Projektdokumentation Software-Entwicklung 3

Modul-Nr.	113330a	
Mitwirkende	Anh Thu Bui	39310
	Jana Mößner	39519
	Lea Bretz	39217
	Natalie Hußfeldt	39231
	Sandra Herbst	39229
Lehrender	Prof. Dr. Jens-Uwe Hahn	
Betreuer	Alexander Blank	

Einführung und Ziele

Was ist Case20?

MVP (Minimum Viable Product)

The development of a point and click game, which allows users to collect and use items to end the game.

Kurze Beschreibung:

- Case2020 ist ein Point and Click Game.
- Der Hauptzweck von Case2020 dient dem Spaß und das Verwenden eigener Problemlösungsfähigkeiten, um das Rätselspiel zu lösen.
- Der Spieler widmet sich einer Mission und spielt diese anhand einer Storyline durch.
- Die funktionalen Anforderungen sind einfach genug, dass das Spiel selbsterklärend gespielt werden kann.
- Mit Java Version 14 implementiert.
- Die interaktive und multimediale Benutzeroberfläche wurde mit JavaFX in Kombination mit CSS umgesetzt.

Storyline des Spiels:

Wie Sie wahrscheinlich bereits schon gesehen haben, so ist Professor Dr. Deany ohne Spur verschwunden. Gleichzeitig ist mit ihm auch unsere einzige Möglichkeit zum Überleben dieser Pandemie erloschen.
Er ist heute nicht am vereinbarten Treffpunkt aufgetreten und wir vermuten, dass etwas Schlimmes passiert sein könnte. Wir bitten Ihnen um dringende Hilfe.

Fall-Akte Nr. **#2020**

Finde das Impfmittel und dessen Rezept.
Sammeln Sie Hinweise das Verschwinden des Professors Dr. Deany und lösen Sie den Fall.

Nehmen Sie den Auftrag bitte an.

Gezeichnet,

Die Regierung des Staats JANSL.

Wesentliche Features:

- Der Spieler erstellt sich seinen eigenen Character
- Der Spieler steuert „seinen Charakter“ durch Point- und Klickaktionen auf dem Computer
- Bittet den Spieler seine Problemlösungsfähigkeiten zu verwenden, um das Ziel des Spiels zu erreichen (den Corona-Impfstoff zusammen zu mixen)
- Dafür kann der Spieler verschiedene Räume erkunden und durch erfolgreiches Spielen von Minigames Zugang zu diesen erlangen
- Der Spieler befindet sich mitten in der Story und Objekte mit denen er interagiert, geben ihm Hinweise, die in Textfeldern auf dem Bildschirm erscheinen
- Der Spieler sammelt Gegenstände ein , die er zum Lösen von Minigames benötig
- Der Spielstand kann gespeichert werden
- Ergebnisse werden im Highscore durch eine Serverbindung angezeigt

So starten Sie die Anwendung

Unsere Anwendung ist in zwei Teile unterteilt.

- Das Verzeichnis pointnclick_Game ist unser Spiel selbst,
- Das Verzeichnis pointnclick_Server ist unser Server, der online gehostet wird.

Die Hauptklasse für unser Spiel befindet sich in pointnclick_Game/.../gui/Main.java. Führen Sie Main.java aus, um die Anwendung zu starten.

Qualitätsziele

Qualitätsziele	Motivation und Erläuterung
Unabhängig von Betriebssystem (Functionality)	Case2020 lässt sich auf MacOs, Windows und Linux (Linux jedoch teilweise mit Problemen die unlösbar sind) ausführen
Intuitive Spielbarkeit (Usability)	Der Spieler soll nie, garnicht wissen was er weiter tun soll
Verschiedene Sprachen (Changeability)	Der Spieler kann die Sprache im Spiel ändern
Verschiedene Themes (Changeability)	Der Spieler kann zwischen light- und dark-Theme wechseln
Keine Spielunterbrechung (Usability)	Case2020 sollte nicht abstürzen
Ohne viel Equipment (Usability)	Case2020 ist auch ohne professionelles Gaming Setup problemlos spielbar. Es ist lediglich eine Maus und eine Tastatur notwendig.
Minispiel lösen (Effizienz)	Bei lösen der Minispiele ist direkt sichtbar ob sie richtig oder falsch gelöst wurden.
Funktionsfähiger develop branch (Reliability)	Der develop branch ist immer auf einem funktionsfähigen Zustand und kann ausgeführt werden

Randbedingungen und Voraussetzungen

Randbedingungen	Erläuterungen/ Hintergrund
CI/CD	Automatisiertes Testen des Codes, um Bugs frühzeitig zu finden und anschließend bereitzustellen. Ziel ist es, die aktuellste Codebasis zur Verfügung zu haben.
Branching	Verwenden von mehreren Branchen, um gleichzeitig zu arbeiten, ohne andere Teammitglieder zu stören. Zusätzlich wird immer eine "stabile Version" gesichert.
Issue Tracking und Priorisierung der Features	Zeitmanagement - Überblick über alle offenen Aufgaben
Parallelisierung	Verwenden von Threads und parallelen Streams, um beispielsweise die Wartezeit auf angeforderte Daten zu verringern und die Performance zu verbessern
Clean Coding	Unter anderem umgesetzt mit Dependency Injection
Datenbankanbindung und Networking	In Kombination eingesetzt, um die Daten des Players auf einem Server zu speichern
UI	Oberfläche des Spiels

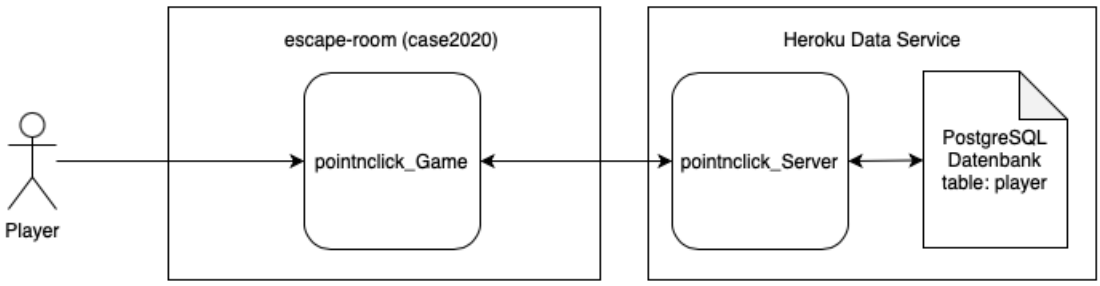
Anmerkung:

- Auf Linux Ubuntu ist der Textfenster im Spiel sehr weit nach oben verschoben. Betriebssysteme wie MacOS und Windows hingegen positionieren diesen Textfenster korrekt am unteren Rand des Spiels.

- Grund der wenigen Testklassen liegt hierbei, dass unsere gesamte essenzielle Logik mit JavaFX zusammenhängt. D.h. zum Testen dieser Logik müsste man die Möglichkeit haben, entsprechende GUI Events ausführen zu können.

Kontextabgrenzung

Fachlicher Kontext



Player

Die Anwendung case2020 ist eine single-player bzw. single-user Anwendung. Der Spieler interagiert alleinstehend und direkt der Anwendung (Pointnclick_Game).

Pointnclick_Game (Client)

Ist eine lokale Anwendung und beinhaltet die „Game Engine“. Also alle Bestandteile der GUI sowie die Spiellogik (Level, Player, GameState etc.) und die Implementation der Server Endpoints.

Pointnclick_Server (gehostet von Heroku)

Die einzige externe Schnittstelle der Anwendung ist der Pointnclick_Server. Mit Pointnclick_Game werden requests an den Server gesendet der die Spielerdaten Online in einer Datenbank abspeichert oder sie als response an die Anwendung zurückgibt.

Lösungsstrategien

Qualitätsziel	Dem zuträgliche Konzepte, Strukturen und Prinzipien
Unabhängig von Betriebssystem (Functionality)	• Verwendung von Java • Betriebssystem-unabhängige libraries und plug-ins
Intuitive Spielbarkeit (Usability)	• Informationen zum Spielablauf werden nicht vorweggenommen. • Bei der Konzeption von Case2020 wurde ein klarer Lösungsweg definiert. • Der Lösungsweg ergibt sich aus dem Spielverlauf. • Das Spielprinzip von Case2020 basiert auf dem Belohnungsprinzip. Da es sich um ein Rätselspiel handelt wird der Spieler bei jedem Fortschritt im Spielverlauf eine entscheidende komponente erhalten, die er benötigt um im Spiel weiter Fortzuschreiten.
Verschiedene Sprachen (Changeability)	• Einsatz von Resource Bundle • Einsatz von .properties files um dynamische änderbarkeit der Sprache zur Laufzeit zu ermöglichen • Implementation einer Schnittstelle zur Änderung der Sprache. • Implementation dieser Schnittstelle die den Zugriff auf Spracheinstellungen vom gesamten Spiel aus ermöglichen
Verschiedene Themes (Changeability)	• Einsatz von zwei getrennten CSS Stylesheets • Implementation einer Schnittstelle die das ändern des Stylesheets zur Laufzeit ermöglicht • Implementation dieser Schnittstelle die den Zugriff auf die Erscheinungsbild - Einstellungen vom gesamten Spiel aus ermöglicht
Keine Spielunterbrechung (Usability)	• Verwendung von Exception Handling an passenden und sinnvollen Stellen • Performante Laufzeit durch das Einhalten von Clean Code Prinzipien und die Konzeption einer Architektur mit Loser Kopplung und hoher Kohärenz • Verwendung weiterer Architekturprinzipien die eine stabile Laufzeit ermöglichen

Ohne viel Equipment (Usability)	• Konzeption und Entwicklung eines Spiels, dass weder einen hohen Grafik Rechenleistung noch eine besondere Prozessorleistung erfordert. • Verzicht auf komponenten die eine Besondere Peripherie erfordern würden.

Minispiel lösen (Effizienz)	• Einsatz des Belohnungsprinzips
	• Wenn ein Minispiel gelöst wurde erfährt der Spieler das entweder:
	  ○ durch das Erhalten einer Belohnung in Form eines Items.
	  ○ durch ein anderes visuelles Feedback wie einen Text oder das automatische schließen des Minigames mit einer folgenden Aktion.

Funktionsfähiger develop branch (Reliability)	• Umsetzung Folgender Regeln innerhalb des entwicklungs Teams:
	  ○ Der develop branch wird in den bestehenden feature branch gemerged der auf develop gepusht werden soll. Somit werden Merge Konflikte immer auf dem feature branch gelöst.
	  ○ Vor jeder weiterarbeit auf einem feature branch wird der develop branch gepullt und mit dem entsprechenden feature branch gemerged um einen entwicklungsrückstand auf dem feature branch zu verhindern.
	  ○ Es werden nur feature branches auf den develop branch gepusht die voll funktionstüchtig und fertig sind.
	  ○ Es dürfen auch feature branches auf develop gepusht werden die, abhängig von anderen Komponenten, an einem Späteren Zeitpunkt weiterentwickelt werden müssen.
	  ○ Nicht mehr bestehende feature branches werden von allen lokal entfernt.

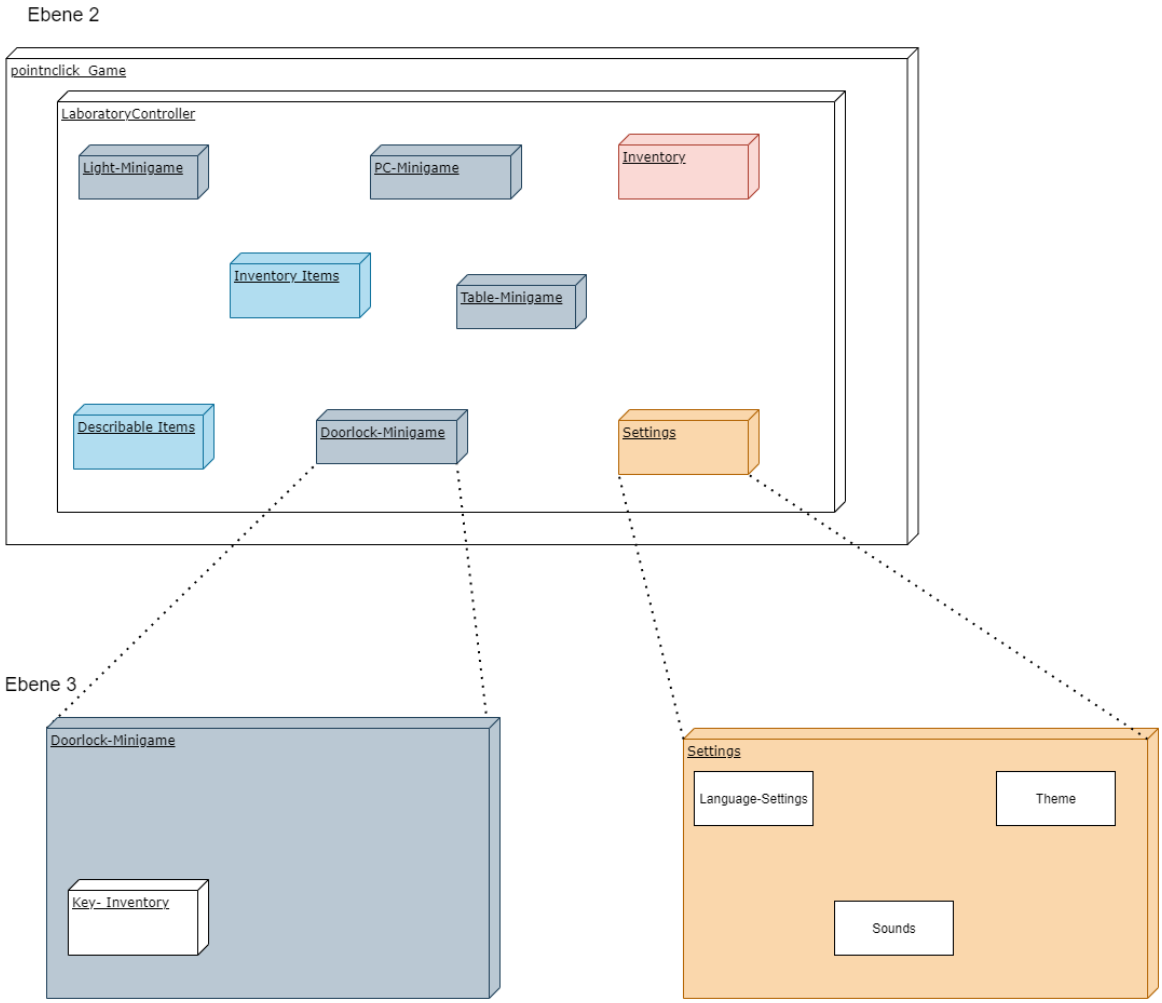
Bausteinsicht

Ebene 1

Siehe Kontextabgrenzung

Ebene 2

Folgend ist vereinfacht die Szene “Laboratory” mit einigen ihrer Komponenten oder auch ihrer “Subsysteme” dargestellt, welche auch als Blackboxen bezeichnet werden können. Die gestrichelten Linien zeigen die Komponenten, die nachfolgend näher beschrieben werden.



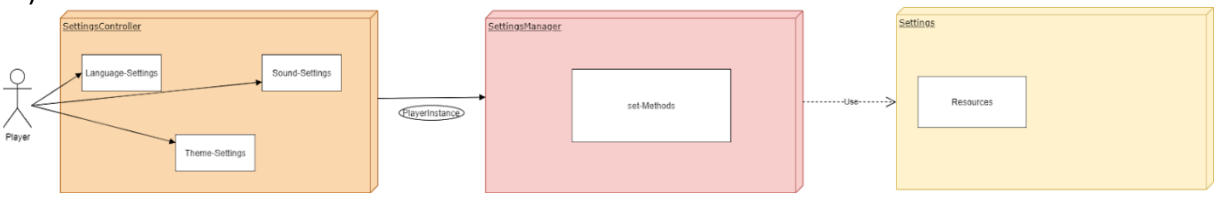
Ebene 3

1.)

Doorlock-Minigame

Enthält als zentrales Modul das Key-Inventory, welches signifikant für das Lösen des Minigames ist. Hier werden alle Keys, die der Spieler bereits gefunden hat, in einem separaten Inventory geladen und für den Spieler zum Lösen des Minigames bereitgestellt. Löst der Spieler das Minigame, erhält er Zugriff zur nächsten großen Whitebox, des “Livingrooms”. Macht der Spieler jedoch Fehler, wird der Failcount erhöht und Strafzeit hinzu berechnet.

2.)



Spieler

Der Spieler hat die Möglichkeit, im Laboratory auf seine Einstellungen bezüglich der Sprache, des Themes (Light oder Dark) und der Lautstärke der Musik und Soundeffects zuzugreifen und diese zu ändern.

SettingsController

Ist das “UI”, welches der Spieler sieht. Die Player-Klasse enthält Methoden, mit denen diese Settings eingestellt werden können.

Settings

Enthält im Prinzip die gesamten Ressourcen, wie z.B. das ResourceBundle für die Sprachen und die CSS-Stylesheets für die Themes. Es kann als eine Art “Verzeichnis” für alle möglichen Einstellungen angesehen werden.

Laufzeitsicht

Vorgang, wie der Spieler in einem Level ein Item aufheben kann.

Alle Items, die man aufheben kann, sind in InventoryItems.java als Enum Konstante definiert und halten jeweils ihre String URL zu ihrem PNG.

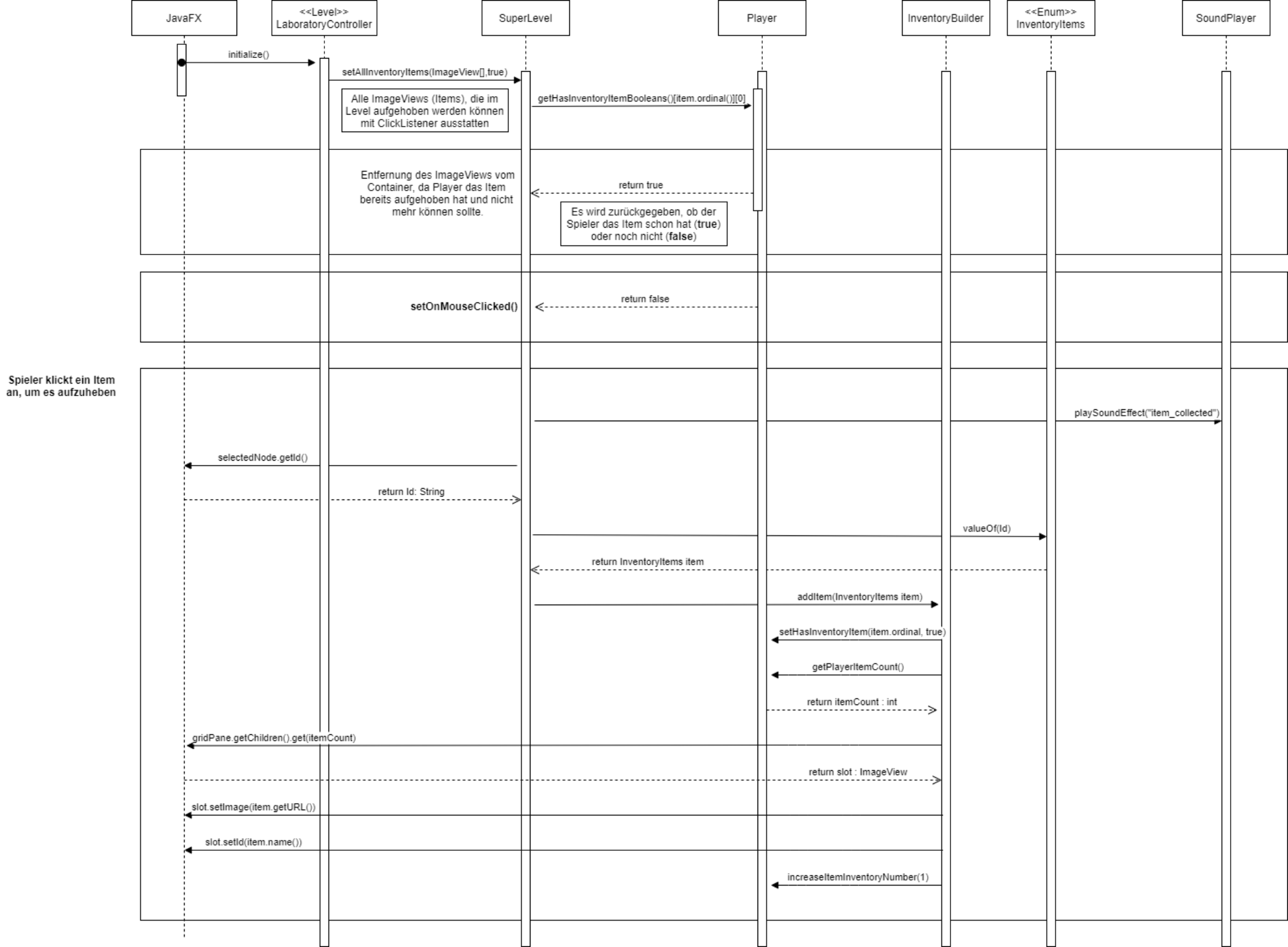
Um die jeweiligen Items mit den ImageViews von JavaFX verbinden zu können und herauszufinden, welches ImageView welches Item repräsentiert, verändern wir die ID des ImageViews zum Namen ihrer Enum Konstante.

Mit **node.getId()** erhält man den Namen des ImageViews (Beispielsweise "**NOTE**"), welches gleichzeitig auch im Enum als "**NOTE**" Konstante definiert ist.

Dadurch erhalten wir mit **InventoryItems.valueOf(node.getId())** das Enum zurück und damit auch dessen PNG URL. Das Item kann hiermit eindeutig identifiziert werden und der InventoryBuilder kann zu seiner GridPane(Die das Inventory des Players darstellt) das Bild zu sich hinzufügen, da dieser nun die **PNG URL** kennt.

Für die Persistenz speichert der **PlayerProgressManager** zusätzlich ab sobald ein Item vom Spieler aufgehoben wurde, um es beim nächsten Spielstart wieder im Inventar laden zu können. *Tada! Der Spieler hat ein Item erhalten!*

Siehe Sequenzdiagramm auf der nächsten Seite.


JavaFX
<<Level>>
LaboratoryController
SuperLevel
Player
InventoryBuilder
<<Enum>>
InventoryItems
SoundPlayer
initialize()
setAllInventoryItems(ImageView[],true)
Alle ImageViews (Items), die im Level aufgehoben werden können mit ClickListener ausstatten
getHasInventoryItemBooleans()[item.ordinal()][0]
Entfernung des ImageViews vom Container, da Player das Item bereits aufgehoben hat und nicht mehr können sollte.
return true
Es wird zurückgegeben, ob der Spieler das Item schon hat (true) oder noch nicht (false)
setOnMouseClicked()
return false
Spieler klickt ein Item an, um es aufzuheben
playSoundEffect("item_collected")
selectedNode.getId()
return Id: String
valueOf(Id)
return InventoryItems item
addItem(InventoryItems item)
setHasInventoryItem(item.ordinal, true)
getPlayerItemCount()
return itemCount : int
gridPane.getChildren().get(itemCount)
return slot : ImageView
slot.setImage(item.getURL())
slot.setId(item.name())
increaseItemInventoryNumber(1)

Querschnittliche Konzepte

Abhängigkeit zwischen Modulen

In Case2020 sind die Klassen lose miteinander gekoppelt. Klassen die abhängig von anderen Modulen sind kümmern sich nicht selbst um die Auflösung dieser Abhängigkeit. Stattdessen wird über Dependency Injection mit Spring die notwendigen Objekte injiziert.

Benutzungsoberfläche

Case2020 hat eine graphische Oberfläche, mit der der User interagiert (pointed und clicked). Die herausstechende Besonderheit ist der 16 Bit Pixel Style indem die Elemente gezeichnet worden sind. Außerdem ist es möglich das Theme der Benutzeroberfläche nach Nutzer Belieben auf Light oder Dark zu stellen.

Logging

Um Case2020 leichter erweitern zu können, sind Analysemöglichkeiten vorhanden die in Form von Logs realisiert wurden. Welches mit der Library log4j umgesetzt ist.
Der Verlauf des Spiels, das Verhalten des Spielers und Erfolg oder Misserfolg in Minispielen sind über das log file detailliert nachvollziehbar.

Auch abgefangene Error Cases werden im log ausgegeben, wodurch die Fehlerfindung erleichtert wird.

Testbarkeit

Die Funktionalität der Grundmodule von Case2020 wird durch Unit-Tests sichergestellt. In der Quelltextstruktur ist neben dem Ordner src/main, wo die Java-Quelltexte der Module abgelegt sind, ein Ordner *src/test* zu finden. Dort sind die Klassen zu finden die mit JUnit 4 getestet werden.

Da sehr viel in Case2020 von einer User Interaktion, wie klicken, abhängt und dies mit JUnit nicht abgebildet werden kann, kann der Spielablauf nicht getestet werden.

Entwurfsentscheidungen

Entscheidung für die wesentlichen Libraries

Server

Spring Boot

Der Server basiert auf einer Webanwendung mithilfe des Spring Boot Frameworks. Spring Boot ist in Java sehr bekannt und weit verbreitet, außerdem unterstützten uns die erworbenen Kenntnisse vom Modul "Web Development Backend" bei der Nutzung von einem Node JS Express REST Server sehr.

PostgreSQL

Das war weniger eine Entscheidung, die von uns als Gruppe kam, sondern mehr Zwang. **Heroku.com** bietet lediglich für diese eine Datenbankart eine Unterstützung an, welche noch im kostenlosen Rahmen verbleibt. Demnach haben wir uns nicht groß weiter über andere Möglichkeiten umgesehen. Letztendlich macht es seinen Job wie es soll und wir haben bisher keine schlechte Erfahrung damit sammeln können.

Game (Client)

JavaFX als UI

Diese Library war uns durch SE2 noch vertraut. JavaFX kann durchaus ziemlich stur und hartnäckig sein. Da wir bei diesem Projekt aber tatsächlich unseren Workload um das X-fache sprengen würden, wenn wir uns etwas anderes angesehen hätten, verblieben wir bei dieser Wahl.

Spring Framework - IoC I Dependency Injection

Bei unserer Recherche stellte sich heraus, dass Spring sehr gerne für DI genutzt wird. Demnach haben wir es folglich mit Spring umgesetzt.

Für die Bean-Erstellung haben wir uns für die XML Art der Beschreibung der Beans entschieden, da es übersichtlich ist und alles in einer einzigen Konfigurationsdatei steckt.

Für die Erstellung des Playerbeans haben wir eine PlayerFactory.java Klasse angelegt, damit wir bei Vorhandensein einer Speicherdatei den Spieler in die Bean hineinladen können, ansonsten wird ein neuer Spieler generiert.

Architekturentscheidung

Interfaces

Als Interface haben wir uns für **ILevel.java** entschieden. Letztendlich soll **ILevel** alle zu implementierenden Methoden auflisten, damit in einem Level folgende Dinge gemacht werden können:

- Der Spieler kann Level Räume wechseln
- Der Spieler kann Minigames spielen
- Der Spieler kann sich Gegenstände ansehen, die auf dem Boden liegen
- Der Spieler kann sich Items im Inventar ansehen
- Der Spieler kann Items vom Boden aufheben und im Inventar sammeln
- Der Spieler kann sein Spiel abspeichern
- Der Spieler kann sein Inventar nach Itemtypen sortieren lassen

SuperLevel.java implementiert dieses Interface und ihre Methoden, um die JavaFX Logik zu diesen Methoden anzupassen. Durch das Interface bezwecken wir, dass nicht versehentlich

wichtige Methoden der Level Logik entfernt werden, die wesentlich für das Spielgeschehen sind.
Nicht jeder **LevelController** muss Items implementieren, die aufgehoben werden können/beschrieben werden können etc. Daher rufen die jeweiligen **LevelController** die Methode aus **SuperLevel.java** auf, sofern sie diese Logik implementieren möchten.

Abstrakte Klassen

SuperLevel.java ist eine abstrakte Klasse, sie soll selbst nicht als Objekt instanziiert werden, sondern von jeweiligen GUI Controllern extended werden, wenn diese ein Level darstellen sollen.

SuperController.java ist eine abstrakte Klasse und hat jene Methoden implementiert, die ein Controller generell benötigt. Dies betrifft beispielsweise Methoden zum Szenenübergang und Animationen.

Factories

PlayerFactory.java ist unsere Factory und erzeugt die Player Bean durch Spring. Dabei wird durch GameState.java der Spielstand geladen. Sofern aber kein Spielstand vorhanden ist, wirft es eine Exception in der Factory und diese gibt eine neu erzeugte Player Instanz heraus. Dadurch sind diese 2 Fälle abgedeckt:

- Der Spieler speichert und startet die Software etwas später erneut. Er erhält seinen Spielstand zurück.
- Der Spieler hatte noch nie einen Spielstand und spielt zum ersten Mal.

Erläuterung der Logik von einzelnen Klassen (Die evtl etwas unverständlich sind)

Sound

Für die Musik in unserem Spiel nutzen wir den JavaFX MediaPlayer. Dazu erstellten wir eine dazugehörig verantwortliche Klasse **SoundPlayer.java.**

Diese Klasse unterscheidet zwischen diesen Anwendungsfällen:

- Eine Hintergrundmusik wird abgespielt
- Ein Soundeffekt wird abgespielt

Diese Unterscheidung ist dazu da, um die Hintergrundmusik über mehrere Szenen hinweg weiter manipulieren zu können, da darauf eine Referenz existiert. Uns ist bewusst, dass diese Referenz verloren geht, wenn eine zweite Hintergrundmusik gestartet wird. In unserer Applikation jedoch wird niemals mehr als eine Hintergrundmusik gleichzeitig laufen.

Die Referenz auf ein Soundeffekt-MediaPlayer-Objekts wird nicht gespeichert, da ein Soundeffekt nur wenige Sekunden läuft und niemals in andere Szenen übergehen wird und daher keine weitere Manipulation benötigt.

Weitere Anmerkungen:

- Light Mode CSS wurde nicht designed, da schlichtweg die Zeit dazu fehlte. Dark Mode hingegen ist vollständig designed.

Qualitätsszenarien

Dieser Abschnitt beinhaltet konkrete Qualitätsszenarien, welche die zentralen Qualitätsziele aber auch andere geforderte Qualitätseigenschaften besser fassen. Sie ermöglichen es, Entscheidungsoptionen zu bewerten.

Nr.	Szenario
1	Der Spieler möchte das Spiel auf Deutsch spielen, da er nicht fließend Englisch spricht. In den Einstellungen kann er zu jeder Zeit zwischen Deutsch und Englisch wechseln .
2	Das Design ist zu hell für den Spieler, weil er nachts spielt. In den Einstellungen kann er zu jeder Zeit zwischen Light- und Dark-Theme wechseln
3	Der Code kann aus unbekannten Gründen nicht ausgeführt werden. Durch Auslösen von Exceptions und Implementieren von Protokollierungen in jeder Klasse, können die Ausnahmemeldungen in der A1.log-Datei nachvollzogen werden.
4	Der Spieler weiß nicht, wie er in dem Spiel vorankommt und wie er es zu bedienen hat. Das Tutorial Pop-Up „?“erklärt die Wichtigsten Funktionen des Spiels.
5	Der Spieler hat keine hochprofessionelle Spielausrüstung. Der Spieler benötigt nur eine Maus und eine Tastatur, um das Spiel zu spielen
6	Der Developer pushed etwas, das fehlerhaft ist. Durch das Verwenden von git CI / CD zur Durchführung automatisierter Tests (Build-Test), sieht man sofort, dass es zu Fehlern kam.
7	Der Developer möchte nachvollziehen, wie weit sein Branch hinter develop ist und/oder auf welchen Branches seine Mitentwickler arbeiten. Durch die Verwendung von Source Tree und Git flow hat man jederzeit einen guten Überblick.
8	Der Spieler möchte die Highscore Liste einsehen. Damit er nicht auf die Antwort des Servers warten muss, wenn er den Highscore-Controller öffnet, wird der Request an den Server gesendet, sobald das Spiel gestartet wird. Somit kann man diese Bonuszeit nutzen, um die Daten zu bekommen.
9	Dem Spieler ist die Hintergrundmusik zu laut. Er kann dies jederzeit in den Einstellungen anpassen
10	Der Spieler möchte sein Spiel zwischenspeichern, um ein anderes mal weiter zu spielen. Er kann das Spiel jeder Zeit speichern und seinen Spielstand erneut laden.

Risiken und technische Schulden

Die folgenden Risiken wurden zu Beginn des Vorhabens identifiziert.

Risiko: Aufwand der Implementierung

Es liegt keinerlei Erfahrung mit der Programmierung eines Point and Click Escape Rooms vor. Gleichzeitig wirkt die gesamte Story durch Spiele, Grafiken und Rätsel zu implementieren umfangreich und kompliziert.

Die Implementierung von Case2020 verläuft als Semesterprojekt Es ist unklar, ob die Zeit reicht, um innerhalb der vorgegebenen Zeit vorzeigbare Ergebnisse zu präsentieren.

Risikominderung

Der Aufwand wird dadurch reduziert, dass zunächst an der Umsetzung des MVP gearbeitet wird und erst dann nach und nach Erweiterungen folgen werden.

Früher Beginn mit der Erstellung von Grafiken und Story-Entwicklung. Beschränkung der GUI auf zwei Räume (Konsequenzen bezüglich der Spielstärke, und keine bezüglich der Umsetzungen der Anforderungen siehe Randbedingungen)

Risiko: Java für Spieleentwicklung

Programmierer implementieren Computerspiele meist mit C++ als Programmiersprache. Java wird eher selten zur Spielentwicklung genutzt . Java hat eine Garbage Collection, verwaltete Laufzeit. 99% der Zeit ist dies ein großer Vorteil und macht das Programmieren einfacher und weniger fehleranfällig. Allerdings verursacht der Garbage-Collector gelegentliches Latenzproblem für Spiele, da Garbage Collection-Zyklen spürbare Pausen verursachen können.

Risikominderung

Die Entwicklung mit JavaFX war schon bekannt und leicht verständlich. JavaFX dient dazu interaktive und multimediale grafische Benutzeroberflächen zu entwickeln und kann durch Scene Builder und CSS einfach implementiert werden. Zudem ist das Spiel noch nicht so umfangreich und Latenzprobleme durch Garbage Collection sind nicht bemerkbar. Zudem hat Case2020 eine 2D-Grafik und keine 3D-Animation.

Risiko: Anbindung an Server für Highscoreliste

Es liegt keinerlei Erfahrung mit der Programmierung eines Servers vor. Gleichzeitig muss die Verbindung kostenlos sein und die Connection muss stabil sein, der Server darf nicht zur Bewertung des Projekts nicht erreichbar sein.

Risikominderung

Die Serveranbindung wurde nicht in das MVP aufgenommen und erst später als Feature hinzugefügt, nachdem man eine gute Möglichkeit zur Umsetzung gefunden hatte.

Lösungsvorgang

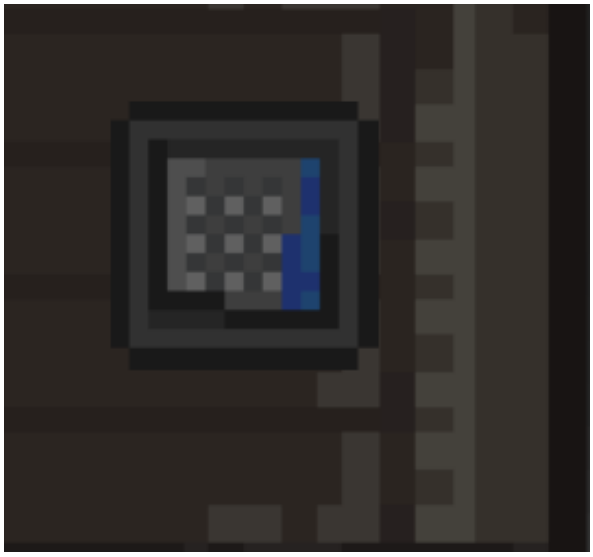
Ziel des Spiels ist es die vier essenziellen Zutaten für den Impfstoff zu finden und diese zusammen zu mixen. (siehe Minigame-Table)

Die vier Zutaten sind:

- **Hefe**
- **NaCl**
- **Heißes Wasser**
- **Zucker**

Code-Minigame

Klicken Sie auf das Tastenfeld, um es zu benutzen



Zu Beginn des Spieles muss der vierstellige Code der Eingangstür geknackt werden. Dazu geben die abgenutzten Tasten sowie die Kontrollleuchte Hinweise. So lässt sich das Rätsel über reines Brute Forcing lösen. Der Code wird zufällig erstellt, deswegen kann keine eindeutige Lösung angegeben werden. Allerdings leuchtet die Lampe grün, wenn eine Zahl an der richtigen Position eingegeben wurde.

Light-Minigame

Anschließend betritt man das Labor des verschwundenen Professors. Hier ist es allerding noch dunkel und der Stromkasten muss repariert werden.

Dazu müssen die logischen Ausdrücke aufgelöst werden.

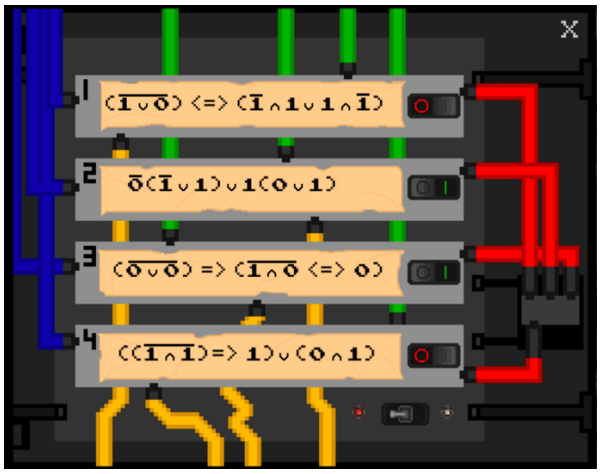
Das Minigame öffnet sich, wenn man auf den Stromkasten klickt.

Lösung der Ausdrücke

1. Ausdruck: wahr → Schalter auf 1
2. Ausdruck: wahr → Schalter auf 1
3. Ausdruck: falsch → Schalter auf 0
4. Ausdruck: wahr → Schalter auf 1

Die Reihenfolge der Ausdrücke variiert dabei. Um die Schalterstellungen zu validieren muss der Strom, durch Klicken auf den Switch, eingeschaltet werden.

Hat man dies geschafft, geht das Licht an und man kann sich auf die Suche nach dem Impfstoff machen

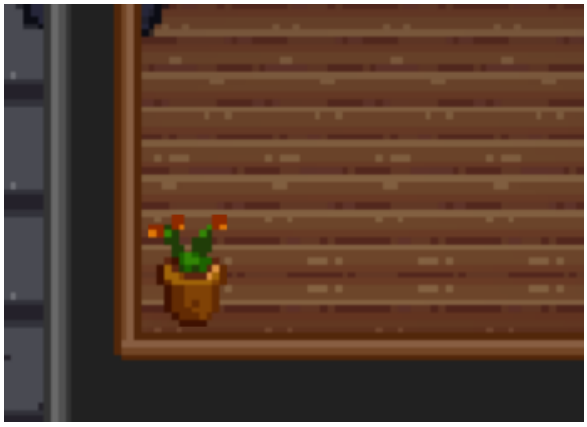


Doorlock Minigame

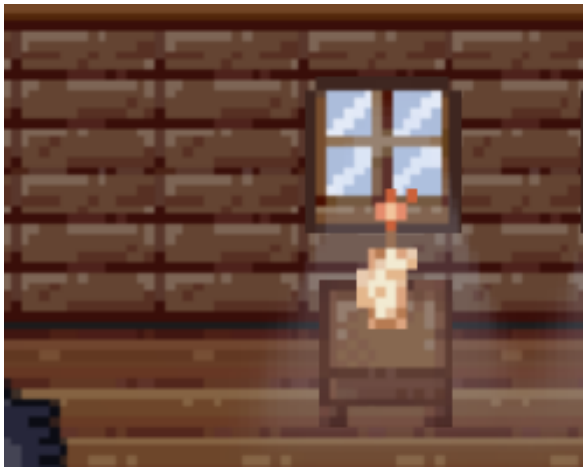
Um die Tür dieses Raumes zu öffnen, muss man zunächst vier verschiedene Schlüssel im Raum finden (Kreuz, Herz, Pik und Karo)

Diese sind an den folgenden Orten versteckt.

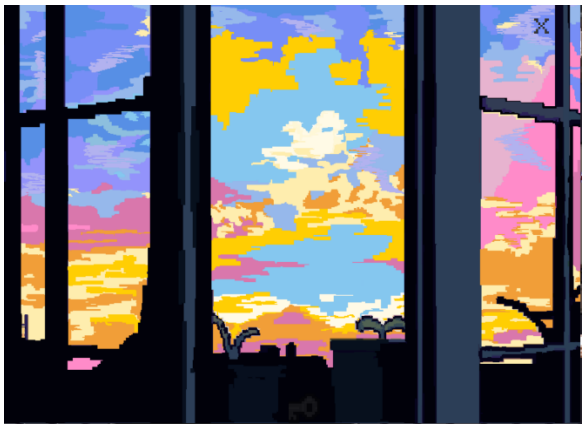
Doppelklick auf die Pflanze nötig, um den Schlüssel zu sammeln.



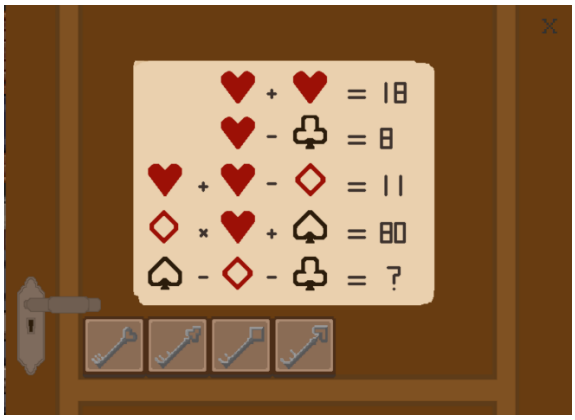
Für diesen Schlüssel muss man zunächst einen Blick aus dem Fenster werfen.



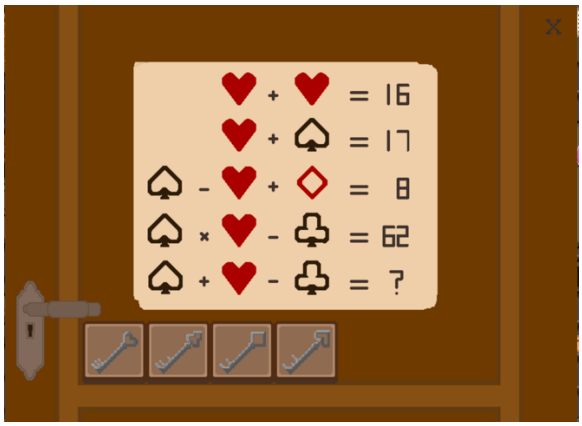
Hier am unteren Rand zwischen den Pflanzen befindet sich dann der Schlüssel



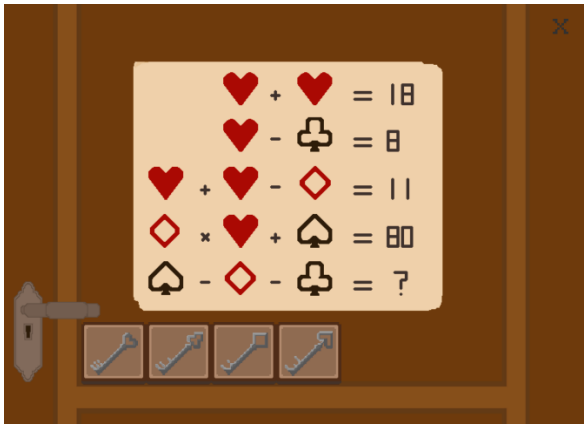
Anschließend, muss man das mathematische Rätsel lösen und den Schlüssel mit dem richtigen Symbol auswählen, um die Tür zu öffnen.



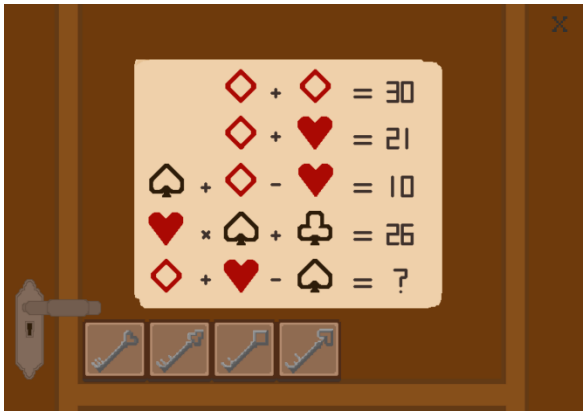
Lösungsschlüssel: Key-Karo



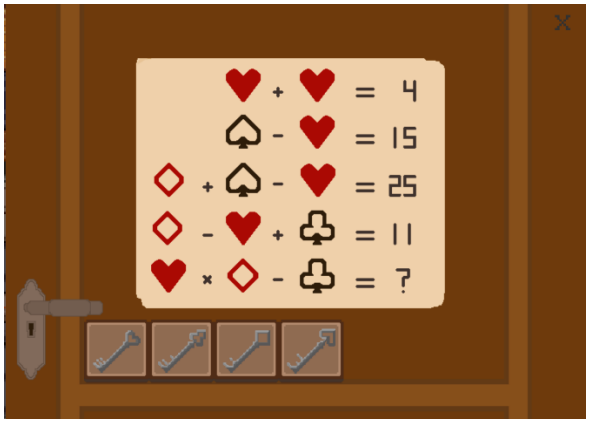
Lösungsschlüssel: Key-Heart



Lösungsschlüssel: Key-Cross



Lösungsschlüssel: Key-Peek



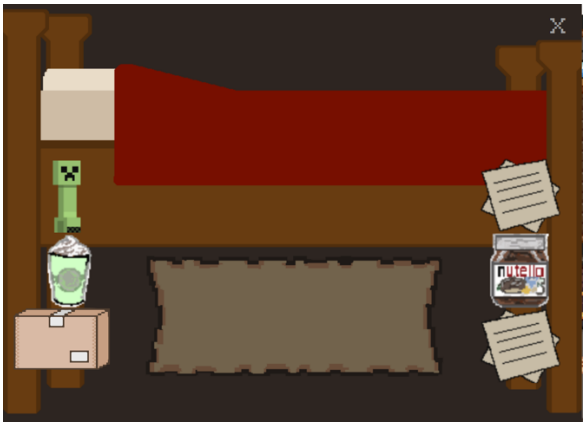
Nun kann man den zweiten Raum betreten.

Tidy-Up-Minigame

Dieses Minigame öffnet sich, wenn man das Bett im zweiten Raum (dem Wohnbereich anklickt)

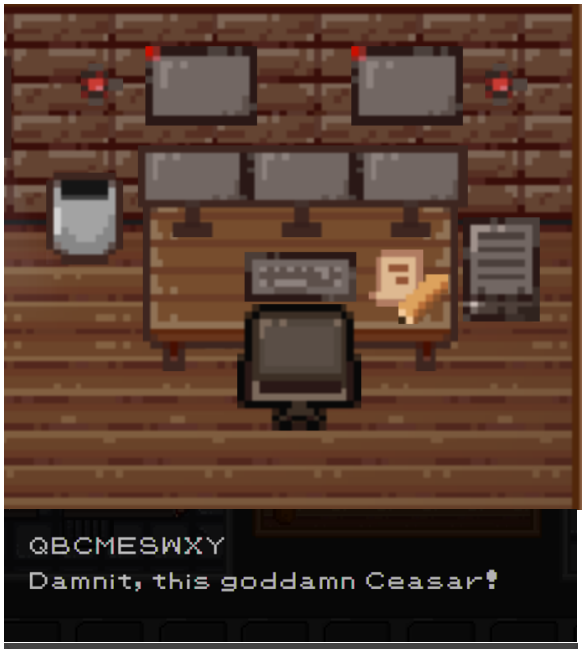
Das Ziel des Minigames ist es, das Chaos unter dem Bett des Professors zu beseitigen. Dazu müssen die einzelnen Objekte angeklickt werden. Hat man alle Objekte “zur Seite” geräumt, erhält man eine Notiz.

Auf dieser befindet sich das verschlüsselte Passwort, um das PC-Minigame zu lösen.



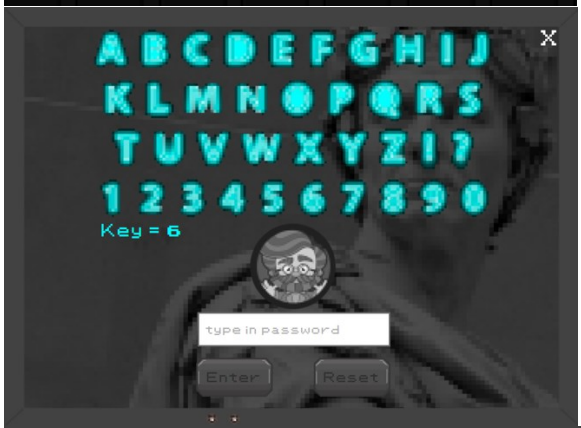
PC Minigame

Um das PC Minigame zu starten muss man auf den Computer im Labor klicken

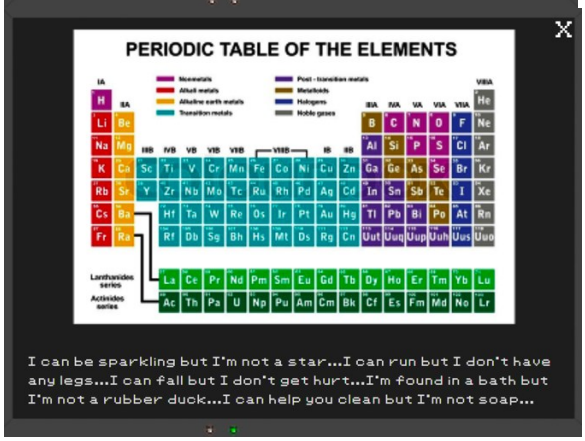


Das erhaltene PW wird durch Anklicken des Items im Inventar angezeigt.

Es muss mit der Ceasar Cipher Key = 6 entschlüsselt werden. Das Passwort lautet whisky123. Dabei ist es egal, ob man whisky123, Whisky123 oder WhISKy123 eingibt.

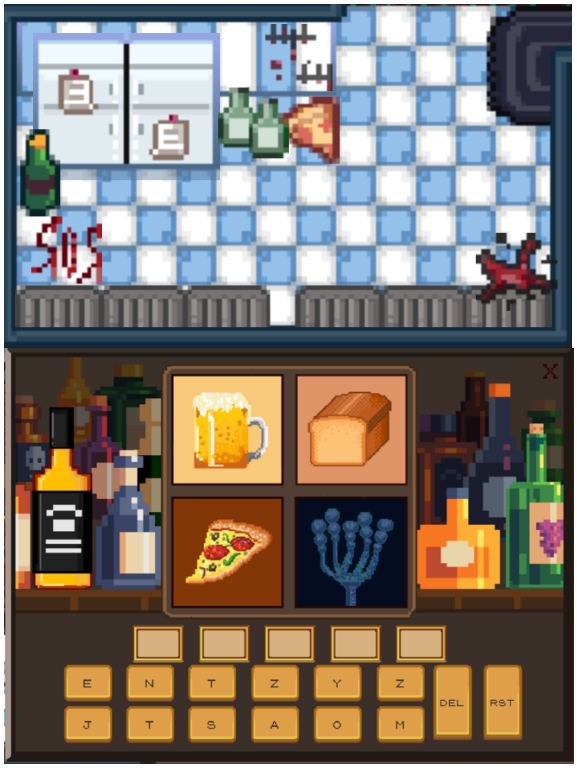


Anschließend muss zweimal auf das Element H und einmal auf das Element O geklickt werden um das Wasser Item (H2O) zu erhalten.



4-Images Minigame

Dieses Minigame öffnet sich, sobald der Kühlschrank im zweiten Raum (livingroom) angeklickt wird und auf die Whiskyflasche geklickt wurde.



Es funktioniert wie das Spiel “4 Bilder 1 Wort” und die Lösung ist “Yeast” oder “Hefe” zu deutsch. Sobald das Minigame gelöst wird, erhält der Spieler die Hefe, die essenziell für den Impfstoff ist.

Table-End-Minigame

Für das Table End Minigame müssen vier Notizen gefunden werden. Auf ihnen steht das Korrekte Mischverhältnis aller Zutaten.

Truhe anklicken (Wohnbereich links neben Kamin)

Ganz rechter Schacht im Raum

Schließlich fehlen noch Zucker und Nacl. Nacl wird im zweiten Raum (Wohnbereich) des Professors gefunden. (Mittleres gefäß)

Zucker wird auf dem großen Metalltisch im Labor gefunden. (das kleine Säckchen)

Anschließend kann es losgehen. Man muss den Metalltisch im Labor anklicken, um das finale Minigame zu starten.

Alle Zutaten müssen entsprechend den Angaben auf den Notizen abgewogen werden. Dazu klickt man ein Item an. Daraufhin erhöht oder verringert man die Menge und klickt auf den confirmation Button rechts neben der Waage.

Sugar: 30
Nacl: 15
HotWater: 35
Yeast: 25

Man erhält das Abgewogene Item.

Zu beachten ist, dass Wasser erst mit dem Bunsenbrenner erhitzt werden muss, um es abwiegen zu können. Dazu wird auf Wasser geklickt und anschließend der Bunsenbrenner mit dem Aschaltknopf eingeschaltet. Man erhält heißes Wasser.

Sobald alle Zutaten in der richtigen Menge vorhanden sind, kann man sie in Reihe auswählen. Dadurch füllt sich der Mixer. Durch Betätigen des Einschalters werden die Zutaten nun gemixt.

Man erhält den Impfstoff und das Spiel wurde erfolgreich beendet.