Anforderungsspezifikation: 2DGameSDK

Aleistar Markóczy

Projektteam: 2D Game Engine
Markóczy Alain Alistair

Version: 1.0
Status: Freigegeben
Datum: 22.03.2019

Berner Fachhochschule
Departement für Technik und Informatik

Versionskontrolle

Version	Datum	Beschreibung	Autor
0.1	06.03.2019	Entwurf	A.M.
1.0	18.03.2019	Erste Version	A.M.
1.1	22.03.2019	Finale Version	A.M.

Inhhaltsverzeichnis

1 Zweck des Dokuments

Dieses Dokument beschreibt die Ziele und Anforderungen für das Projekt 2DGameSDK.

2 Projektvision

Ziel der Arbeit ist es, ein Software Development Kit für unterschiedliche Arten von 2D Computerspielen zu erstellen. Zur Unterstützung soll die Library SFML [1] verwendet werden, die es ermöglicht Multi Plattform fähige Applikationen zu erstellen, wobei die Darstellung von Texturierten Elementen auf dem Bildschirm (mit Zoom, Screen Offset etc.) bereits gelöst ist. Der Fokus liegt daher auf der Implementation einer Gamelogik die individuell angepasst werden kann, so dass unterschiedliche Arten von 2D Games (Shooter, Adventure, Roleplay Game) mit wenig Aufwand erstellt werden können.

Für die Implementation von Animationen, soll im Rahmen dieses Projekts zusätzlich abgeklärt werden, ob allenfalls eine 3D Modellierung besser geeignet ist als eine reine 2D Modellierung. Das gewählte Framework SFML bietet hierzu auch eine geeignete Schnittstelle zu OpenGL [2] an.

3 Projektziele

3.1.1 Anwender / Spiele Entwickler

Im Kontext dieses Projekts, ist der Anwender ein Spieleentwickler, der das SDK verwendet, um ein beliebiges 2D Spiel zu erstellen. Diesem Anwender soll eine Sammlung von Entwicklertools in Form einer C++ Library bereitgestellt werden, wodurch der Entwicklungsaufwand für die Erstellung eines 2D Computerspiels massgeblich verringert werden kann. Die Tools sollen den unterschiedlichsten Ansprüchen von unterschiedlichen Spielkonzepten und -Ideen gerecht werden, um einen möglichst hohen Nutzwert für den Spieleentwickler darzustellen.

3.1.2 Code Qualität und -Wartbarkeit

Der Hersteller des Produkts möchte den Source Code so implementieren, dass das Produkt über einen längeren Zeitraum mit geringem Aufwand gewartet und erweitert werden kann. Um dies zu ermöglichen, sollen die für das Projekt gültigen Coding Standards [3] strikt eingehalten werden. Zusätzlich soll für die Wahl der Softwarearchitektur wann immer möglich auf gängige Design Patterns der Industrie [4] zurückgegriffen werden.

4 Systembeschreibung

4.1 Prozessumfeld

Im folgenden Teil sind die Prozesse beschrieben, die in einem 2D Computerspiel implementiert werden müssen. Diese Prozesse sind der zentrale Bestandteil der Spiel Engine die im Rahmen dieses Projekts entwickelt werden soll:

4.1.1 Spielkontext und -Objekte aufbereiten

Das folgende UML Activity Diagramm zeigt die Schritte, die nötig sind um die Spielobjekte aufzubereiten und den Spielkontext zu erstellen:

Abbildung 1 Activity Diagramm: Laden des Spielkontexts (Grün: Aktivität, Grau: Resultierende Daten)

Zuerst werden die Spielressourcen (Sounds, Texturen, Konfigurationen) geladen, danach werden anhand dieser Daten die Spielobjekte (Spielwelt, Sprites) erstellt. Durch Aufrufen des Game Loops wird das Spiel gestartet.

4.1.2 Game Loop

Zuletzt wird der Game Loop aktiviert, dieser Zyklus definiert das Kernstück der Spiellogik und ist zuständig für die Input Abfrage, die Berechnung des jeweiligen Folgezustands und die Ausgabe des Resultats. Im Folgenden eine detaillierte Darstellung des Game Loops als UML Activity Diagramm.

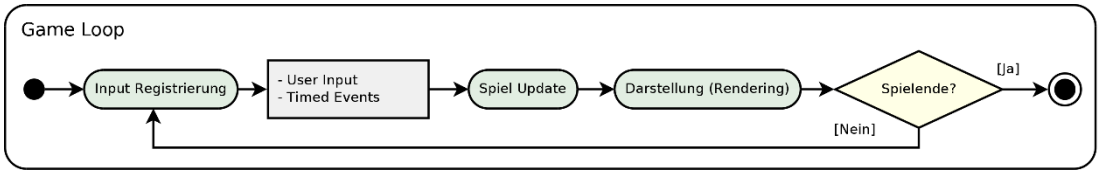


Abbildung 2 Activity Diagramm: Game Loop (Grün: Aktivität, Grau: Daten, Gelb: Entscheidung)

4.2 Systemumfeld

4.2.1 Simple and Fast Multimedia Library SFML

Im Rahmen einer vorgelagerten Machbarkeitsstudie wurde entschieden, dass das Projekt mithilfe der Softwarebibliothek SFML [1] erstellt werden soll, die Bibliothek bietet eine einfache, plattformunabhängige Lösung für folgende Aufgabenbereiche:

- **Maus und Tastatureingabe:** Registrieren der Benutzereingabe.
- **Visuelle Ausgabe 2D:** Applikationsfenster, Darstellung texturierter Elemente, Zoom, Welt-/ Bildschirmkoordinatensystem.
- **Visuelle Ausgabe 3D:** Integrierte Schnittstelle zu OpenGL.
- **Audio Ausgabe:** Buffern und Abspielen von Audio Dateien.
- **Uhr:** Messung der vergangenen Zeit.

4.2.2 UML Kontextdiagramm

Das folgende Kontextdiagramm zeigt die Hauptkomponenten des zu entwickelnden Game SDK's (innerhalb der Systemgrenze), sowie die Verbindung zu Komponenten der externen Bibliothek SFML:

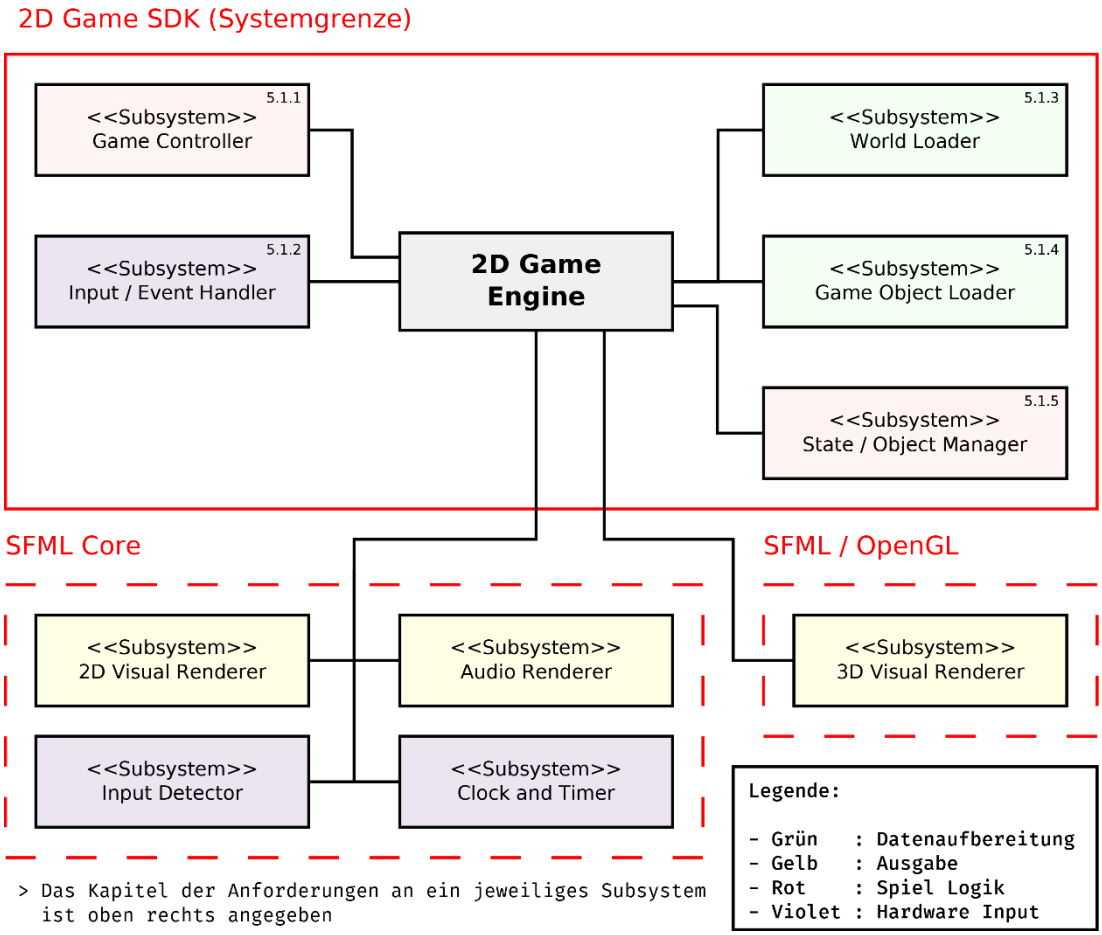


Abbildung 3 UML Kontextdiagramm 2DGameSDK

4.2.3 Beschreibung der Subsysteme

Im folgenden Teil sind die Subsysteme des zu implementierenden Game SDK's und deren Funktionsweise beschrieben:

Game Controller

Das Game Controller Subsystem ist für die getaktete, wiederholte Abarbeitung des Spiel Kontrollflusses (Game Loops) zuständig (Siehe Kapitel 4.1.2). Bei der Initialisierung des Spiels, werden vom Game Controller Subsystem die notwendigen Ressourcenlader (World/Game Object Loader) initialisiert, welche beim Laden der Spielwelt zum Erstellen der Spielelemente verwendet werden. Der Game Loop wird so lange vom Game Controller Subsystem ausgeführt, bis das Spiel von einem anderen Thread aus unterbrochen wird.

Input / Event Handler

Das Input und Eventhandler Subsystem ist dafür zuständig detektierte Spielereingaben (z.B. mit dem Spielcontroller oder der Tastatur) oder andere Arten von Events konkreten Ereignissen und Interaktionen im Spiel zuzuordnen. Die Spielereingaben können mit dem Input Detector System aus der SFML Library [1] erkannt werden.

World Loader

Das World Loader Subsystem ist dafür zuständig die Spielwelt zu laden. Die Spielwelt besteht im Wesentlichen aus einer mehrschichtigen 2-dimensionalen Anordnung von Level Kacheln (Tiles). Die Anordnung wird anhand einer Datei im JSON Format definiert, welche vom World Loader Subsystem gelesen und für den Aufbau der Spielwelt verwendet wird. Jede Kachel in der Spielwelt besitzt eine Textur aus einer entsprechenden Bilddatei, sowie bestimmte Materialeigenschaften die z.B. beschreiben ob die Kachel fest oder durchlässig ist. Die Materialeigenschaften werden in den jeweiligen Materialklassen (im Source Code) definiert, sie werden analog zu den Texturen durch das World Loader Subsystem den entsprechenden Abschnitten in der Spielwelt zugeordnet.

Game Object Loader

Das Game Object Loader Subsystem ist dafür zuständig die Spielobjekte mit den jeweiligen Attributen und Verhaltensregeln ins Spiel zu laden. Die Texturdaten für die Darstellung der Objekte werden aus Bilddateien gelesen und in den entsprechenden Spielobjektklassen referenziert. Die Spielobjektklassen (Erweiterungen der Klasse „GameObject“), die die jeweiligen Verhaltensregeln enthalten, werden bei Bedarf vom Game Object Loader Subsystem erstellt und an das laufende Spiel übergeben.

State / Object Manager

Das State and Object Manager Subsystem ist dafür zuständig jegliche Daten zum aktuellen Spielzustand festzuhalten, so dass diese für andere Subsysteme zu jedem Zeitpunkt verfügbar und aktuell sind. Der Spielzustand ist unter Anderem definiert durch die Spielerposition, die Kameraposition, die Objekte die aktuell im Spiel sind (und deren Eigenschaften) und durch die abgelaufene Spielzeit in Anzahl Frames.

4.3 Prototyp

Im Rahmen des Projekts soll ein Prototyp eines 2D Spiels auf Basis des Game SDK‘s erstellt werden. Dieser Prototyp zeigt, ob die Konzepte, die in dem Game SDK implementiert sind, auch in der Praxis sinnvoll eingesetzt werden können. Der Prototyp soll die folgenden Komponenten enthalten:

- **Steuerbarer Spieler:** Animierter Spieler der sich durch Keyboard Steuerung in der Spielwelt fortbewegen kann.
- **Spielwelt:** Eine Spielwelt bestehend aus soliden und nicht-soliden Materialien.
- **Nicht-Spieler-Charaktere:** Gegner die dem Spieler schaden zufügen können.
- **Abschlussbedingungen:** Konditionen unter welchen das Spiel zu einem erfolgreichen oder nicht erfolgreichen Abschluss gebracht werden.

4.4 Variantenstudium

4.4.1 Animationen

Es gibt grundsätzlich zwei Möglichkeiten wie man Animationen in einem 2D Spiel abbilden kann, zum einen durch eine Abfolge von 2D Bildern (Texturen), zum anderen, durch Bewegung / Rotation eines 3D Objekts und der Projektion dieses Objektes auf einer 2D Fläche. Im Rahmen dieses Projekts sollen die Vorteile und Nachteile beider Varianten eruiert werden, um letztendlich zu entscheiden welche Variante für das Projekt besser geeignet ist. Idealerweise werden zum Schluss beide Varianten von der Spiele Engine unterstützt.

4.4.2 Spielobjekte Datenstruktur

Die Spiele Engine muss eine zentrale Datenstruktur enthalten, mit der alle Spielobjekte schnell und mit geringem Einsatz von CPU / RAM Leistung angesprochen werden können. Im Rahmen dieses zweiten Variantenstudiums soll evaluiert werden, welche Datenstruktur hier am besten geeignet ist. Für 3D Spieleengines wie z.B. Unity [2] wird meist ein Scene Graph eingesetzt, dieser gilt es einfacheren Datenstrukturen wie z.B. Trees, Maps oder Arrays gegenüberzustellen.

4.5 Randbedingungen

4.5.1 C++ Standard

Der im Rahmen dieses Projekts erstellte Source Code wird für den aktuellsten C++ Standard C++17 implementiert und auf die Kompilierung mit der GNU Compiler Collection (GCC) [6] ausgerichtet.

4.5.2 SFML Version

Für dieses Projekt wird die neuste Version der SFML Library (SFML 2.5.1) [1] verwendet.

4.5.3 OpenGL Version

Um auch Kompatibilität mit älteren Grafikkarten zu gewährleisten, wird im Rahmen dieses Projekts die OpenGL [2] Version 3.3 verwendet (Die SFML Library setzt voraus, dass eine OpenGL Version höher als 3.0 verwendet wird).

4.5.4 Zusätzliche externe Ressourcen

Die Abhängigkeit zu externen Code Ressourcen (Libraries) soll minimal gehalten werden. Sofern es unvermeidlich ist Code Ressourcen von Drittanbieter einzusetzen, sollen nur Ressourcen eingesetzt werden, die zum einen mit allen gängigen Betriebssystemen kompatibel sind, und zum anderen keinen restriktiven Copyright Lizenzen unterliegen.

4.6 Qualitätssicherung

4.6.1 Source Code

Der Source Code des Produkts soll den Vorgaben der IsoCPP C++ Core Guidelines [3] gerecht werden. Die Namensgebung der Code Strukturelemente (Funktionen, Klassen, Methoden) soll über das gesamte Projekt konsistent gehalten werden.

4.6.2 Tests

Um die Qualität der Software gewährleisten zu können, sollen für die Zentralen Komponenten und Prozesse des Game SDK‘s Unittests geschrieben werden. Im Rahmen der Unittests sollen die folgenden Punkte abgedeckt werden:

- **Softwareverhalten:** Sicherstellen, dass die Software bei gegebenem Input den erwarteten Output generiert.
- **Speichermanagement:** Sicherstellen, dass keine Objekte unnötig dupliziert werden und dass Objekte nach dem Gebrauch vom Speicher gelöscht werden.

5 Anforderungen

5.1 Funktionale Anforderungen

Die funktionalen Anforderungen werden den einzelnen Subsystemen zugeordnet, sie sind im folgenden Teil aufgelistet:

5.1.1 Game Controller

Tabelle 1 Funktionale Anforderungen: Game Controller Subsystem

ID	Prio	Beschreibung
FG1	Muss	Der Spieleentwickler möchte durch Festlegung der Ausgabeparameter (Fenstergrösse, Framerate, Skalierung) eine **Spielinstanz erstellen** können, sodass nach Start der Spielinstanz der **Game Loop** (Tick-Render Zyklus) **aktiviert** wird.
FG2	Muss	Der Spieleentwickler möchte unterschiedliche **Arten von Spielobjekten** festlegen können, sodass einzelne Instanzen dieser Spielobjekte die **Eigenschaften** und das **Verhalten** der eigenen Art **erben.**

5.1.2 Input / Event Handler

Tabelle 2 Funktionale Anforderungen: Input / Event Handler Subsystem

ID	Prio	Beschreibung
FI1	Muss	Der Spieleentwickler möchte die **Benutzereingaben** mit möglichen **Aktionen im Spiel verknüpfen** können, sodass anhand der Benutzereingaben eine Spiel-/Spielersteuerung realisiert werden kann.
FI2	Muss	Der Spieleentwickler möchte in der Spielinstanz **Events und Callbacks registrieren** können, sodass das **Verhalten** des Spiels gezielt beeinflusst werden kann.

5.1.3 World Loader

Tabelle 3 Funktionale Anforderungen: World Loader Subsystem

ID	Prio	Beschreibung
FW1	Muss	Der Spieleentwickler möchte eine **2D Spielwelt** als **Anordnung von Texturflächen** (Tiles) lesen können, sodass diese Spielwelt im Spiel dargestellt werden kann.
FW2	Muss	Der Spieleentwickler möchte die Möglichkeit haben für Texturflächen der Spielewelt (Tiles) **Materialeigenschaften** zu definieren, sodass diese Eigenschaften das Verhalten mit anderen Spielobjekten festlegen. Dies umfasst bspw. ob das Material **fest** oder **durchlässig** ist oder den **Z-Index** beim Rendering.

5.1.4 Game Object Loader

Tabelle 4 Funktionale Anforderungen: Game Object Loader Subsystem

ID	Prio	Beschreibung
FO1	Muss	Der Spieleentwickler möchte unterschiedliche **Arten von Spielobjekten** festlegen können, sodass einzelne Instanzen dieser Spielobjekte die **Eigenschaften** und das **Verhalten** der eigenen Art **erben.**
FO2	Muss	Der Spieleentwickler möchte **Animationen als Abfolge von Bildern** definieren können, sodass Spielobjekte mit dieser Methode animiert werden können.
FO3	Wunsch	(Variante) Der Spieleentwickler möchte **Animationen als Bewegung eines 3D Objekts** auf einem 2D Sichtfeld definieren können, sodass Spielobjekte mit dieser Methode animiert werden können.

5.1.5 State / Object Manager

Tabelle 5 Funktionale Anforderungen: State / Object Manager Subsystem

ID	Prio	Beschreibung
FS1	Muss	Der Spieleentwickler möchte die Möglichkeit haben einzelne **Objekte** im Spiel **effizient ansprechen** zu können sodass komplexe Interaktionen effizient innerhalb eines Frame Zyklus' realisiert werden können. Es bietet sich an hier auf einen **Scene Graph** oder auf eine **ähnliche Struktur** zurückzugreifen.

5.1.6 Zusatzfunktionen

Tabelle 6 Funktionale Anforderungen: Zusatzfunktionen

ID	Prio	Beschreibung
FA1	Wunsch	Der Spieleentwickler möchte die Möglichkeit haben Elemente eines **In-Game Overlay Displays** zu definieren, sodass globale Spielinformationen (z.B. Abgelaufene Levelzeit, Spielerleben) statisch auf dem Bildschirm dargestellt werden können.
FA2	Wunsch	Der Spieleentwickler möchte ein geeignetes Werkzeug haben, um **Künstliche Intelligenz auf Basis einfacher Regeln** zu realisieren, sodass ich das Verhalten von nicht-Spieler Charakter effizient definieren kann.

5.2 Technische Anforderungen

Tabelle 7 Technische Anforderungen

ID	Prio	Beschreibung
T1	Muss	Das Produkt (Softwareentwicklungspaket) soll auf den **gängigen Plattformen** (Windows, Linux, Mac) **bereitgestellt und ausgeführt** werden können, sodass eine möglichst grosse Audienz mit dem damit erstellten Endprodukt erreicht werden kann.
NT1	Abgr.	Das Produkt muss **nicht** für die Bereitstellung auf **Mobiltelefonen oder Tablets** geeignet sein.
T2	Muss	Die **Prozesse** sollen im sinnvollen Masse **parallel** ablaufen, sodass moderne Multi-Core Prozessoren möglichst gut ausgelastet werden können.
T3	Muss	Die **Frame Zyklen** müssen **präzise getaktet** sein, sodass ein flüssiges Spielerlebnis realisiert werden kann.
T4	Wunsch	Bei zu hoher **Zeitüberschreitung** soll ein **Frame übersprungen** werden sodass die Darstellung mit der Spiellogik wieder synchron ist.

5.3 Anforderungen rendering Qualität

Tabelle 8 Anforderungen rendering Qualität

ID	Prio	Beschreibung
Q1	Muss	Das Rendering soll eine **Double- oder Triple Buffering** Strategie verfolgen sodass die Darstellung nahtlos und flüssig erscheint.
Q2	Muss	Die **Framerate** des Spiels soll **stabil** bei 50Hz sein, wenn auf einem 1680x1050 Pixel grossen Bildschirm **1'000 unterschiedlich texturierte 8x8 Pixel grosse Elemente dargestellt werden** (Wunschkriterium: **10'000** Elemente).
Q3	Muss	Die Framerate des Spiels soll **stabil** bei 50Hz sein, wenn **1'000 kollisionsfähige Spielobjekte gleichzeitig im Spiel** sind (Wunschkriterium **10'000** Spielobjekte).

6 Projektmanagement

6.1 Projektorganisation

6.1.1 Prozessmodell

Für die Umsetzung des Projekts wird mit dem agilen Prozessmodell Scrum gearbeitet, die Sprint Länge beträgt jeweils 2 Wochen.

6.1.2 Meetings

Die Projektmeetings mit dem betreuenden Dozenten finden alle zwei Wochen statt. An diesen Meetings werden der aktuelle Projektstand sowie allfällige Fragen zum Projekt diskutiert.

6.1.3 Meilensteine

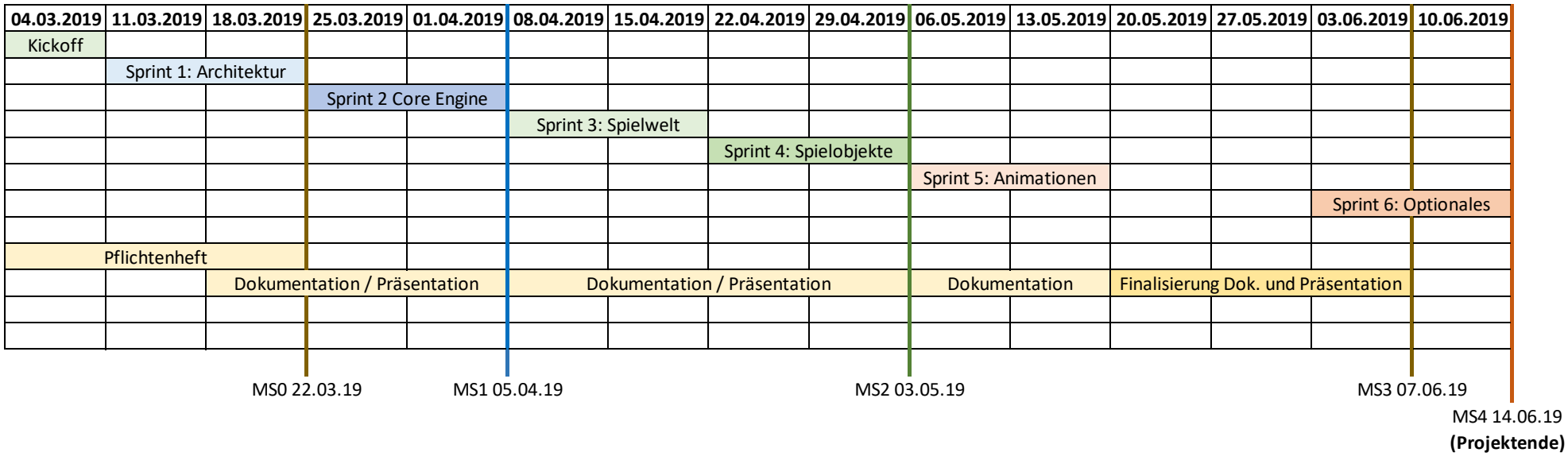
Im Folgenden die definierten Meilensteine des Projekts basierend auf den in diesem Dokument festgehaltenen Zielsetzungen:

Meilenstein	Beschreibung	Datum
MS0	Finale Festlegung der Anforderungen, Pflichtenheft	22.03.2019
MS1	Game Engine: Architektur, Game Loop, Rendering	05.04.2019
MS2	Gameplay Elemente: Spielwelt, Spieler, Gegner	03.05.2019
MS3	Abgabe technischer Bericht	07.06.2019
MS4	Zusatzfunktionalität: Animationen, Optionales **Projektabgabe**	14.06.2019

Die detaillierte Projektplanung orientiert sich an den Zeitvorgaben dieser Meilensteine.

6.2 Zeitplan

Das Projekt wird gemäss Scrum Planung in 6 Sprints mit jeweils einer Dauer von 2 Wochen unterteilt. Kurz vor dem Abgabetermin des technischen Berichts (MS3), werden zwei Wochen für die Finalisierung des technischen Berichts und der Präsentation reserviert, in dieser Zeit werden keine Source Code Anpassungen mehr vorgenommen:



7 Referenzen

[1] L. Gomila, «SFML Home Page,» [Online]. Available: https://www.sfml-dev.org/. [Zugriff am 15 März 2019].

[2] Khronos Group Inc., «OpenGL Home Page,» [Online]. Available: https://www.opengl.org/. [Zugriff am 20 März 2019].

[3] B. Stroustrup und H. Sutter, «Isocpp C++ Core Guidelines,» 7 März 2019. [Online]. Available: http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines. [Zugriff am 22 März 2019].

[4] A. Alexandrescu, Modern C++ Design: Generic Programming and Design Patterns Applied, New Jersey: Addison-Wesley, 2001.

[5] Unity Technologies, «Unity Home Page,» [Online]. Available: https://unity.com/. [Zugriff am 20 März 2019].

[6] Free Software Foundation, Inc., «GCC, the GNU Compiler Collection,» 22 Februar 2019. [Online]. Available: https://gcc.gnu.org/. [Zugriff am 20 März 2019].

8 Glossar

Begriff	Erläuterung
SDK	Ein Software Development Kit ist eine Sammlung von Tools zur Erstellung einer Software in einem bestimmten Aufgabenbereich.
SFML	Simple and Fast Multimedia Library SDML ist eine Open Source C++ Softwarebibliothek die gewisse low-level Tools für die Erstellung von 2D Computerspielen zur Verfügung stellt [1].
Scene Graph	Ein Szenengraph ist eine Datenstruktur, mit der die logische, in vielen Fällen auch die räumliche Anordnung der Objekte in einer darzustellenden zwei- oder dreidimensionalen Szene beschrieben wird.
Tiles (Kachelgrafik)	Als Kachelgrafik (engl. tiles) wird eine Computergrafik bezeichnet, die mosaikartig zusammengesetzt ein vielfach größeres Gesamtbild ergibt. Die Kachel-Technik wird häufig in Computerspielen eingesetzt, da die Kacheln einfacher zu berechnen sind und weniger Arbeitsspeicher verbrauchen.
JSON	Die JavaScript Object Notation, kurz JSON, ist ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen.

9 Abbildungsverzeichnis

10 Tabellenverzeichnis