The Last Package

Spieleentwicklung und Programm

Niklas von Hirschfeld und Jan Knüpfer

Inhalt

Einleitung

In diesem Bericht präsentieren wir, Niklas von Hirschfeld und Jan Knüpfer, unsere Arbeit. Unser Projekt, betitelt "The Last Package", ist das Ergebnis intensiver Planung und Umsetzung.

In diesem Bericht werden wir einen umfassenden Einblick in das Angebot, das Konzept sowie die Dokumentation der Arbeitsabläufe geben. Wir werden auch die Spielanleitung betrachten, die den Spielern eine klare Orientierung bietet, sowie einen Blick auf das Programm werfen, das die Grundlage dieses fesselnden Spielerlebnisses bildet. Abschließend werden wir das Projekt reflektieren und ein Fazit ziehen, das die Erfahrungen und Erkenntnisse unserer Arbeit zusammenfasst.

Sehr geehrter Konrad Dijkstra,

vielen Dank für Ihr Interesse. Gerne unterbreiten wir Ihnen folgendes Angebot.

Angebot

Position	Anzahl	Preis	Einheit	Beschreibung	Steuer	Netto
1	1	368,90€		Konzeptentwicklung und Storyboarding	19%	310,00€
2	1	1856,40€		Spieldesign und Grafiken	19%	1560,00€
3	1	2439,50€		Programmierung und Entwicklung	19%	2050,00€
4	1	476,00€		Musik und Soundeffekte	19%	400,00€
5	1	261,80€		Qualitätssicherung	19%	220,00€
6	1	238,00€		Projektmanagement und Kommunikation	19%	200,00€

Zwischensumme Netto	4740,00€
Umsatzsteuer 19%	900,60€
Gesamtbetrag	**5640,60€**

Wir freuen uns, wenn wir Sie mit unserem Angebot überzeugen können. Bei Fragen dürfen Sie sich gerne auch per E-Mail bei uns melden.

Mit freundlichen Grüßen
Jan Knüpfer und Niklas von Hirschfeld im Namen von Serenity Studios

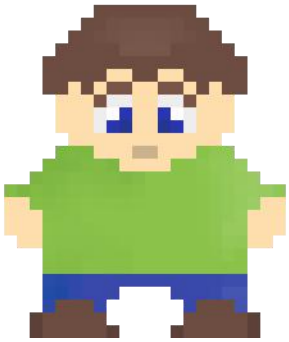
Konzept

"The Last Package" ist ein einzigartiges 8-bit-Spiel, das eine fesselnde Geschichte und ein ansprechendes Gameplay bietet. Das Spiel beginnt an einem gewöhnlichen Morgen, als der Spieler einen mysteriösen Brief durch den Haustürschlitz erhält. Dieser Brief, überbracht vom kürzlich verstorbenen Großvater des Spielers, birgt den letzten Wunsch, ein geheimnisvolles Paket an eine gewisse "Amelia" zu liefern. Die einzigen Anhaltspunkte sind das Ziegelhaus im Dorfzentrum von Nota Village.

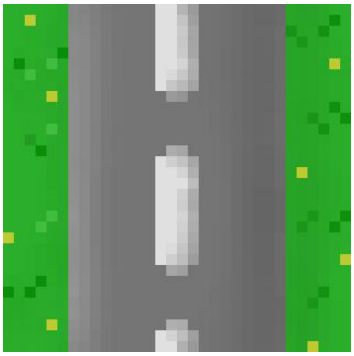
Design und Musik

Beispiele

Charakter Sprite



Straßen Sprite



Natürlich besitzt das Spiel eigene Soundtracks im nostalgischen 8-bit Stil

Klicken sie hier für eine Hörprobe!

Programmierung:

Wir programmieren unser Spiel in Java. Um den Fokus auf das Spiel selber zu legen, aber trotzdem genug Kontrolle über die Grafische Darstellung zu behalten, haben wir uns entschieden OpenGL zu nutzen. Da OpenGL in C geschrieben ist, benötigen wir art Übersetzung. Diese wird von der Java Bibliothek LWJGL bereitgestellt.
Wir planen das Spiel Modular zu gestalten, so das neue Features und

Serenity Studios

Dokumentation der Arbeitsschritte

1. Konzept:

Entwicklungsziel: Unser Ziel war es, ein einzigartiges 8-Bit-Spiel zu entwickeln, das die Spieler mit einer fesselnden Geschichte und einem ansprechenden Gameplay begeistert.

Erstellung einer Firmenidentität: Wir „gründeten“ das Spiele Studio „Serenity Studios“, welches sich mit Indie-Games, insbesondere 8-bit Spielen beschäftigt. Das Logo soll mit der Orange-Gelben Farbe und der Symbolik der Sonne eine beruhigende, trotzdem seriöse Wirkung ausstrahlen. Erstellt wurde dies mit Adobe Illustrator 2023.

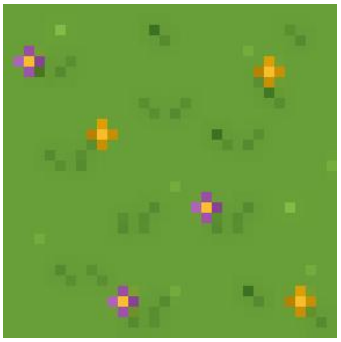
2. Grafik

Die Grafik des Spieles ist gehalten im klassischen 8-bit Stil. Auflösung ist 192px x 128px, wobei es auf die passende Bildschirmgröße hochskaliert wird. Gearbeitet haben wir mit dem Programm Aseprite, was ein Programm für Pixelart und derer Animationen ist.

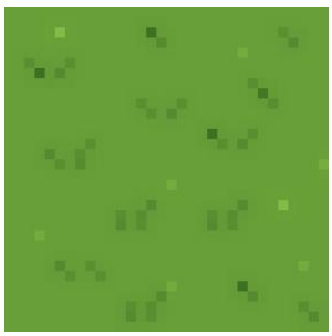
Beispiele:

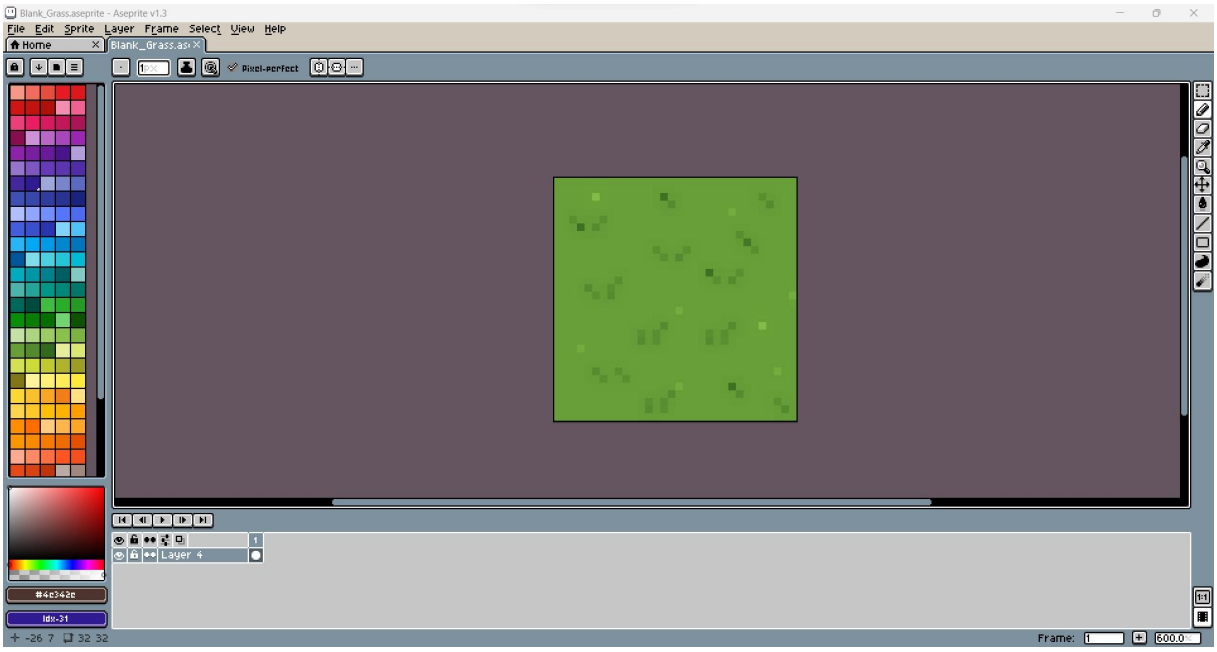
Haus von Pat und Amelia

Erstellung von Pat und seinen Animationen



Gras und Blumenfeldtexturen

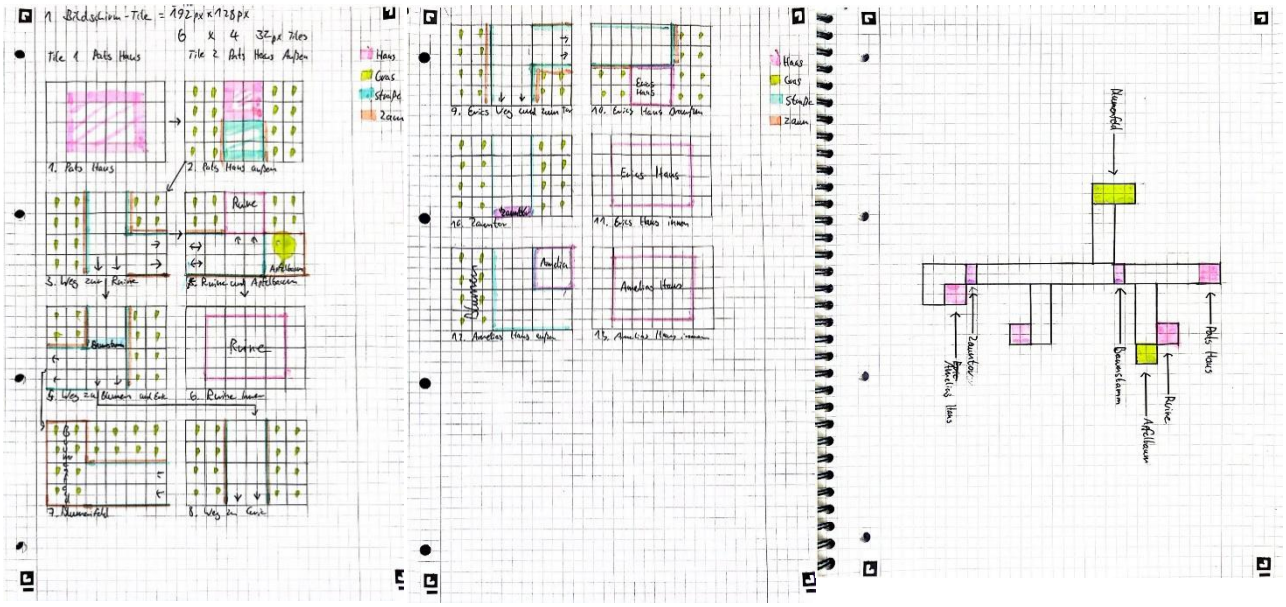




Interface von Aseprite

3. Level-Design:

Dorf Nota: Das Level-Design umfasst die Gestaltung eines lebendigen Dorfes mit verschiedenen Bereichen und Gebäuden, die erkundet werden können. Jeder Bereich wird seine eigenen Geheimnisse und Hindernisse haben, die es zu überwinden gilt. Zur Vereinfachung der Implementierung in Java, zeichneten wir das Level-Design als Entwurf per Hand.



4. Gameplay

Controlls

Taste	Action
w,a,s,d	Bewegung Spieler 1
Tab	Mögliche Interaktionen durchlaufen
e	Ausgewählte Aktion ausführen

5. Editor

Der Editor bietet eine umfangreiche Oberfläche um die Level zu gestalten und Objekte zu bearbeiten. Der Modulare aufbau im Code wird hier aufgegriffen und es wird ermöglicht dynamisch Objekte hinzuzufügen. Die hier verwendete Libary für die grafische darstellung *Dear ImGUI* steuert dem bei, in dem sie die manuelle veränderung von Variablen zur Laufzeit ermöglicht.

Der Editor besteht grundsätzlich aus mehreren Fenstern. Diese lassen sich beliebig verschieben und anorden. In einem Fenster wird die Spiel-Kamera wieder gegeben. Dies wird duch OpenGL ermöglich. OpenGL kann eine Scene in einen so gennanten Framebuffer Rendern. Dieser kann damit als Textur für das Fenster in ImGUI verwendet werden.

Controlls

Taste	Aktion
Ctrl - d	Duplizieren
r	um 90 Grad rotieren
Rücktaste	löschen
Bild-Auf	z-Index erhöhen
Bild-Runter	z-Index erniedriegen
Pfeil-Tasten	Um eine Zelle bewegen
Shift + Pfeil-Tasten	Um 0.1 Zellen bewegen
E	Gizmos auf Scalieren stellen
M	Gizmos auf Bewegen stellen

> Alle Aktionen beziehen sich, wenn nicht explizit angegeben, auf das aktive Objekt, oder eine Auswahl.

Maus	Action
Klicken und Ziehen	Mehrere Objekte bewegen
Klicken	Aktives Objekt auswählen

Gizmo

Die Gizmos sind Komponente des LevelEditor Stuff GameObjekt. Sie erscheinen, wenn ein Objekt ausgewählt wird und ermöglichen das verändern der Größe und der Position mit der Maus. Sie bestehen aus zwei Pfeilen, einer für jede Achse in den zwei Dimensionen.

Fenster:

LevelEditor Stuff

LevelEditor Stuff ist schlussendlich auch ein GameObjekt. Hier werden allgemeine Variablen festgelegt. Auch die “Gizmos” können hier bearbeitet werden.

Scene Hierarchy

Hier werden alle GameObjekts der aktuellen Scene aufgelistet. Eine Hierarchie unter den GameObjects ist noch nicht Implementiert, aber geplant. Eine solche ermöglicht es ganze Gruppen anzusteueren und die Scene besser im Überblick zu behalten.

Objects

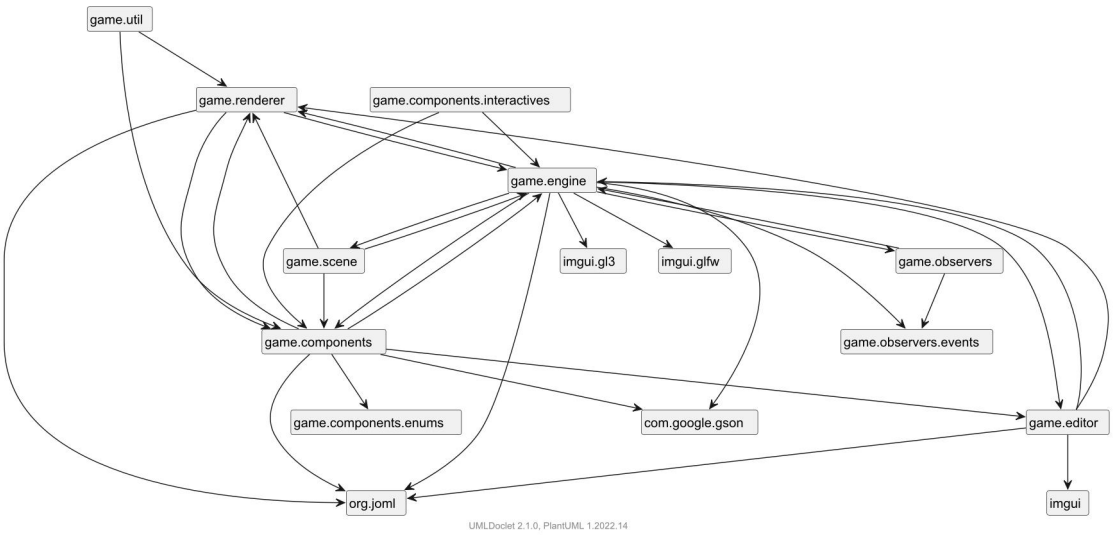
Hier wird das aktuell ausgewählte *SpriteSheet* dargestellt und es wird ermöglicht die Texturen als eigene GameObjects der Scene hinzuzufügen. Diese haben standardmäßig keine erweiternden Komponenten, solche können aber nach belieben hinzugefügt werden.

Prefabs sind voreingestellte GameObjecte. Diese werden im Code frei erstellt und können im nachhinein ebenfalls frei bearbeitet werden. Prefabs ersparen viel arbeit, da sie iterative Prozesse erleichtern.

Game Viewport

Hier wird das Spiel selber angezeigt werden. Dabei gibt es zwei Modis zwischen denen man hin und her wechseln kann. Den Bearbeitungsmodus, bei dem das Spiel pausiert ist und diverse Debuging-Tools angezeigt werden können, und dem Spiel-Modus, bei welchem das Spiel ausgeführ wird.

Umsetzung in Java



Abhängigkeiten der Packete untereinander

Die automatisch generierten JavaDOCs mit erweiterter grafischer Darstellung durch plantuml.com sind auf https://vonhirschfeld.eu/lieferfix/javadoc/ (Zuletzt aufgerufen am 2024-03-14)

Der Code ist Modular aufgebaut, mit dem Ziel, neue Features einfacher zu implementiern und eine saubere Strucktur durchzusetzen. Für einen solchen Modularen aufbau ist eine Objekt Orientierte Programmiersprache ebenfalls hilfreich.

Packete:

Um einen Überblick zu behalten ist es wichtig, die Klassen semantisch zu Strukturieren. Viele Programmiersprachen bieten dafür packete an. Diese sind auch “Namespaces” vergleichbar.

Components

Hier sind all Komponenten für die GameObjects declariert. Dafür wurde in “Component.java” eine abstract Klasse definiert, welche die anderen jeweils erweitern. Dadurch ist gewährleistet, das alle Componenten kompatiebel sind und benötigte Funktionen vorhanden sind. Eine Auswahl dieser ist folgende:

- AnimationState
- Component
- ComponentDeserializer
- EditorCamera
- GameCamera
- GizmoSystem
- GridLines
- KeyControls
- MouseControls
- PlayerController
- Rigidbody
- ScaleGizmo
- Sprite
- StateMachine
- TranslateGizmo

Editor

In diesem Packet wird ImGUI aufgesetzt und alles was nötig ist, um den Editor zum Laufen zu bringen.

Engine

Hier lebt das Herz des Spieles. Die grundlegendsden Automtisierungen finden hier statt. Neben den Maus- und Key-Listener werden hier auch die Objekte serialisiert.

Observer

Obverser tragen das EventSystem. In desem können diverse Objekte als Observer deklariert werden, welche dann auf Events, als Enums, reagieren können.

Renderer

OpenGL war und ist ein standard für die Interaktion mit der GPU. Dabei werden direkt Verticies in eine Speicher der GPU kopiert um dort von Shadern verarbeitet zu werden. Im

prinzip gibt es zwei Shader, welche programmiert werden müssen. Einemal den Vertex-Shader und den Fragmen-Shader. Der Vertex-Shader verarbeitet die Verticies und gibt informationen an den Fragment-Shader weiter. Diese iteriert über jeden Pixel auf dem Bildschirm und legt seine Farbe fest.

Util

In Utils werden einiege Einstellungen und Hilfefunktionen bereitgestellt.

Bibliotheken

LWJGL

- Liezens: BSD-3-Clause
- Link: https://www.lwjgl.org/

“Leight Weight Java Game Libary” bietet Zugang zu nativen APIs, welche bei der Entwicklung von Grafik- (OpenGL, Vulkan), Audio- (OpenAL) und Parallel-Computin-Anwendungen (OpenCL) notwendig sind. Es besteht aus sogenannten “Bindings” für die Jeweiligen Anwendungen. Die ursprünglichen APIs sind in diesem fall nicht in Java geschrieben und werden so zu sagen übersetzt.

Der Quellcode ist unter der BSD-3-Clause Liezens offen verfügbar und wird von einem Kollektive und Freiwilligen aktive erweitert und aktualisiert.

Zu dem ist LWJGL modular aufgebaut. Man kann das benutzen, was man braucht. Bei der Entwicklung unseres Spieles brauchen wir folgende Module:

GLFW

- Liezens: Zlib license
- Link: https://www.glfw.org/
- Entwickelt in: c

GLFW (Graphics Library Framework) ist eine Multi-Plattform Bibliothek für OpenGL, OpenGL ES und Vulkan. Sie erlaubt es über eine simplere API Fenster, Kontexte, Oberflächen, Input und Events zu Managen. Über diese Bibliothek werden wir in erster Linie das Fenster an sich und den Input verarbeiten.

OpenGL

- Liezens: Eigene, erlaubt es die Software zu verändern und zu verteilen (Mit Bedingungen)
- Link: https://www.khronos.org/opengl/
- Entwickelt in: c

OpenGL® ist die am weitesten verbreitete 2D- und 3D-Grafik-API der Branche, die Tausende von Anwendungen auf einer Vielzahl von Computerplattformen ermöglicht. Sie ist unabhängig von Fenstersystemen und Betriebssystemen sowie netzwerktransparent. OpenGL ermöglicht es Entwicklern von Software für PC-, Workstation- und Supercomputing-Hardware, hochleistungsfähige und visuell ansprechende Grafiksoftwareanwendungen für Märkte wie CAD, Inhaltserstellung, Energie, Unterhaltung,

Spieleentwicklung, Fertigung, Medizin und virtuelle Realität zu erstellen. OpenGL stellt alle Funktionen der neuesten Grafikhardware zur Verfügung.

Über OpenGL können wir direkt mir der GPU Kommunizieren, Texturen speicher und mit eigenen Shadern beeinflussen, wie die Grafikkarte etwas verarbeitet. Gerade diese Shader, welche in einer eigenen Sprache geschrieben sind, erweitern die Möglichkeiten, auch enorm für PixelGrafik. So können wir zum Beispiel dynamisches Licht gestalten und vieles mehr. Essenziell ist unter anderem die Möglichkeit “Vertex” Buffer direkt an die GPU zu senden und so optimal wie möglich die Oberfläche zu gestalten.

OpenGL wird unter anderem aktive weiterentwickelt von: Google, Nvidia, Apple, AMD, Huawei, Intel, …

OpenAL

- Liezens: GNU LIBRARY GENERAL PUBLIC LICENSE
- Link: Aktuellster Fork: https://github.com/kcat/openal-soft
- Entwickelt in: c++

OpenAL ist eine plattformübergreifende API für Audio, die hauptsächlich für die Wiedergabe von 3D-Sound und räumlichen Audioeffekten verwendet wird. Es ermöglicht Spieleentwicklern, realistischen Klang in ihren Spielen zu erzeugen, indem sie Klangeffekte basierend auf der Position und Bewegung von Objekten im Spielumfeld platzieren können. OpenAL bietet eine einfache Möglichkeit, komplexe Audioeffekte zu erzeugen und bietet Unterstützung für verschiedene Audioformate und -geräte.

Wir werden uns in erster Linie darauf Konzentrieren, dass wir überhaupt Sound haben und auch dafür ist OpenAL ideal.

GSON

- Entwickelt von: Google
- Liezens: Apache-2.0 license
- Link: https://github.com/google/gson
- Entwickelt in: Java

Gson ist eine Java-Bibliothek, die verwendet werden kann, um Java-Objekte in ihre JSON-Darstellung zu konvertieren und wieder zurück. Dies ermöglicht es uns Level, Szenen, Entitäten und Items extern zu speichern und dynamisch zu Laden. Wir planen auch den Spielstand und viele andere Daten so zu Speichern und zugänglich zu machen. Da wir uns bemühen unser Spiel flexibel zu gestalten nimmt uns GSON sehr viel Arbeit und Mühe ab.

Dear Im GUI

- Liezens: MIT license
- Link: https://github.com/ocornut/imgui
- Entwickelt in: C++

Dear ImGui ist eine leichte Bibliothek für grafische Benutzeroberflächen in C++. Sie gibt optimierte Vertex-Buffer aus, die jederzeit in die 3D-Pipeline-fähigen Anwendungen gerendert werden können. Sie ist schnell, portabel, Renderer-unabhängig und in sich geschlossen (keine externen Abhängigkeiten).

Diese Bibliothek ermöglicht es uns und auch später ihnen Level individuell zu gestalten und zu formen. Es ermöglicht uns während der Laufzeit Variablen zu verändern und so ein optimales Spielerlebnis für den Spielenden vorzubereiten.

JOML

- Liezens: MIT license
- Link: https://joml-ci.github.io/JOML/
- Entwickelt in: Java

Eine einfache Bibliothek die Operationen der linearen Algebra vereinfachen. Diese sind zum verarbeitet von Buffern für OpenGL notwendig.

Java

Dies sind Java eigenen Pakete die wir verwenden.

Weiter Informationen zu diesen Paketen finden Sie unter:
https://docs.oracle.com/en/java/javase/22/docs/api/index.html

io

- File: um zu checken, ob ein Datei existiert
- IOException: Scheitern von Lese- und Schreiboperationen während der Laufzeit signalisieren

nio

Definiert Puffer, die Container für Daten sind.

- ByteBuffer
- FloatBuffer
- IntBuffer
- files.Files
- file.Paths

util

- ArrayList
- Collections
- HashMap
- List - Map
- Vector

Umsetzung im JavaEditor

Wie man das Spiel auf einem Windows-System zum laufen bringt.

1. Projekt herunterladen und ggf. entpacken (Falls als ZIP heruntergeladen)
2. JavaEditor öffnen
3. Folgende datei im JE öffnen: {Projektordner}/src/main/java/game/App.java
4. Benötigte Einstellungen setzen
5. Kompilieren
6. Ausführen

Benötigte Einstellungen:

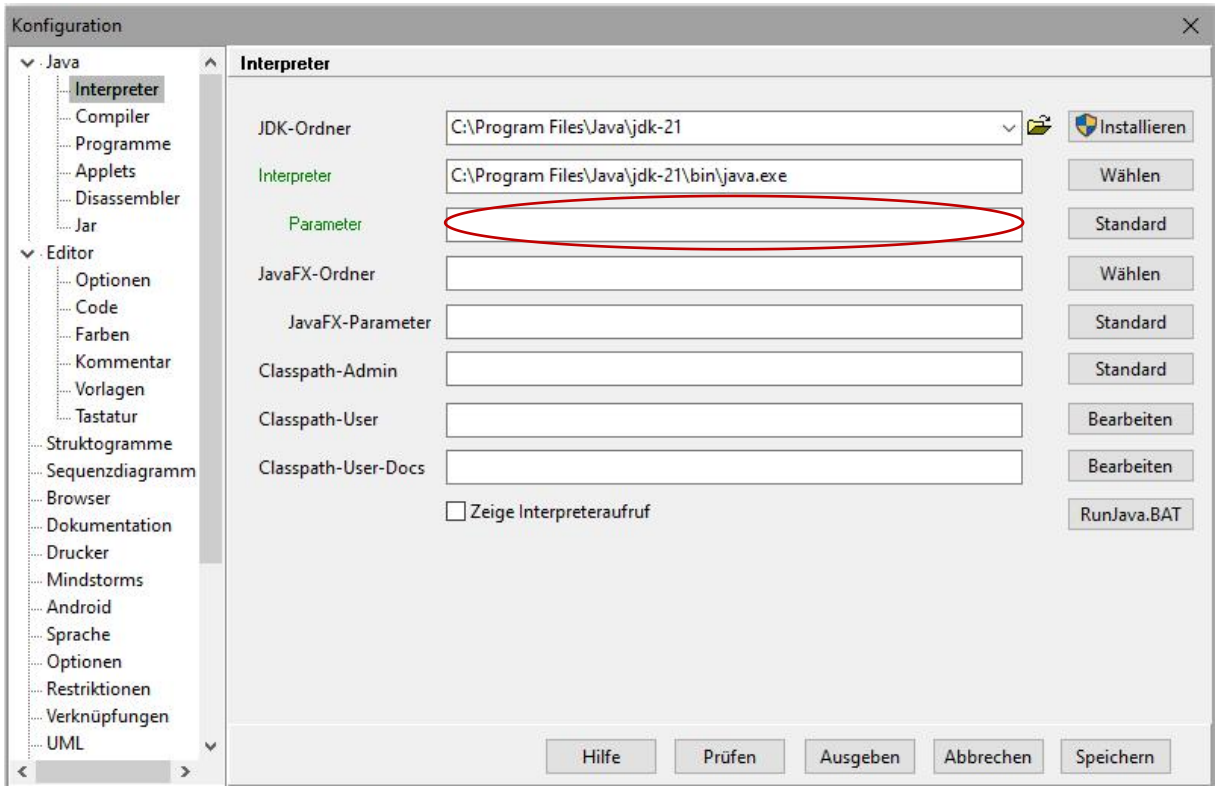
Die Einstellungen sind unter: *Window -> Configuration*; zu finden

Bei den Parametern den absoluten Pfad zum Projektordner angeben, und vor dem Pfad:
„*-Duser.dir=*“ setzen.

Beispiel:

_____

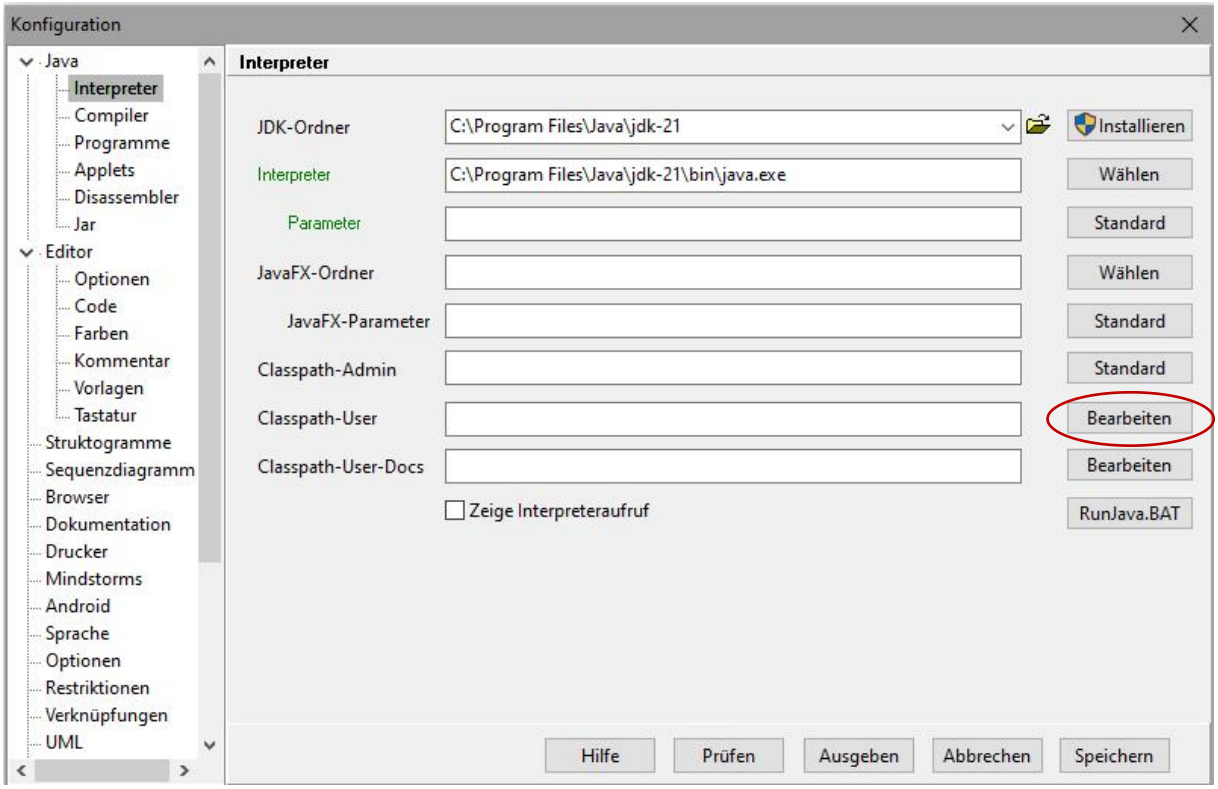
-Duser.dir=C:\Users\{Benutzer}\{Pfad zum Projekt}

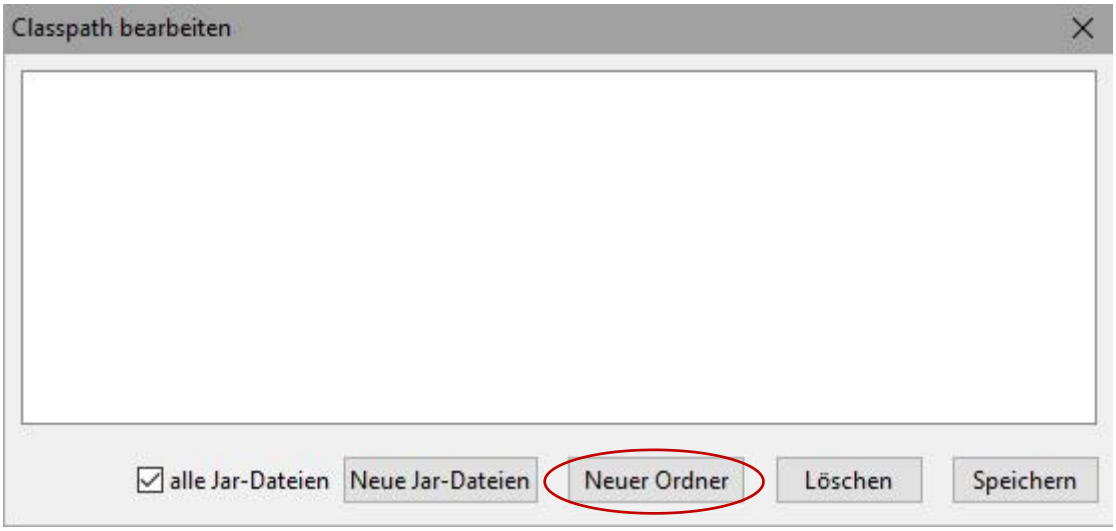


_____

Dann unter: *Java -> Interpreter -> Classpath user*

auf *Edit* klicken, in dem neuen kleinen Fenster nun auf *New Folder* klicken und dann in den Ordner "*lib*" in dem Projektordner auswählen und hinzufügen. Nach dem Hinzufügen sollte der Pfad in dem kleinen Fenster angezeigt werden und die blaue Checkbox links daneben sollte ebenfalls ausgewählt sein.





Wichtig: Es muss in dem Fenster auf den Pfad geklickt werden, um unten links en dem Fenster die Option "*all jar files*" zu aktivieren. Die ist notwendig, damit alle ".jar" Dateien mit eingebunden werden.

Arbeitsaufteilung

Programmierung der Gameengine:	Niklas
Pixelart der GUI und Nebenfiguren:	Niklas
Implementierung des Pixelarts:	Niklas
Konzeptionierung und Design:	Jan
Pixelart der Welt und von Hauptfigur:	Jan
Programmierung von der Kamera und Rigidboxen:	Jan

Fazit

In diesem Informatikprojekt zur Spieleentwicklung mit Java konnten wir uns nicht nur in die Tiefen der Programmierung, sondern auch in die grafische Entwicklung vertiefen. Java präsentierte sich zwar als eine herausfordernde Wahl, besonders im Hinblick auf die grafischen Aspekte, was auch OpenGL nicht besser gemacht hat, dennoch war der Prozess insgesamt eine überaus lehrreiche und unterhaltsame Erfahrung. Die zeitliche Begrenzung des Projekts war zwar spürbar, führte aber auch zu immer kreativer werdenden Lösungsansätzen. Unsere ursprüngliche Vision einer klaren Trennung zwischen der Game Engine und dem eigentlichen Spiel konnte aufgrund des Zeitdrucks nicht vollständig realisiert werden. Die Grenzen verschwammen zunehmend.

Durch die effektive Aufteilung der Arbeit konnten wir beide unsere individuellen Stärken einbringen. Während einer von uns sich verstärkt auf die Programmierlogik konzentrierte, widmete sich der andere intensiver den grafischen Elementen des Spiels. Diese synergetische Zusammenarbeit ermöglichte es uns, nicht nur voneinander zu lernen, sondern auch unsere Fähigkeiten in neuen Bereichen zu erweitern. Insbesondere die grafische Entwicklung war eine facettenreiche Herausforderung, die uns dazu brachte, Java und seine Möglichkeiten in diesem Bereich tiefer zu erkunden.

Abschließend war dieses Projekt trotz einiger Hürden, insbesondere in Bezug auf die grafische Entwicklung und die Handhabung von Java, eine wertvolle Erfahrung. Es hat nicht nur unser technisches Know-how, sondern auch unsere Fähigkeit zur Teamarbeit und kreativen Problemlösung gestärkt. Die Erfahrungen aus diesem Projekt haben unsere Begeisterung für die Spieleentwicklung bestärkt und uns wichtige Lektionen für zukünftige Projekte mit auf den Weg gegeben. Die Mischung aus technischer Herausforderung und kreativer Gestaltung in der Spieleentwicklung fasziniert uns weiterhin und motiviert uns, in diesem Bereich weiter zu lernen und zu wachsen.