit leistet die Risiko-Managementstrategie einen wichtigen Beitrag zu einem erfolgreichen Projektergebnis.

Der Umfang und die Durchführung der Aktivitäten wird in den folgenden Abschnitten erläutert.

7.2 Risiko-Managementverfahren

Zunächst werden mögliche Risiken identifiziert. Anschließend werden Verfahren zur Risikovermeidung und Risikobehandlung analysiert und es wird eine Bewertung hinsichtlich der Eintrittswahrscheinlichkeit und der Auswirkungen durchgeführt. Die Gesamtbewertung ergibt sich aus dem Produkt aus Eintrittswahrscheinlichkeit und Auswirkungen. Die Risiken werden absteigend nach ihrer Bewertung sortiert und in einer Tabelle dokumentiert (siehe 8.10).

Die Planung des Risikomanagements und das Berichtswesen werden in Abschnitt 8.5 erläutert.

7.3 Tools und Techniken

Die analysierten Risiken werden in einer Tabelle im Risikoregister erfasst (siehe 8.10).

7.4 Dokumentation

Die analysierten Risiken werden in einer Tabelle im Risikoregister erfasst (siehe 8.10). Die Tabelle wird in GitHub abgelegt um etwaige Anpassungen zur Projektlaufzeit zentral vornehmen zu können.
Die Tabelle hat folgenden Aufbau:

- Risikonummer
- Risikokategorie
- Risikobeschreibung
- Risikovermeidungsstrategie
- Risikobehandlungsstrategie
- Eintrittswahrscheinlichkeit (Skala 1-3)
- Auswirkungen (Skala 1-3)
- Bewertung (Skala 1-9)

7.5 Planung und Berichterstattung

Das Projektteam bespricht mindestens einmal pro Sprint den Stand der Risikobewertung und nimmt wenn nötig Änderungen vor. Somit können neue Risiken frühzeitig identifiziert werden und es ist möglich Risiken zu vermeiden oder beim Eintritt eines Risikos die Auswirkungen so gering wie möglich zu halten. Bei Eintritt eines Risikos bespricht das Projektteam untereinander und ggf. anschließend mit dem Auftraggeber das weitere Vorgehen.

7.6 Zeitplanung der Risiko-Managementaktivitäten

Die initiale Risikoanalyse und -bewertung wird vor dem Beginn des Projektes vorgenommen (im Vorprojekt). Das Projektteam bespricht mindestens einmal pro Sprint den Stand der Risikobewertung und nimmt wenn nötig Änderungen vor (siehe 8.5 Berichterstattung).

7.7 Bewertungsskalen für Wahrscheinlichkeit und Auswirkungen

Die Risiken werden nach Eintrittswahrscheinlichkeit und Auswirkungen bewertet. Es wird eine Skala von 1 bis 3 mit folgender Bedeutung verwendet:
Eintrittswahrscheinlichkeit:
1 = Es ist unwahrscheinlich, dass das Risiko eintritt
2 = Es ist möglich, dass das Risiko eintritt
3 = Es ist sehr wahrscheinlich, dass das Risiko eintritt
Auswirkungen:
1 = Die Auswirkungen bleiben gering
2 = Es ist mit mittleren Auswirkungen zu rechnen
3 = Das Eintreten des Risikos wird große Auswirkungen auf das Projekt haben
Die Bewertung eines Risikos ergibt sich aus dem Produkt aus Eintrittswahrscheinlichkeit und Auswirkungen.

7.8 Risikokategorien

Es werden folgende Risikokategorien verwendet:

- Technisch: Zu dieser Kategorie gehören alle Risiken, die aufgrund der eingesetzten Technik und der verwendeten Technologien auftreten (z.B. fehlende Features eines Frameworks)
- Personal: Zu dieser Kategorie gehören alle Risiken, die durch die Teammitglieder verursacht werden (z.B. Ausfall eines Teammitglieds)
- Kunde: Zu dieser Kategorie gehören alle Risiken, die vom Kunden/Auftraggeber ausgehen (z.B. mangelnde Kooperation des Auftraggebers mit dem Projektteam)
- Anforderungen: Zu dieser Kategorie gehören alle Risiken, die durch fehlerhafte oder unvollständige Anforderungen verursacht werden
- Zeitplan: Zu dieser Kategorie gehören alle Risiken, die durch fehlerhafte Zeitplanung entstehen

7.9 Risikotoleranzen

Auftretende Risiken werden zunächst innerhalb des Projektteams mit dem Product Owner besprochen. Der Product Owner entscheidet dann ob eine Eskalation an den Auftraggeber notwendig ist. Dies ist insbesondere bei Eintritt eines Risikos, das große Auswirkungen auf das Projektergebnis hat, notwendig.

7.10 Risikoregister

Zweck des Risikoregisters ist die Erfassung und Pflege aller Informationen bezüglich der Risiken eines Projekts.

Der Aufbau des Risikoregisters ist in Abschnitt 8.4 erläutert.

Nr	Kategorie	Beschreibung	Risikovermeidung	Risikobehandlung	Eintritts-W.	Auswirkung	Bewertung
1	Technisch	Unerfahrenheit in Technologien	Seminare, gegenseitiger Austausch	In Technologien einarbeiten, Seminare	3	3	9
2	Anforderungen	Spielkonzept fehlerhaft	Spielkonzept auf Papier austesten	Spielkonzept anpassen, Workaround	2	3	6
3	Technisch	Inkompatibilität Software/Technologien	Kompatibilität vor Verwendung prüfen	Alternative Software, Ausweichstrategie	2	3	6
4	Zeitplan	Terminüberschreitung	Das Wichtigste zuerst entwickeln, Puffer einplanen	Features weglassen	2	3	6
5	Technisch	Datenverlust	Daten redundant speichern, Backups	Wenn möglich Backups zurückspielen	2	3	6
6	Technisch	Technologie besitzt erwartete Features nicht	Prototyp	Andere Technologie verwenden	2	3	6
7	Kunde	Änderung der Anforderungen	Agile Vorgehensweise	Änderung der Planung	3	2	6
8	Personal	Teammitglied fällt kurzfristig aus	Wissen teilen	Aufgaben neu verteilen	3	2	6
9	Anforderungen	Aufgabenstellung ist widersprüchlich	Aufgabenstellung eindeutig formulieren	Aufgabenstellung mit Auftraggeber präzisieren	2	2	4
10	Personal	Team arbeitet aneinander vorbei	Regelmäßige Meetings	Regelmäßige Meetings	2	2	4
11	Zeitplan	Zu optimistische Planung	Agile Vorgehensweise, Puffer einplanen	Planung der Realität anpassen	2	2	4
12	Anforderungen	Design nicht umsetzbar	Prototyp	Design anpassen	1	3	3
13	Kunde	Auftraggeber kooperiert nicht	Auftraggeber in Entwicklung einbeziehen	Kooperation einfordern	1	3	3
14	Technisch	Apple lehnt App ab	Kriterien kennen und einhalten	App anpassen	1	3	3

			https://developer.apple.com/app-store/review/rejections/				
15	Personal	Teammitglied fällt komplett aus	Wissen teilen, keine *Gurus*, Dokumentation	Aufgaben neu verteilen	1	3	3
16	Personal	Kompetenzkonflikte im Team	Aufgaben klar abgrenzen	Kommunikation der beteiligten Teammitglieder	3	1	3
17	Technisch	Schlechte Qualität	Testgetriebene Entwicklung, Reviews	Reviews, Tests, Fehler beheben	1	2	2
18	Zeitplan	Zu spätes Testen	Testgetriebene Entwicklung	Zeitplan anpassen, ggf. Features weglassen	1	2	2
19	Kunde	Zu hohe Erwartungen	Umsetzbarkeit prüfen	Anforderungen ggf. ablehnen	2	1	2
20	Personal	Wissen wird nicht geteilt	Wissen teilen, z.B. Dokumentation in Wiki	Seminare	2	1	2

8 Kommunikationsmanagementstrategie

Julian Haspel

8.1 Einführung

Das Team trifft sich an 2 Terminen, regelmäßig jede Woche, außer an Feiertagen.

Montags 13:00-17:15 Uhr, mit Professor Wanner, Raum 2/411

Donnerstags 13:00 - 14:00 Uhr, nur Studenten, Raum 2/411

Als Kommunikationsmittel der Studenten wird Slack und E-Mail benutzt.

8.2 Kommunikationsverfahren

Das Team kommuniziert via E-Mail und Slack, ein webbasierter Instant Messaging Dienst. (Einzel-Gruppenchats möglich, gemeinsame Dokumentenbearbeitung & Integration von Github)

8.3 Regeln der Dokumentation

Das Team erfasst die Zeiten in google docs.

Ein Teammitglied führt Protokoll während den Teammeetings, das Protokoll wird bei Github hochgeladen.

Mögliche Probleme als Issues bei Github eintragen.

8.4 Regeln der Berichterstattung

Wöchentliche Treffen

8.5 Rollen und Verantwortlichkeiten

Alle Teammitglieder sind zur Dokumentation ihrer Arbeit & Zeiterfassung verpflichtet.

Jasmin führt Protokoll

8.6 Stakeholderanalyse

8.6.1 Identifikation der Stakeholder

Studenten, Professor, Apple, Konkurrenz(?)

8.6.2 Stakeholderanalyse

Alle Studenten, sowie der Professor wollen das Projekt erfolgreich beenden.

Apple prüft (Funktionalität der) Anwendung und prüft auf Plagiate, hat kein Interesse an erfolgreichem Bestehen des Projekts**.**

Es gibt keine bekannte direkte Konkurrenz (nur ein Team aus Studenten)

8.7 Informationsbedarf aller beteiligten Parteien

Informationen vom Projekt, Informationen für das Projekt, Informationsquellen, Informationsempfänger, Kommunikationsintervalle, Kommunikationswege, Kommunikationsformate.

9 Projektplan

Ridvan Esin

9.1 Planungsbeschreibung

Das Projekt beginnt am 1. April und endet am 6. Juni. Sie werden in 5 Sprints mit jeweils 2 Wochen unterteilt.
Der erste Sprint dient primär in die Einarbeitung in die Technologien und Software, die mit dem Spiel in Verbindung stehen. Darunter zählt zum Einen in die Game Engine Unity und zum Anderen in die Unterstützungssoftwares wie Git, GitHub, ZenHub und NUnit.
Als Ergebnis des Sprints entsteht die Version mit einem einzigen Spieler, der sich in einer Map Area bewegen lässt.
Der zweite Sprint konzentriert sich auf die Multiplayer Unterstützung sowie Umsetzung einzelner wichtiger Features wie Übergänge zwischen einzelnen Map Areas und Angreifen und Zerstören von Elementen.
Resultat dieses Sprints ist das Grundgerüst des Spiels, welche in fortlaufenden Sprints um weitere Features erweitert wird.
Die Funktionalitäten werden in verschiedene Kategorien eingeteilt. Aufgrund dieser Granulierung lässt sich eine Priorisierung der Features einführen.

9.2 Planungsvoraussetzungen

Wichtigster Punkt damit die Planung wie beschrieben funktionieren kann, ist die endgültige Festlegung der grundlegenden Spielidee. Es kann erst begonnen werden, wenn das Spiel von allen Beteiligten abgesegnet wurde.

9.3 Planungsannahmen

Annahme der Planung ist die Verfügbarkeit des Zeitraums vom 01.04 zum 06.06. Fallen Teammitglieder während diesem Intervall aus kann dies fatale Folgen für den Plan haben. Folglich wird die Teilnahme aller Mitglieder bis zum Schluss angenommen.

9.4 Überwachung und Steuerung

Bearbeitete Features und Issues werden in den wöchentlichen Meetings besprochen. Zudem können dort Probleme, die während der Bearbeitung aufgetreten sind, noch einmal angesprochen werden.
Es gibt keinen Überwacher in dem Sinne. Jeder ist für seinen Teil des Spiels verantwortlich. Issues die nicht von der betroffenen Person bearbeitet werden können, müssen abgegeben werden.
Die Steuerung der Issues erfolgt auf Abstimmung. Wenn sich ein Mitglied für ein Feature entscheidet nimmt er sich dieses an. Die Features werden über GitHub und ZenHub verwaltet.
Es können auch mehrere Personen gleichzeitig an einem Issue arbeiten. Diese werden dann in der entsprechenden View vermerkt.

Werden Features nicht nach Plan abgearbeitet, werden sie in folgende Sprints mit aufgenommen. Dafür sind Sprint 4 und 5 als Puffer vorgesehen.

9.5 Toleranzen in Umfang, Zeit und Kosten

Um eine gewisse Toleranz für die Umsetzung des Spiels bereitzustellen, sind die einzelnen Features mit Prioritäten versehen. Jene Funktionalitäten, welche den Kern des Spiels ausmachen sind so früh wie möglich fertigzustellen. Alle anderen gelten als Nice-To-Have.
Es gilt möglichst viele dieser Features zu integrieren. Jedoch ist es für das Spiel im Allgemeinen nicht kritisch, wenn sie fehlen.
Bezüglich der Zeit existiert keinerlei Toleranz. Das Projekt hat ein fixen Starttermin sowie ein fixen Endpunkt. Es kann nicht verlängert werden. Fällt die Zeit gegen Ende knapp, muss der Umfang gekürzt werden.

9.6 Produktbeschreibung inklusive Qualitätstoleranzen

9.7 Zeitplanung

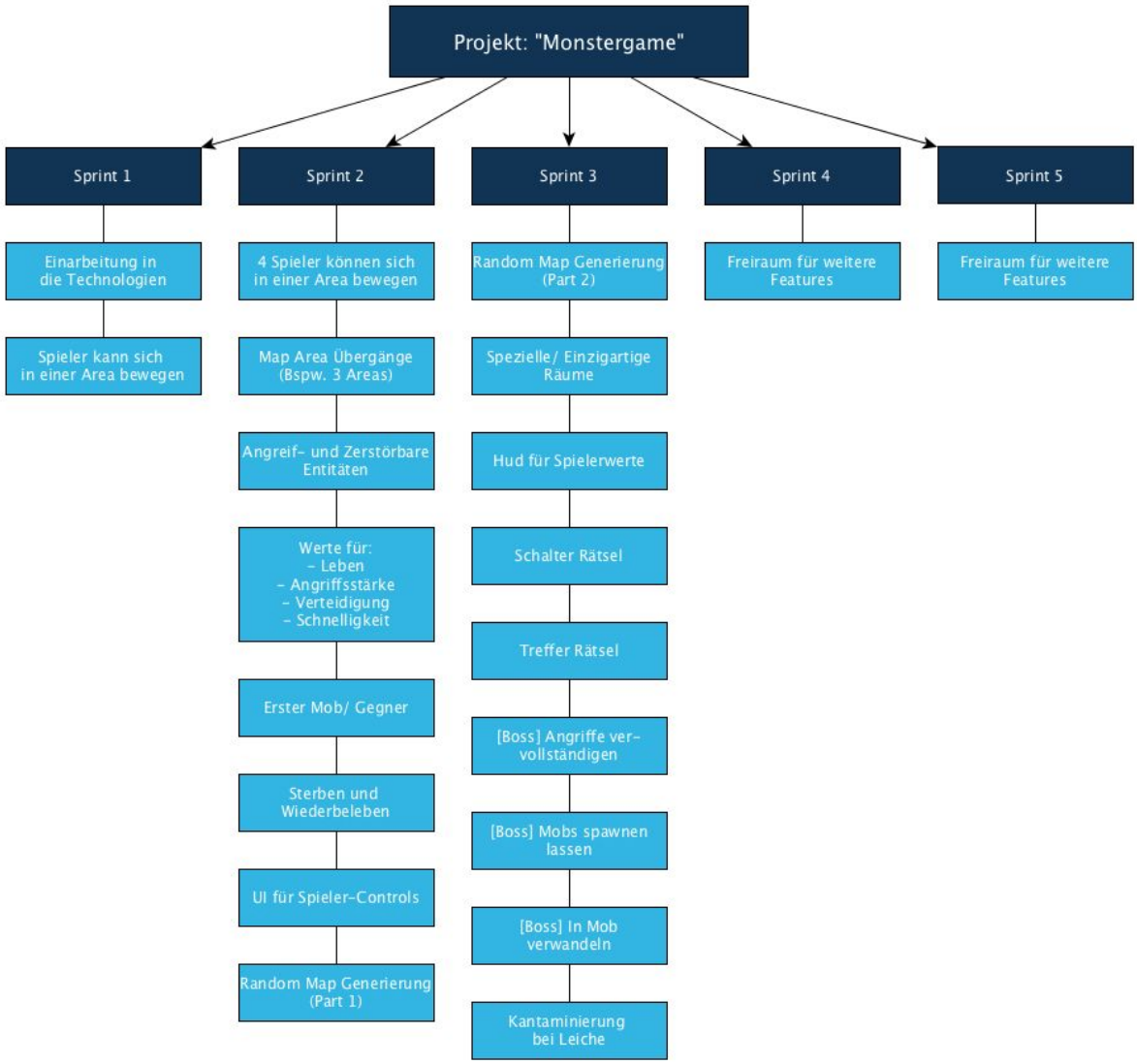
Für die Zeitplanung wurde eine vorläufige Tabelle der Features mitsamt Priosrisierung/ Kategorisierung angefertigt.

Kategorie	**Erläuterung der Kategorien**
Fundamental	Kernfunktionalität des Spiels. Muss so früh wie möglich umgesetzt werden.
Wichtig	Damit das Spiel nach was aussieht.
To be discussed	Muss erst noch mit dem Team besprochen werden.
Sugar	Zur Versüßung

Nr.	**Issue**	**Kategorie**
1	ein Spieler kann sich in einer Map Area bewegen	Fundamental
2	vier Spieler können sich in einer Map Area bewegen	Fundamental
3	drei Areas mit Übergängen	Fundamental
4	angreifbare Entities (Kollisionserkennung, Angreifaktion)	Fundamental
5	ein Non-Player Character Typ	Fundamental
6	Spielerwerte für Leben, Angriffsstärke, Verteidigung, Schnelligkeit (Angriff und Bewegung)	Fundamental
7	Map Generierung, Spielerplatzierung	Fundamental
8	Spielerrollen spezifizieren (Nahkampf, Distanz)	Fundamental
9	Sterben und Wiederbelebung	Fundamental
10	Head-up-Display für Spielerwerte	Wichtig
11	spezielle oder einzigartige Räume	Wichtig
12	weitere Non-Player Character Typen	Wichtig
13	Monster - Moveset vervollständigen (Angriffe)	Wichtig

14	Monster - Non-Player Character spawnen	Wichtig
15	Monster - in Non-Player Character verwandeln mit Cooldown	Wichtig
16	Schalter-Rätsel für Kooperation	Wichtig
17	Treffer-Rätsel für Kooperation (Distanzkämpfer öffnet Tor, Nahkämpfer zerstört etwas)	Wichtig
18	Kontaminierung bei Leiche - o.ä.	Wichtig
19	User Interface für Startmenü - etwa: Play, Practice, Options, Exit	To be discussed
20	Pfeile signalisieren Positionen der anderen Spieler	Wichtig
21	Spielerwert für Leben steigt nach und nach	To be discussed
22	Ausweich-Rolle für Kämpfer	Wichtig
23	Fehlerbehandlungen: Verbindung eines Spielers bricht ab	To be discussed
24	mehrere Monster Typen mit verschiedenen Fähigkeiten	Sugar
25	High-Score listen	Sugar

Überführung in Strukturdiagramm



Überführung in Gantt Diagramm