

Projektarbeit TxtRpg

Liam Gloggner

Inhaltsverzeichnis

Ziele.....	4
Ziel 1.....	4
Ziel 2.....	4
Ziel 3.....	4
Ziel 4.....	4
Risikoanalyse	5
Erkannte Risiken.....	5
Risiko 1	5
Risiko 2	5
Risikomanagement.....	5
Risiko 1	5
Risiko 2:	5
Verwendete Technologien	6
VCS (Version Control System).....	6
Git.....	6
IDE (Integrated Development Environment).....	6
IntelliJ – Community Edition	6
Programmiersprache.....	7
Java 22.....	7
Entwurf Software	7
Draw.io	7
Kommunikation	8
Discord	8
Dateiablage	8
OneDrive	8
UI Entwurf	9
Startbildschirm.....	9
Statistik Abschnitt & Tabs.....	10
Kampf Tab.....	11
Skills Tab	12
Shop Tab.....	13
Programmierung & Grundaufbau	14
Klassendiagramm	14
Character Interface	15
Beschreibung.....	15

Funktionen	15
Player Klasse.....	16
Beschreibung.....	16
Funktionen	16
Enemy Klasse.....	18
Beischreibung.....	18
Funktionen	18
SaveManager Klasse.....	19
Beschreibung.....	19
Funktionen	19
Programmierung	20
Flussdiagramm Programminitialisierung.....	20
Flussdiagramm Kampf.....	21
Test-Bericht	22
Testfall 001: Spielstart und Charakterbenennung.....	22
Situation	22
Soll-Resultat	22
Ist-Resultat	22
Beweis	22
Testfall 002: Spiel speichern.....	23
Situation	23
Soll-Resultat	23
Ist-Resultat	23
Beweis	23
Testfall 003: Kämpfen.....	24
Situation	24
Soll-Resultat	24
Ist-Resultat	24
Beweis	24
Testfall 004: Würfeln	25
Situation	25
Soll-Resultat	25
Ist-Resultat	25
Beweis	25
Abbildungsverzeichnis.....	26
Glossar	27

Ziele

Ziel 1

Zielformulierung:

Bis zum Abschluss meiner Projektarbeit will ich ein Spiel-User-Interface programmieren, welches Formular-Ähnlich ein Text-basiertes Rollenspiel simuliert. Das User-Interface wird einfache Animationen beinhalten.

Durch die Hilfe von Dokumentationen und Tutorials, wie auch eigenes mitgebrachtes Wissen von meiner Programmier-Erfahrung bin ich zuverlässig dieses Ziel zu erreichen.

Ziel 2

Zielformulierung:

Die Spielfigur in meinem Spiel soll bis zum Ende der Projektarbeit Charakterwachstum haben. Sie soll Stufen hochsteigen können, Fähigkeitspunkte sammeln um diese wiederum in Fähigkeitssteigerungen investieren zu können. Diese Fähigkeiten sind: Leben, Trefferchance, Schaden, kritische Trefferchance, kritischer Schaden und Rüstung.

Durch meine Spiel-Erfahrung welche ich in meinem Leben bereits in diversen Spielen gesammelt habe denke ich keine Probleme zu haben diese Mechaniken zu implementieren.

Ziel 3

Zielformulierung:

In meinem Spiel soll sich die Skalierung des Charakters und der Gegner natürlich sein. Die Gegner sollen sich anhand des Spielers skalieren damit das Spiel nicht zu Schwer wird aber auch nicht zu leicht.

Mit Online Ressourcen und meiner Spiele-Erfahrung sollte dieses Ziel keine grossen Probleme bereiten und ich bin zuversichtlich dieses zu erreichen.

Ziel 4

Zielformulierung:

Das Spiel soll Zufallsmechaniken beinhalten. Die Zufallsmechaniken werden in Form von Trefferchance, kritischer Trefferchance sowie einer Stufenaufstiegsmechanik mit Würfeln in dem Spiel vorhanden sein.

Durch die Hilfe von Freunden und Online Artikeln über Java wird dieses Ziel kleine Probleme bereiten, jedoch bin ich zuversichtlich dieses zu erreichen.

Risikoanalyse

Erkannte Risiken

Risiko 1

Ein grosses Risiko, welches ich schon bei der Ideensuche erkannt habe, war die von mir fehlende Erfahrung mit der Programmiersprache Java beziehungsweise mit den Frontend Frameworks Java Swing und Java Awt. Die von mir gesammelte Erfahrung war bisher nur mit Winforms und WPF.

Risiko 2

Die Zeit die ich zur Verfügung stellt für mich ein weiteres Risiko dar. Da ich noch viel dazulernen muss um mit dem Frontend Framework Swing besser klar zu kommen wird das ein wenig Zeit in Anspruch nehmen, da ich nur 60 Stunden zur Verfügung habe.

Risikomanagement

Risiko 1

Da Java gewisse Ähnlichkeiten mit der .NET-Sprache C# hat mache ich mir dazu nicht zu grosse Gedanken. Da die Art zu programmieren einigermaßen dieselbe ist, denke ich es reicht für die fehlende Java Erfahrung eine Art Cheat-Sheet offen zu haben. Bei dem Frontend Framework Swing werde ich grössere Probleme haben aber auch für dieses Problem werde ich mir einen Guide zur Hand nehmen um die fehlende Erfahrung auszugleichen.

Als Java Cheat-Sheet habe den von der Webseite «W3Schools» bereitgestellte Guide verwendet:

<https://www.w3schools.com/java/default.asp>

Als Guide um Swing und Awt ein wenig näher zu kommen habe ich die Webseite «Javatpoint» verwendet:

<https://www.javatpoint.com/java-swing>

Risiko 2:

Als Risikomanagement für meine Zeitknappheit habe ich einen Zeitplan erstellt welchen ich strikt einhalten muss. Dafür habe ich mir in Excel einen Wasserfall Zeitplan erstellt. Einem Agilen Vorgehens Modell habe ich mich bewusst nicht genähert da es als Einzelperson wenig Sinn macht.

Verwendete Technologien

VCS (Version Control System)

Git

Als Version Control System habe ich mich für Git entschieden. Git ist eine Open-Source-Software zur Versionsverwaltung von diversen Dateien. Git wurde von Junio Hamano, Shawn Pence und Linus Torvalds und vielen mehr entwickelt. Ich habe mich für Git entschieden, da ich bereits Erfahrung mit Git aus meinem Berufsleben mitbringen.

Link zu Git: <https://git-scm.com/>



Abbildung 1: Logo von Git

IDE (Integrated Development Environment)

IntelliJ – Community Edition

Als Entwicklungsumgebung für mein Spiel habe ich mich für IntelliJ entschieden. IntelliJ Community Edition wurde von der Firma «JetBrains» entwickelt. Die Erstveröffentlichung des Programms war im Jahr 2001. IntelliJ ist nicht gratis, jedoch mit einer gültigen Schul-E-Mail-Adresse kann es kostenfrei heruntergeladen werden. Ich habe mich für IntelliJ entschieden da einige Freunde von mir, mir das Programm zur Entwicklung mit Java geraten haben.

Link zu IntelliJ: <https://www.jetbrains.com/de-de/idea/>



Abbildung 2: Logo von IntelliJ

Programmiersprache

Java 22

Als Programmiersprache habe ich mich für Java, Version 22, entschieden. Java ist eine Objektorientierte Programmiersprache die von «Sun Microsystems» Jahr 1995 veröffentlicht. Der Grund warum ich mich für Java als Programmiersprache entschieden habe ist: Viele Menschen benutzen Java und es ist eine der Weitverbreitesten Programmiersprachen weltweit. Es kann nicht schaden ein wenig mehr über Java zu wissen.

Link zu Java 22: <https://www.oracle.com/ch-de/java/technologies/downloads/>



Abbildung 3: Logo von Java

Entwurf Software

Draw.io

Als Software für all meine Diagramme und Entwürfe für UI's habe ich mich für «Draw.io» entschieden. «Draw.io» ist eine Webseite, die zum Zeichnen für UI-Entwürfe, Flowcharts, Mindmaps und vieles mehr verwendet werden kann.

«Draw.io» wurde von «JGraph Ltd» entwickelt.

Link zu Draw.io: <https://app.diagrams.net/>



Abbildung 4: Logo von Draw.io

Kommunikation

Discord

Als Kommunikationsmittel habe ich Discord verwendet. Discord ist eine Sprach- und Videochat-Software entwickelt von Discord Inc. Ich habe Discord hier aufgelistet da ich einige Freunde von mir kontaktiert habe um mir bei diversen Problemen zu helfen.

Aufgrund der grösseren Erfahrung, die ich mit Discord habe, habe ich Discord und nicht Microsoft Teams als Kommunikationsmittel gewählt.

Link zu Discord: <https://discord.com/>



Abbildung 5: Logo von Discord

Dateiablage

OneDrive

Als Dateiablage habe ich das von der TEKO zur Verfügung gestellte OneDrive verwendet. OneDrive ist eine Cloud-Ablage für Dateien, die von Microsoft entwickelt wurde.

Mit OneDrive habe ich die Möglichkeit jederzeit von jedem Gerät aus, problemlos an meiner Dokumentation für die Projektarbeit zu schreiben.

Link zu OneDrive: <https://www.microsoft.com/de-ch/microsoft-365/onedrive/online-cloud-storage?market=ch>



Abbildung 6: Logo OneDrive

UI Entwurf

Startbildschirm

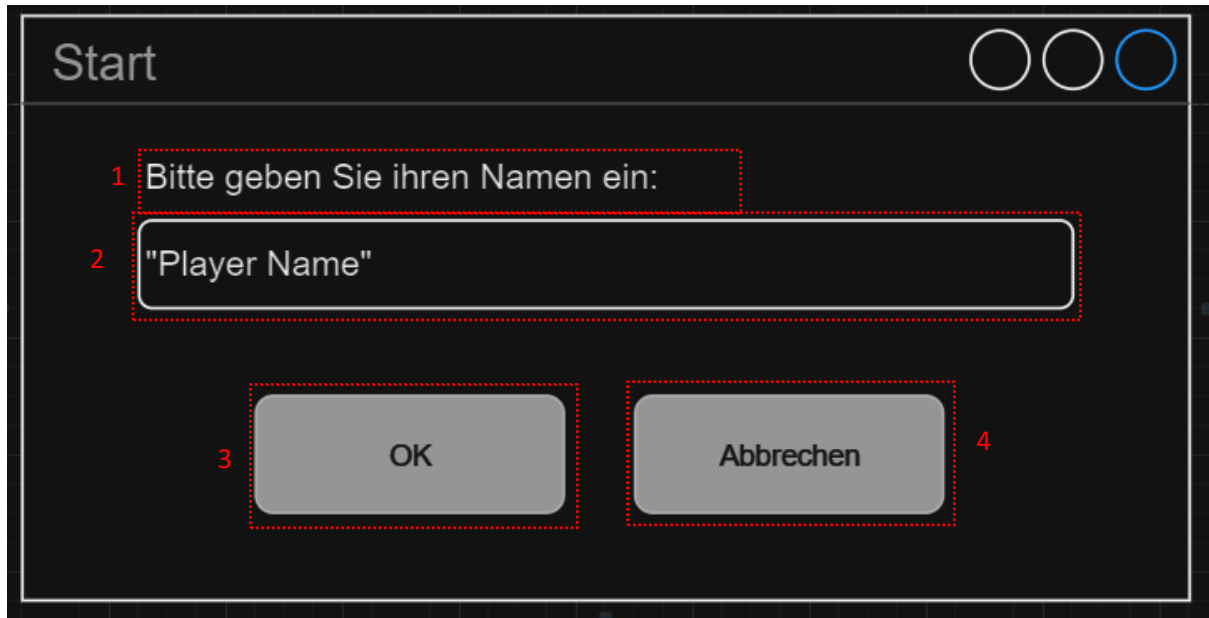


Abbildung 7: UI-Entwurf Startbildschirm:

Funktion	UI-Element
1. Titel Anzeigen	Label
2. Textfeld um Namen einzugeben	Textfeld
3. Startet das Spiel	Knopf
4. Beendet das Spiel	Knopf

Statistik Abschnitt & Tabs

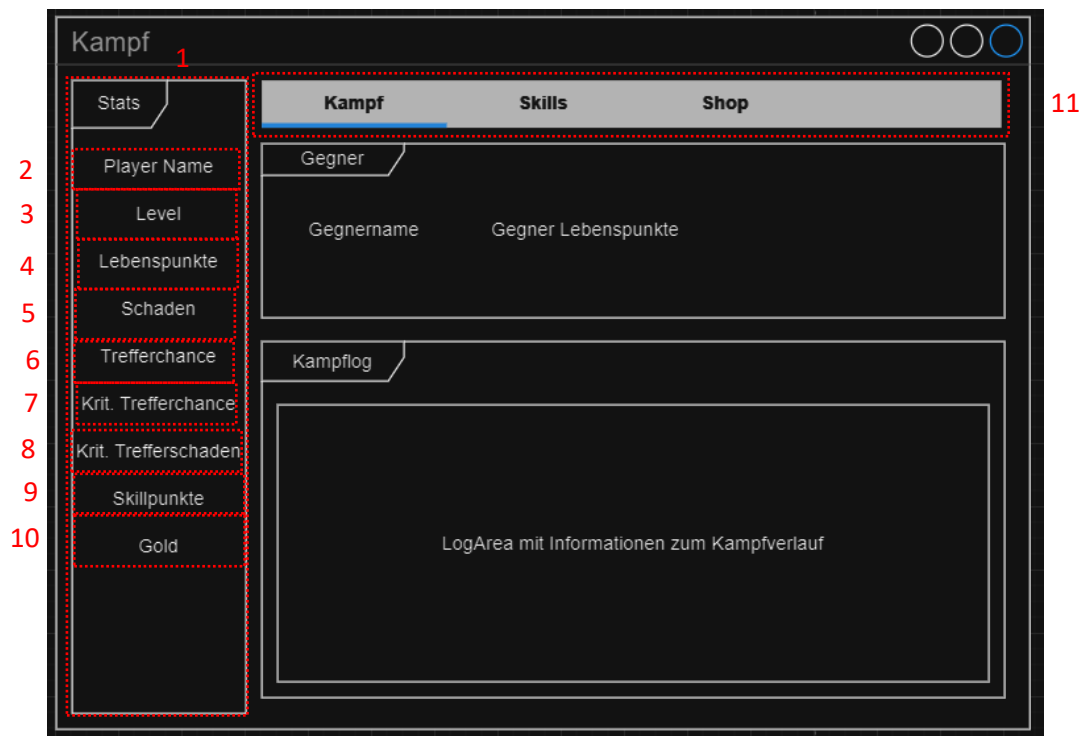


Abbildung 8: UI-Entwurf Statistik Abschnitt

Funktion	UI-Element
1. Container für die Statistiken des Spielers	Frame
2. Zeigt den Namen des Spielers an	Label
3. Zeigt das Level des Spielers an	Label
4. Zeigt die Lebenspunkte des Spielers an	Label
5. Zeigt den Schaden des Spielers an	Label
6. Zeigt die Trefferchance des Spielers an	Label
7. Zeigt die Kritische-Trefferchance des Spielers an	Label
8. Zeigt den Kritischen Trefferschaden des Spielers an	Label
9. Zeigt die Skillpunkte des Spielers an	Label
10. Zeigt das Gold des Spielers an	Label
11. Tabs für die verschiedenen Ansichten	TabView

Kampf Tab

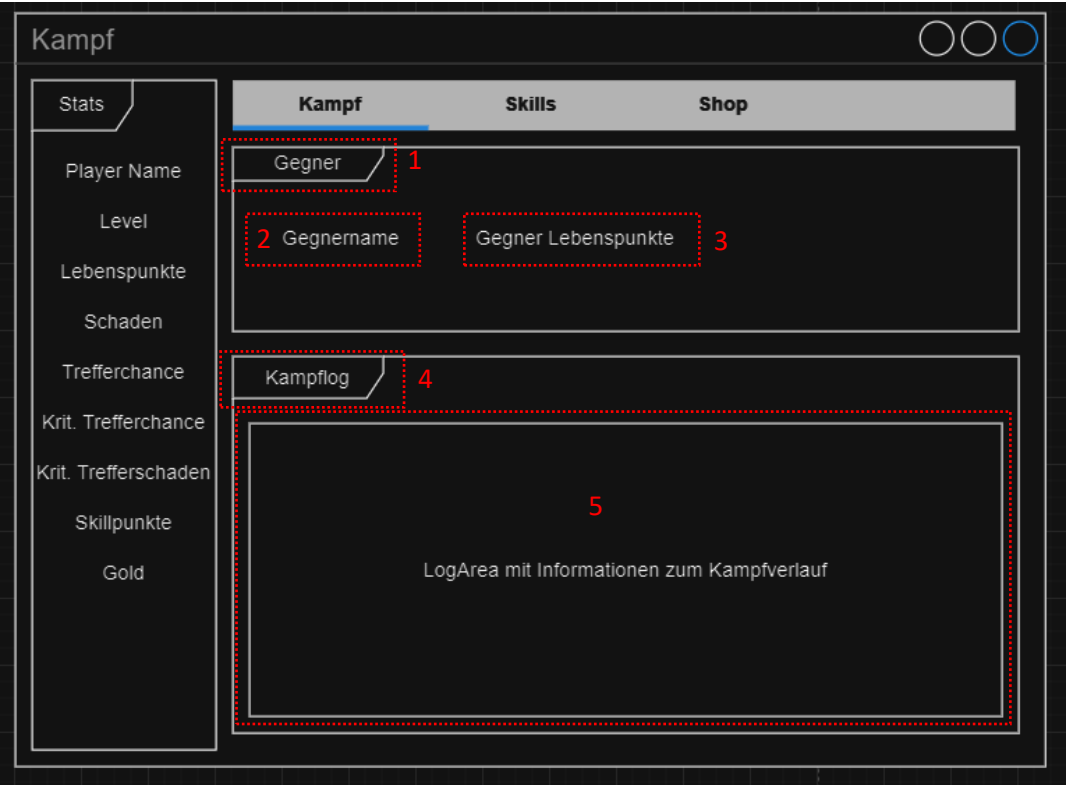


Abbildung 9: UI-Entwurf Kampf Tab

Funktion	UI-Element
1. Container für die Gegnerinformationen	Frame
2. Zeigt den Namen des Gegners an	Label
3. Zeigt die Lebenspunkte des Gegners	Label
4. Container für den Kampf log	Frame
5. Zeigt das Log des Kampfes und erhaltene Ressourcen an	Textfeld

Skills Tab

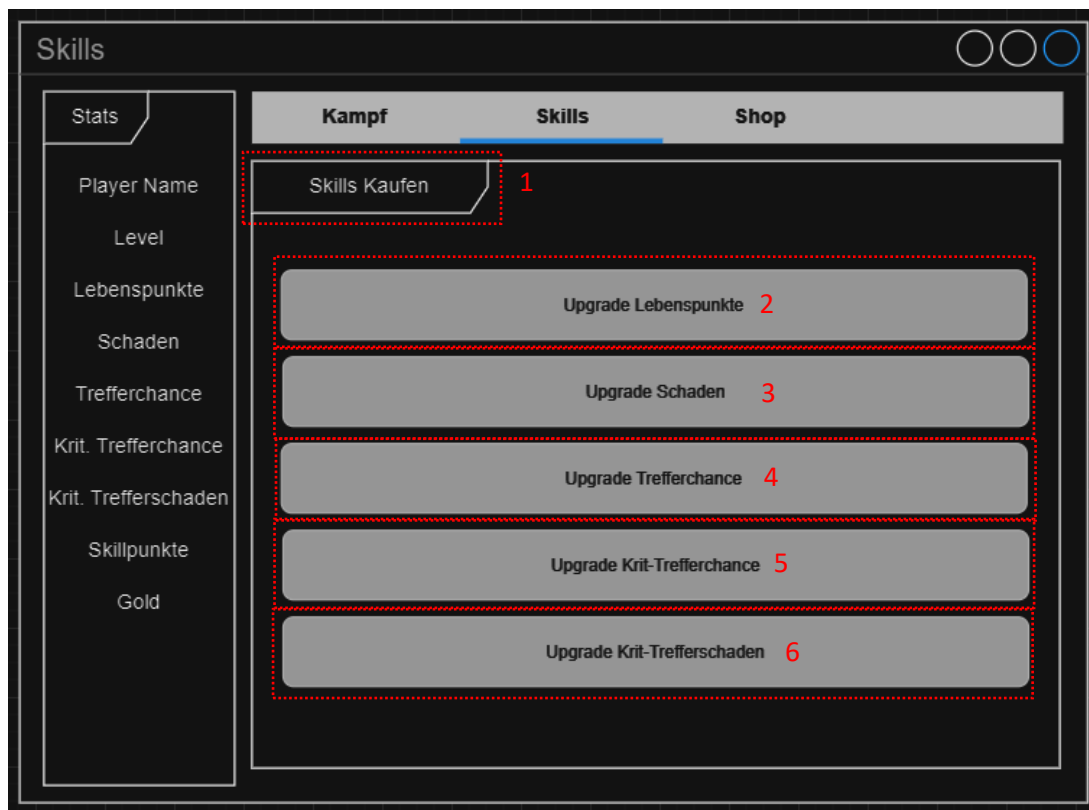


Abbildung 10: UI Entwurf Skills Tab

Funktion	UI-Element
1. Container für das Skillfenster	Frame
2. Erhöht die Lebenspunkte des Spielers	Button
3. Erhöht den Schaden des Spielers	Button
4. Erhöht die Trefferchance des Spielers	Button
5. Erhöht die Kritische Trefferchance des Spielers	Button
6. Erhöht den Kritischen Trefferschaden des Spielers	Button

Shop Tab



Funktion	UI-Element
1. Container für den Shop	Frame
2. Button um ein Schwert zu kaufen das den Angriff erhöht	Button
3. Button um Rüstung zu kaufen die den erhaltenen Schaden reduziert	Button
4. Button um einen Bogen zu kaufen der den Kritischen Trefferschaden erhöht	Button
5. Button um Schuhe zu kaufen welchen die Trefferchance erhöhen	Button

Programmierung & Grundaufbau

Klassendiagramm

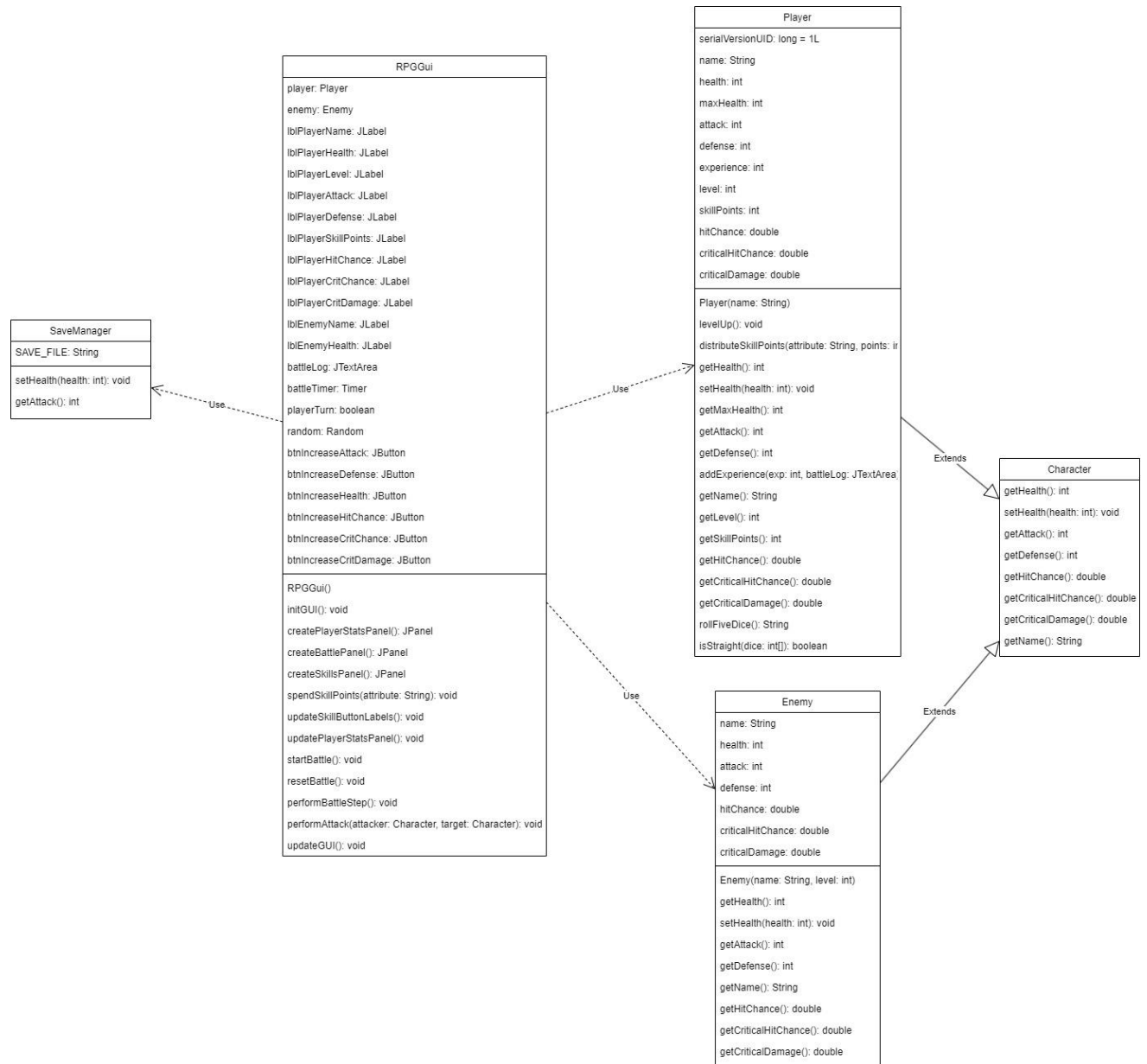
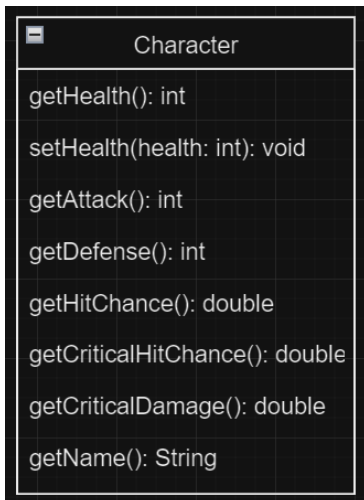


Abbildung 11: Klassendiagramm

Character Interface



Character	
getHealth(): int	
setHealth(health: int): void	
getAttack(): int	
getDefense(): int	
getHitChance(): double	
getCriticalHitChance(): double	
getCriticalDamage(): double	
getName(): String	

Abbildung 12: Character Interface

Beschreibung

Character ist ein Interface, das die grundlegenden Methoden für ein spielbares oder nicht-spielbares Charakterobjekt definiert. Dieses Interface stellt sicher, dass alle Klassen, die es implementieren, bestimmte Eigenschaften und Methoden wie Gesundheit, Angriff, Verteidigung, Trefferchance und kritische Trefferchance besitzen.

Funktionen

Name	Funktion
getHealth()	Gibt die aktuelle Gesundheit des Charakters zurück.
setHealth (health: int)	Setzt die Gesundheit des Charakters auf einen bestimmten Wert.
getAttack()	Gibt den Angriffswert des Charakters zurück.
getDefense()	Gibt den Verteidigungswert des Charakters zurück.
getHitChance()	Gibt die Trefferchance des Charakters zurück.
getCriticalHitChance()	Gibt die kritische Trefferchance des Charakters zurück.
getCriticalDamage()	Gibt den kritischen Schadensfaktor des Charakters zurück.
getName()	Gibt den Namen des Charakters zurück.

Player Klasse

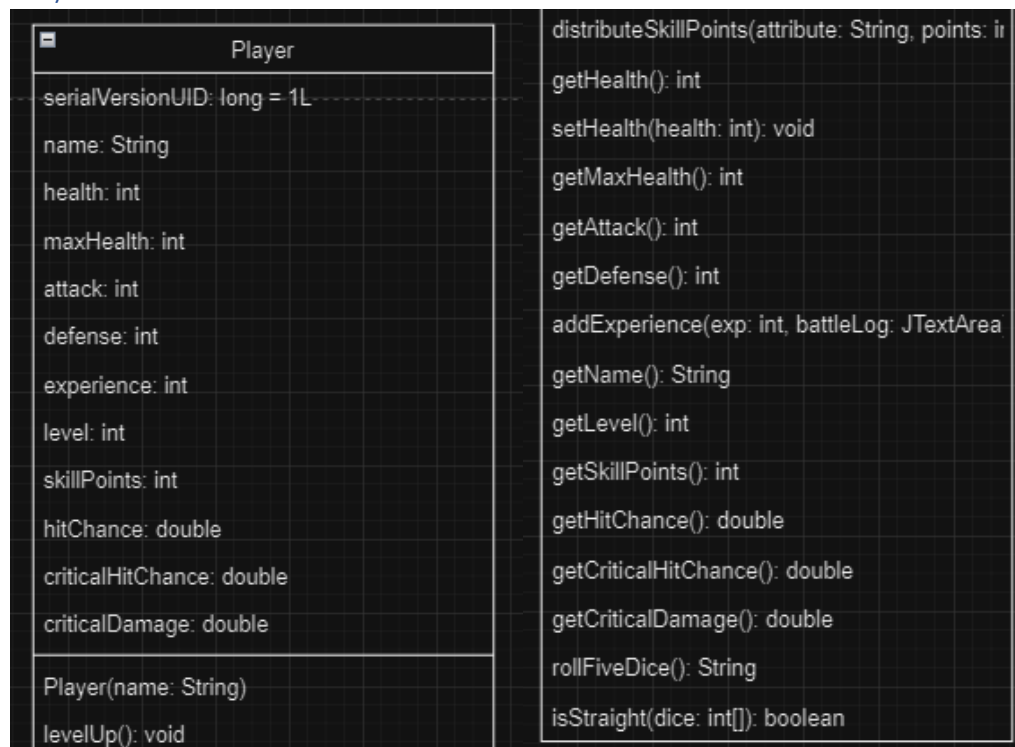


Abbildung 13: Player Klasse 1/2

Abbildung 14: Player Klasse 2/2

Beschreibung

Player ist eine konkrete Klasse, die das Interface «Character» implementiert und die Logik sowie die Attribute für den spielbaren Charakter definiert. Diese Klasse enthält spezifische Eigenschaften wie Gesundheit, Angriff, Verteidigung, Erfahrung, Level, Fähigkeitspunkte und verschiedene Wahrscheinlichkeiten für Treffer und kritische Treffer.

Funktionen

Name	Funktion
Player(name: String)	Konstruktor zur Initialisierung eines Spielers.
levelUp()	Erhöht das Level des Spielers und verbessert seine Attribute.
distributeSkillPoints(attribute: String, points: int)	Verteilt die Fähigkeitspunkte auf verschiedene Attribute.
getHealth()	Gibt die aktuelle Gesundheit des Spielers zurück.
setHealth(health: int)	Setzt die Gesundheit des Spielers auf einen bestimmten Wert.
getAttack()	Gibt den Angriffswert des Spielers zurück.
getDefense()	Gibt den Verteidigungswert des Spielers zurück.
addExperience(exp: int, battleLog: JTextArea)	Fügt Erfahrungspunkte hinzu und aktualisiert das Level bei Bedarf. «battleLog» wird verwendet um das Würfelergebnis ins Log zu schreiben.
getName()	Gibt den Namen des Spielers zurück.
getLevel()	Gibt das Level des Spielers zurück.

<code>getSkillPoints()</code>	Gibt die Skillpunkte des Spielers zurück.
<code>getHitChance()</code>	Gibt die Trefferchance des Spielers zurück.
<code>getCriticalHitChance()</code>	Gibt die kritische Trefferchance des Spielers zurück.
<code>getCriticalDamage()</code>	Gibt den kritischen Schadensfaktor des Spielers zurück.
<code>rollFiveDice()</code>	Würfelt fünf Würfel und überprüft auf einfache Kombinationen. (Paare, drei, vier oder Fünf gleiche).
<code>isStraight(dice: int[])</code>	Private Methode zur Überprüfung, ob die Würfel eine Straße bilden. (1,2,3,4,5 / 2,3,4,5,6)

Enemy Klasse

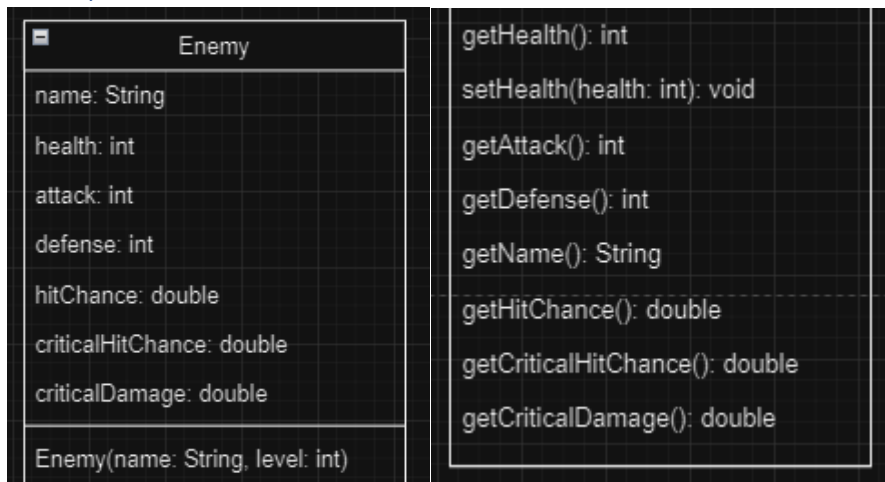


Abbildung 15: Enemy Klasse 1/2

Abbildung 16: Enemy Klasse 2/2

Beischreibung

Enemy ist eine konkrete Klasse, die das Interface «Character» implementiert und die Eigenschaften und Methoden für einen feindlichen Charakter definiert. Diese Klasse enthält Attribute wie Gesundheit, Angriff, Verteidigung und Wahrscheinlichkeiten für Treffer und kritische Treffer.

Funktionen

Name	Funktion
Enemy(name: String, level: int)	Konstruktor zur Initialisierung eines Gegners mit einem bestimmten Namen und Level.
getHealth()	Gibt die aktuelle Gesundheit des Gegners zurück.
setHealth(health: int)	Setzt die Gesundheit des Gegners auf einen bestimmten Wert.
getAttack()	Gibt den Angriffswert des Gegners zurück.
getDefense()	Gibt den Verteidigungswert des Gegners zurück.
getName()	Gibt den Namen des Gegners zurück.
getHitChance()	Gibt die Trefferchance des Gegners zurück.
getCriticalHitChance()	Gibt die kritische Trefferchance des Gegners zurück.
getCriticalDamage()	Gibt den kritischen Schadensfaktor des Gegners zurück.

SaveManager Klasse

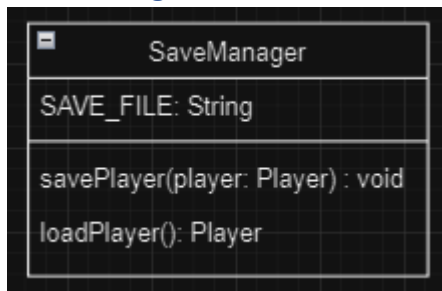


Abbildung 17: SaveManager Klasse

Beschreibung

Die Klasse SaveManager dient zur Verwaltung der Speicherung und des Ladens von Spieldaten für die Player-Objekte. Sie bietet statische Methoden, um das Player-Objekt in einer Datei zu speichern und es bei Bedarf wieder zu laden. Dies ermöglicht das Speichern des Fortschritts eines Spielers und das Fortsetzen des Spiels zu einem späteren Zeitpunkt.

Funktionen

Name	Funktion
savePlayer(player: Player)	Methode um die «Player» Klasse in eine Datei zu speichern.
loadPlayer()	Methode um die «Player» Klasse von einer Datei zu laden.

Programmierung

Zur Veranschaulichung meiner Programmierung habe ich ein Flussdiagramm gezeichnet. Das Flussdiagramm dient dazu, die Funktionsabläufe meines Spiels in einer sehr einfachen Form zu verstehen und nachzuvollziehen.

Flussdiagramm Programminitialisierung

Nachfolgend ein Flussdiagramm zur Initialisierung des Spiels.

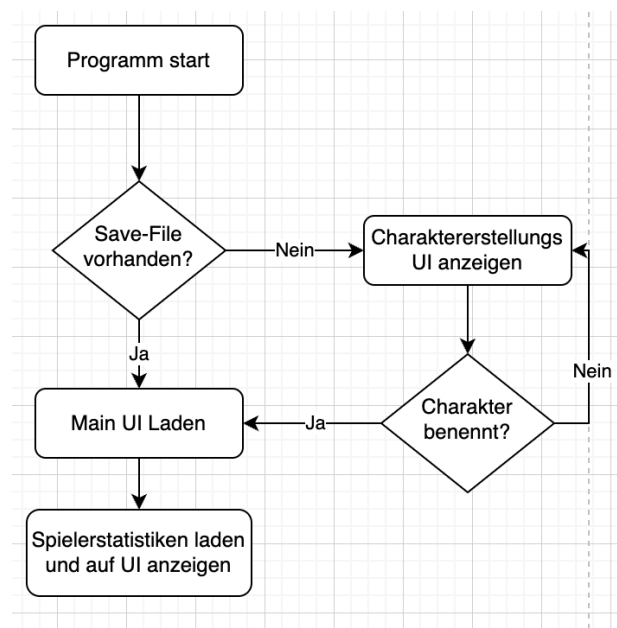


Abbildung 18: Flussdiagramm Programm Initialisierung

Flussdiagramm Kampf

Nachfolgend ein Flussdiagramm das den Programmablauf des Kampfes veranschaulichen soll.

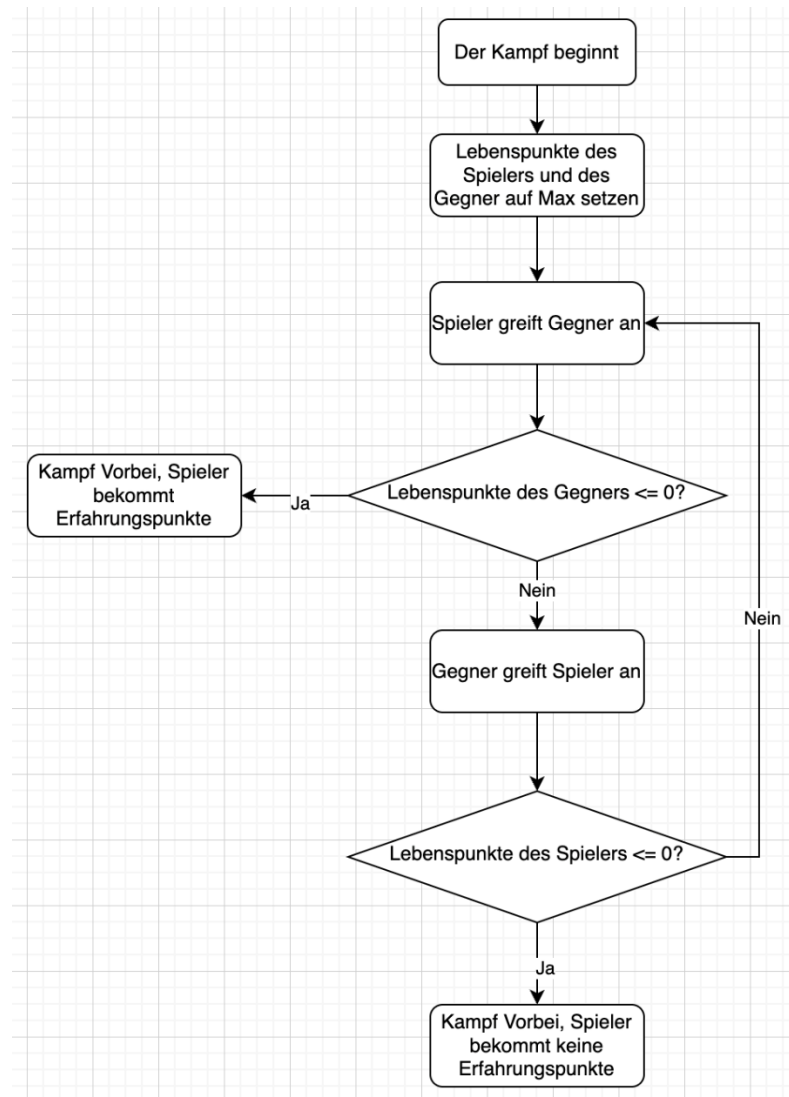


Abbildung 19: Flussdiagramm des Kampfes

Test-Bericht

Testfall 001: Spielstart und Charakterbenennung

Situation

Ich will ein neues Spiel starten und meinen Charakter benennen.

Soll-Resultat

Es soll ein Fenster erscheinen, welches ein Textfeld beinhaltet, um meinen Charakter zu benennen. Der Eingegabene Name soll anschliessend im Hauptfenster angezeigt werden damit der Spieler weiss, wie er seinen Charakter benennt hat.

Ist-Resultat

Erfolgreich:

Ein Fenster mit einem Textfeld erscheint und ich kann mein Charakternamen eingeben.

Beweis



Abbildung 20: Testfall 001, Namenseingabe

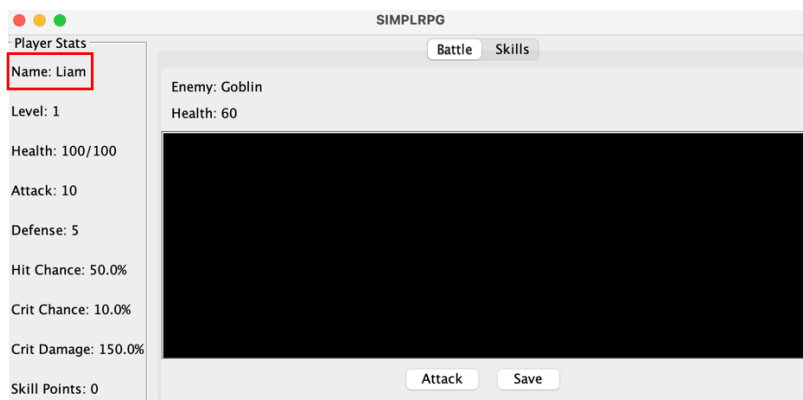


Abbildung 21: Testfall 001, Namenskontrolle

Testfall 002: Spiel speichern

Situation

Ich habe eine Weile gespielt und möchte nicht bei jedem Programmstart ein neues Spiel beginnen. Ich will meinen Fortschritt speichern damit ich bei meiner nächsten Spielesitzung dort fortfahren kann, wo ich aufgehört habe.

Soll-Resultat

Nach einem Klick auf den «Speichern» Knopf wird das «Player» Objekt serialisiert in einer Datei gespeichert.

Bei einem Start des Spiels wird überprüft, ob die Datei existiert. In meinem Testfall soll die Datei vorhanden sein. Das Spiel wird mit dem vorherigen Charakter weitergeführt werden.

Ist-Resultat

Erfolgreich:

Der Klick auf «Speichern» erstellt die Datei namens «player.sav» und speichert die Informationen. Das UI wird beim Start automatisch aktualisiert und ich kann dort weiterspielen, wo ich zuvor aufgehört habe.

Beweis

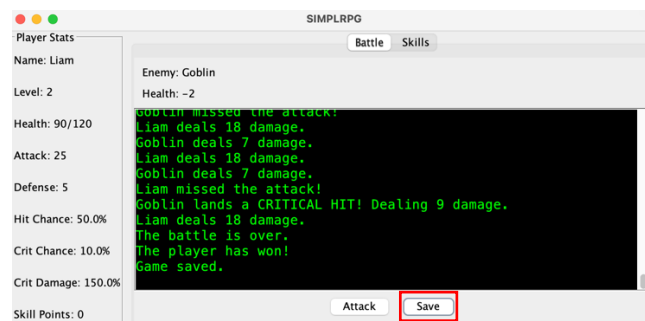


Abbildung 22: Testfall 002, Spiel speichern

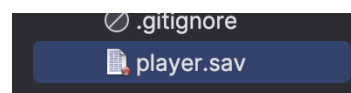


Abbildung 23: Testfall 002, Speicherdatei



Abbildung 24: Testfall 002, Spiel geladen

Testfall 003: Kämpfen

Situation

Ich will einen Kampf starten, um Erfahrungspunkte zu sammeln und meine Attribute zu verbessern.

Soll-Resultat

Nach einem Klick auf den «Kampf» Button sollen die Lebenspunkte des Gegners und die Lebenspunkte des Spielers auf ihre Maximalwerte zurückgesetzt werden. Anhand des Spielerlevels wird ein angemessen starker Gegner erstellt und der Kampf wird Schritt für Schritt durchgeführt. Der Verlauf des Kampfes kann im Log-Bereich der Applikation mitverfolgt werden. Falls der Spieler den Kampf gewinnt, werden dem Spieler Erfahrungspunkte hinzugefügt.

Ist-Resultat

Erfolgreich:

Es wurde der «Kampf» Button geklickt. Im Kampf-Log steht, das der Kampf begonnen hat. Es steht im Log abwechselnd, dass Spieler und Gegner sich angreifen mit jeweils dem verursachten Schaden. Am Schluss des Kampfes steht das der Spieler gewonnen hat, nachdem die Lebenspunkte des Gegners auf 0 oder kleiner fallen.

Beweis

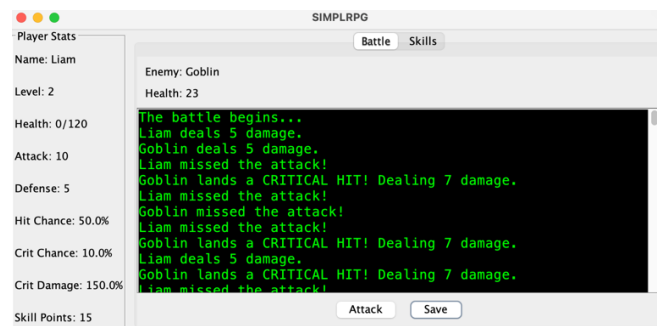


Abbildung 25: Testfall 003, Kampf beginn

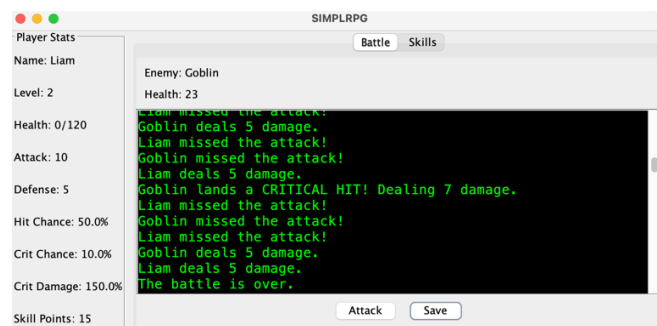


Abbildung 26: Testfall 003, Kampf ende

Testfall 004: Würfeln

Situation

Ich habe genug Erfahrungspunkte für ein Levelaufstieg gesammelt. Ich möchte nun Skillpunkte erhalten, um meine Attribute zu verbessern.

Soll-Resultat

Nach einem Stufenaufstieg sollen fünf Würfel gewürfelt werden und im «Poker-Style» ausgewertet werden. Je nach erreichtem Bild werden mehr oder Weniger Skillpunkte vergeben. Die Anzahl erhaltene Skillpunkte sowie das erreichte Bild werden im Kampf-Log ausgegeben.

Ist-Resultat

Erfolgreich:

Es wurden genug Erfahrungspunkte gesammelt, um einen Stufenaufstieg zu erreichen. Im Kampf-Log erscheint die Meldung «Player rolled Three of a Kind for 15 Skill Points!». Ich habe 3 Gleiche Zahlen gewürfelt und 15 Skillpunkte erhalten.

Beweis

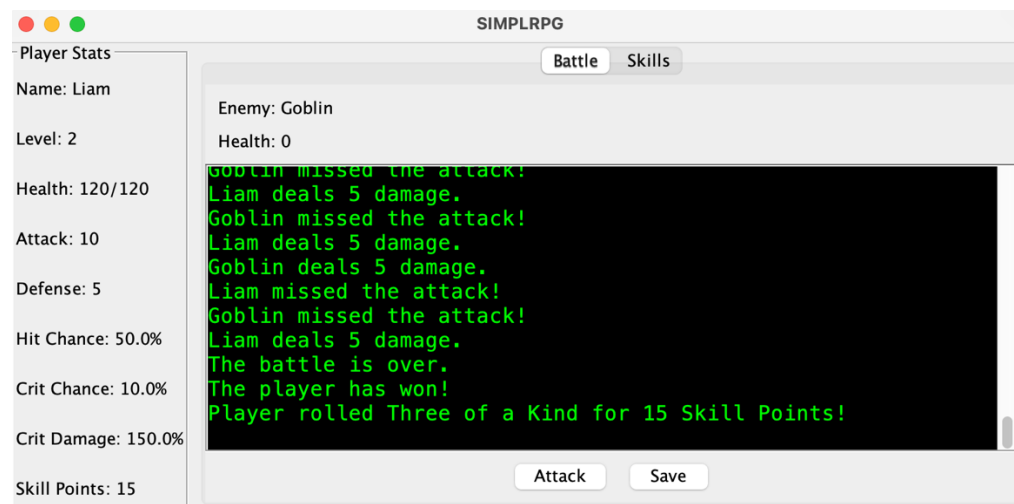


Abbildung 27: Testfall 004, Würfelresultat

Abbildungsverzeichnis

Abbildung 1: Logo von Git	6
Abbildung 2: Logo von IntelliJ	6
Abbildung 3: Logo von Java	7
Abbildung 4: Logo von Draw.io	7
Abbildung 5: Logo von Discord	8
Abbildung 6: Logo OneDrive	8
Abbildung 7: UI-Entwurf Startbildschirm:	9
Abbildung 8: UI-Entwurf Statistik Abschnitt.....	10
Abbildung 9: UI-Entwurf Kampf Tab.....	11
Abbildung 10: UI Entwurf Skills Tab	12
Abbildung 11: Klassendiagramm.....	14
Abbildung 12: Character Interface	15
Abbildung 13: Player Klasse 1/2.....	16
Abbildung 14: Player Klasse 2/2.....	16
Abbildung 15: Enemy Klasse 1/2.....	18
Abbildung 16: Enemy Klasse 2/2.....	18
Abbildung 17: SaveManager Klasse	19
Abbildung 18: Flussdiagramm Programm Initialisierung	20
Abbildung 19: Flussdiagramm des Kampfes	21
Abbildung 20: Testfall 001, Namenseingabe	22
Abbildung 21: Testfall 001, Namenskontrolle	22
Abbildung 22: Testfall 002, Spiel speichern.....	23
Abbildung 23: Testfall 002, Speicherdatei	23
Abbildung 24: Testfall 002, Spiel geladen.....	23
Abbildung 25: Testfall 003, Kampf beginn.....	24
Abbildung 26: Testfall 003, Kampf ende.....	24
Abbildung 27: Testfall 004, Würfelresultat.....	25

Glossar

1. **Roguelike-Videospiel**: Ein Subgenre von Videospielen, das oft durch zufällig generierte Level, permanenter Tod des Spielers und rundenbasiertes Gameplay gekennzeichnet ist.
2. **Idle-Videospiel** (auch bekannt als **Clicker Game** oder **Incremental Game**) bezeichnet ein Genre von Videospielen, bei dem der Fortschritt des Spielers weitgehend automatisch und ohne ständige Interaktion erfolgt. Der Spieler startet mit einfachen Aktionen, wie zum Beispiel dem Klicken auf ein Objekt, um Ressourcen zu sammeln oder Upgrades zu kaufen. Diese Upgrades steigern die Ressourcenproduktion, sodass im Laufe der Zeit zunehmend größere Fortschritte auch dann erzielt werden, wenn der Spieler nicht aktiv am Spiel teilnimmt.
3. **Version-Control-System**: Eine Software, die verwendet wird, um Änderungen an Dateien zu verfolgen, zu verwalten und zu speichern. Dies ermöglicht es Entwicklern, sicher zusammenzuarbeiten, verschiedene Versionen von Dateien zu verwalten und auf frühere Versionen zurückzugreifen.
4. **Integrated-Development-Environment (IDE)**: Eine Software, die Werkzeuge und Funktionen zur Entwicklung von Software bereitstellt, oft einschließlich Code-Editor, Compiler, Debugger und Build-Automatisierungswerkzeuge.
5. **UI (User Interface)**, auf Deutsch **Benutzeroberfläche**, bezeichnet die Schnittstelle zwischen einem Benutzer und einem System, beispielsweise einem Videospiel. Die UI umfasst alle visuellen und interaktiven Elemente, wie Buttons, Menüs, Icons, Texte, und Eingabefelder, die der Spieler nutzt, um mit dem Spiel zu interagieren.
6. **RPG (Role-Playing Game)**, auf Deutsch **Rollenspiel**, ist ein Videospiel-Genre, in dem der Spieler die Rolle eines oder mehrerer Charaktere übernimmt, deren Fähigkeiten und Eigenschaften er im Verlauf des Spiels entwickeln und anpassen kann. RPGs zeichnen sich durch tiefgründige Geschichten, Charakterentwicklung, strategische Kämpfe und eine Vielzahl von Quests und Missionen aus.
7. Ein **kritischer Treffer** ist ein Konzept in Videospielen, insbesondere in Rollenspielen (RPGs), Strategie und Actionspielen, bei dem ein Angriff eine erhöhte Menge an Schaden verursacht. Kritische Treffer treten zufällig auf, wobei die Wahrscheinlichkeit oft durch Spielmechaniken wie **Kritische Trefferchance** oder **Krit-Chance** bestimmt wird.