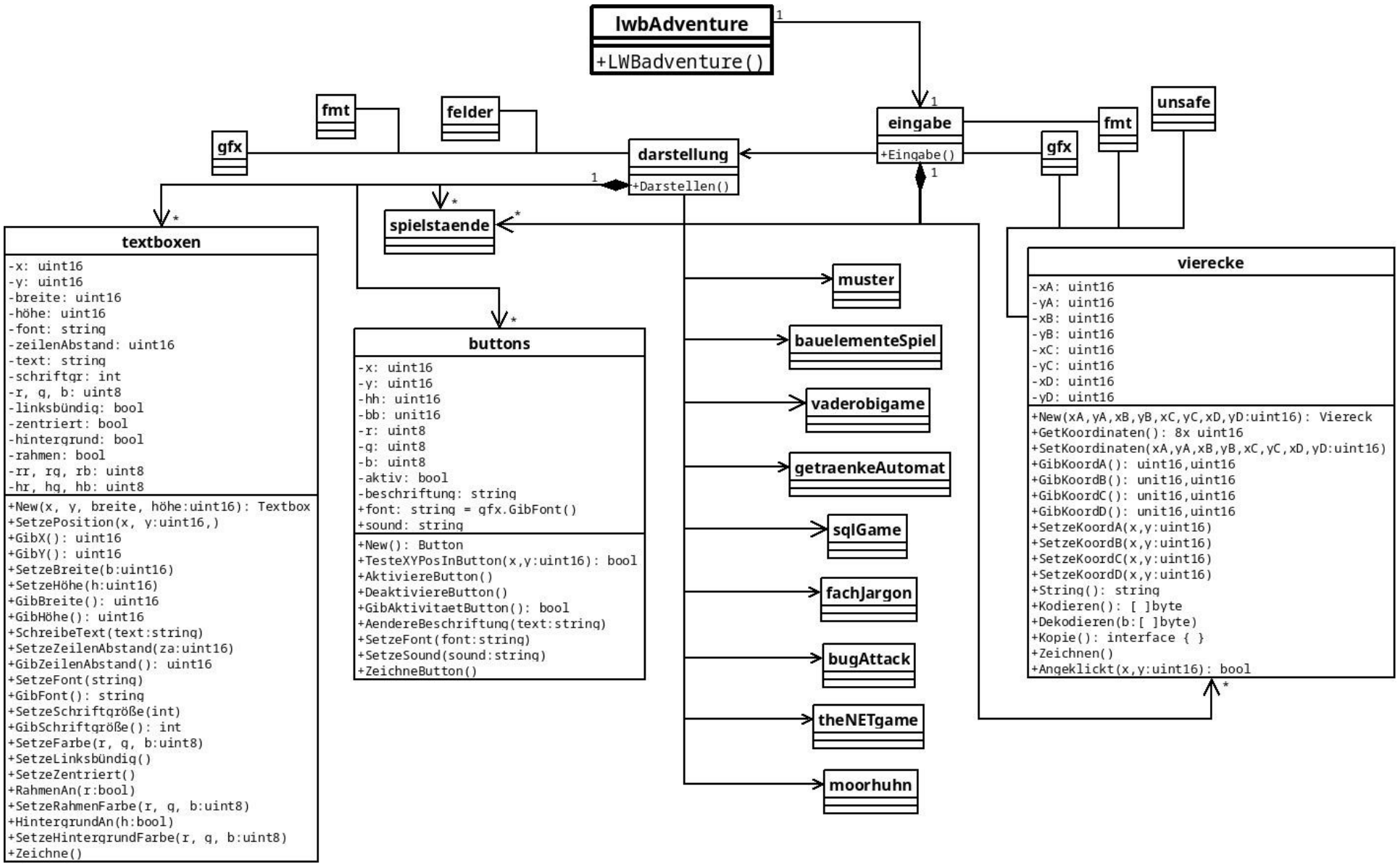

lwbAdventure
+LWBadventure()
fmt
felder
gfx
darstellung
+Darstellen()
eingabe
+Eingabe()
fmt
unsafe
gfx
textboxen
-x: uint16
-y: uint16
-breite: uint16
-höhe: uint16
-font: string
-zeilenAbstand: uint16
-text: string
-schriftgr: int
-r, g, b: uint8
-linksbündig: bool
-zentriert: bool
-hintergrund: bool
-rahmen: bool
-rr, rg, rb: uint8
-hr, hg, hb: uint8
+New(x, y, breite, höhe:uint16): Textbox
+SetzePosition(x, y:uint16,)
+GibX(): uint16
+GibY(): uint16
+SetzeBreite(b:uint16)
+SetzeHöhe(h:uint16)
+GibBreite(): uint16
+GibHöhe(): uint16
+SchreibeText(text:string)
+SetzeZeilenAbstand(za:uint16)
+GibZeilenAbstand(): uint16
+SetzeFont(string)
+GibFont(): string
+SetzeSchriftgröße(int)
+GibSchriftgröße(): int
+SetzeFarbe(r, g, b:uint8)
+SetzeLinksbündig()
+SetzeZentriert()
+RahmenAn(r:bool)
+SetzeRahmenFarbe(r, g, b:uint8)
+HintergrundAn(h:bool)
+SetzeHintergrundFarbe(r, g, b:uint8)
+Zeichne()
spielstaende
buttons
-x: uint16
-y: uint16
-hh: uint16
-bb: unit16
-r: uint8
-g: uint8
-b: uint8
-aktiv: bool
-beschriftung: string
+font: string = gfx.GibFont()
+sound: string
+New(): Button
+TesteXYPosInButton(x,y:uint16): bool
+AktiviereButton()
+DeaktiviereButton()
+GibAktivitaetButton(): bool
+AendereBeschriftung(text:string)
+SetzeFont(font:string)
+SetzeSound(sound:string)
+ZeichneButton()
muster
bauelementeSpiel
vaderobigame
getraenkeAutomat
sqlGame
fachJargon
bugAttack
theNETgame
moorhuhn
vierecke
-xA: uint16
-yA: uint16
-xB: uint16
-yB: uint16
-xC: uint16
-yC: uint16
-xD: uint16
-yD: uint16
+New(xA,yA,xB,yB,xC,yC,xD,yD:uint16): Viereck
+GetKoordinaten(): 8x uint16
+SetKoordinaten(xA,yA,xB,yB,xC,yC,xD,yD:uint16)
+GibKoordA(): uint16,uint16
+GibKoordB(): unit16,uint16
+GibKoordC(): unit16,uint16
+GibKoordD(): unit16,uint16
+SetzeKoordA(x,y:uint16)
+SetzeKoordB(x,y:uint16)
+SetzeKoordC(x,y:uint16)
+SetzeKoordD(x,y:uint16)
+String(): string
+Kodieren(): [ ]byte
+Dekodieren(b:[ ]byte)
+Kopie(): interface { }
+Zeichnen()
+Angeklickt(x,y:uint16): bool
1
1
1
1
1
*
*
*
*
*
*

Designentscheidungen für das Minigame *Muster-Spiel*

Das Minigame Muster-Spiel ist in das Paket **„muster“ ausgelagert** und exportiert nur die Hauptfunktion, die den Spielaufbau und -ablauf steuert. Alle weiteren Funktionen werden so vor dem Hauptprogramm verborgen, sodass keine Kollisionen entstehen. Die Rückgabe erfolgt als *punkte* (uint32) und *note* (float32) zurück an das Hauptspiel.

Innerhalb des Spieles existieren **neben der Hauptroutine** weitere **vier nebenläufige Routinen**, die im Folgenden näher erläutert werden. Dabei übernimmt die Hauptroutine selbst die Rolle der **Tastaturabfrage** nachdem die übrigen Routinen gestartet sind. Damit gewährleistet ist, dass die nebenläufigen Routinen nach Beenden der Hauptroutine nicht im Hauptprogramm weiterlaufen, wurde eine Wait-Group eingerichtet. Die Kommunikation zwischen den Routinen geschieht über geteilte Adressen von, vor allem Bool-Werten und einen gemeinsamen Channel.

Muster-Spiel nutzt die sehr umfangreiche Klasse **„objekte“** für klickbare, angezeigte Elemente und lagert komplexere Texte und Text-Arrays/Slices in der Klasse **„texte“** aus. Für den gegenseitigen Ausschluss, vor allem beim Zeichnen aller Objekte, werden Mutexe aus der Klasse **„sync“** genutzt.

Verwendete Routinen:

- **Tastaturabfrage:** Go-Hauptroutine als for-Schleife, die über das ganze Spiel hinweg auf Eingaben wartend läuft, jedoch nur in den Eingabe-Teilen der Spiele freigeschaltet ist (über den bool *tastatur*).

- **view_komponente:** Die nebenläufige Go-Routine, die alleine für die visuelle Darstellung zuständig ist. Sie zeichnet, also aktualisiert ständig die Maus, die als Bild angezeigt wird. Alle anderen Elemente, die auf den Hintergrund gezeichnet sind werden aus Performance-Gründen einmalig archiviert und ständig restauriert. Nur wenn neue Objekte erscheinen oder verschwinden wird der Hintergrund einmalig im Hintergrund (gfx-Update) gezeichnet und wieder archiviert. Die for-Schleife wird durch adaptive Verzögerung auf etwa 100 FPS gehalten.

- **spielablauf:** Die nebenläufige Go-Routine, die alle Spielelteile und -funktionen zeitlich plant und nacheinander einsetzen lässt. Über Bools und den Channel erhält die Routine die Signale zum fortschreiten in folgende Abschnitte von hauptsächlich der Maussteuerung. Im spielablauf werden alle Objekte erstellt und damit im geteilten Objekt-Slice (*obj) für View-Komponente und Maussteuerung verfügbar gemacht, sodass Anzeige und Interaktion ermöglicht werden.

- **maussteuerung:** Die nebenläufige Go-Routine, welche die Maus-Koordinaten ständig aktualisiert (und an restliche Routinen weitergibt) und nach Maus-Klicks auf Treffer von, an diese Routine übergebenen, Objekten überprüft und entsprechend Signale an restliche Routinen sendet bzw. Sounds abspielt.

- **musikhintergrund:** Die nebenläufige Go-Routine, welche für die Hintergrund-Musik einen Loop eines Musikausschnitts abspielt, bis die anderen Routinen beendet werden.

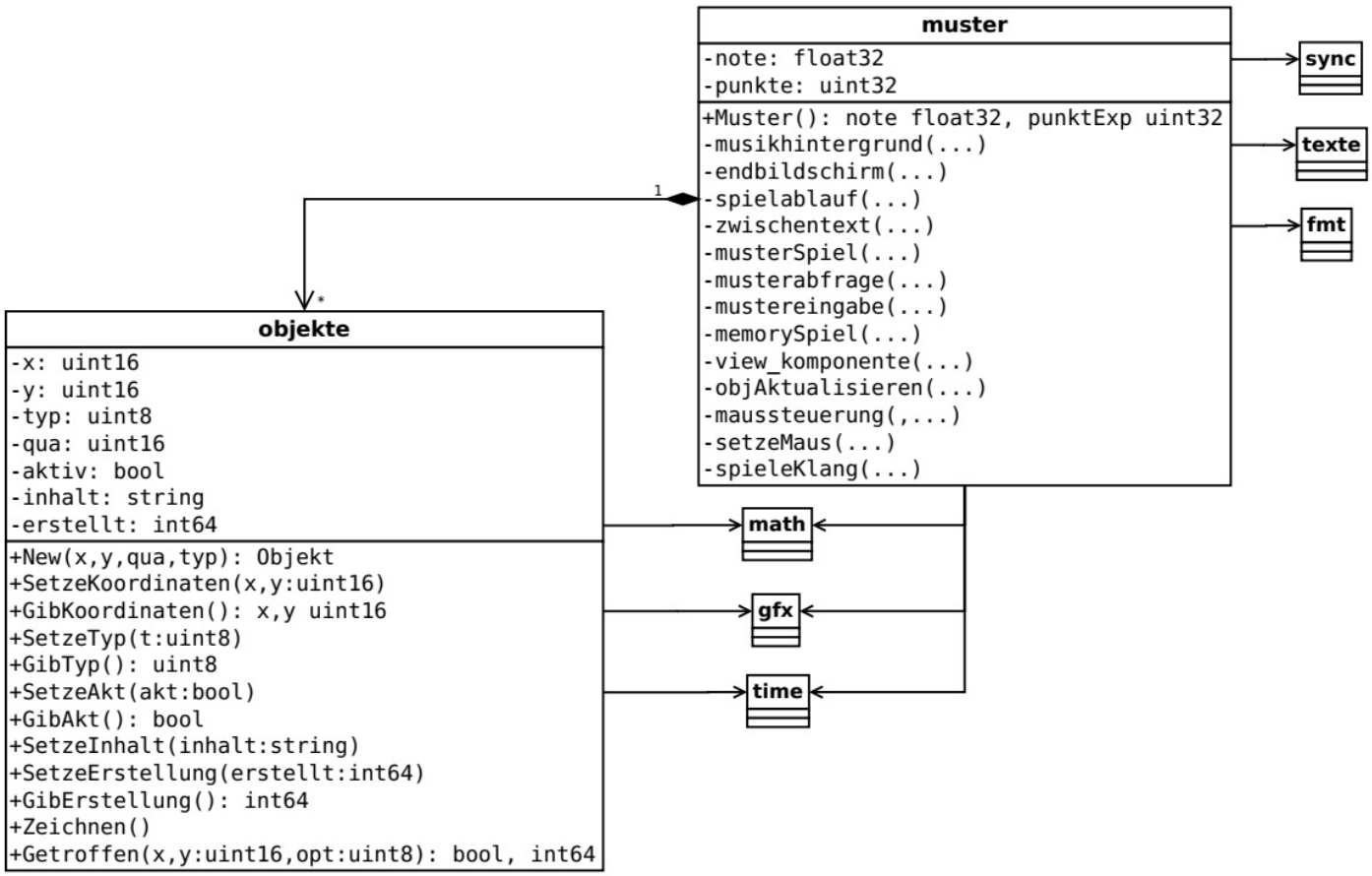
Verwendete Klassen:

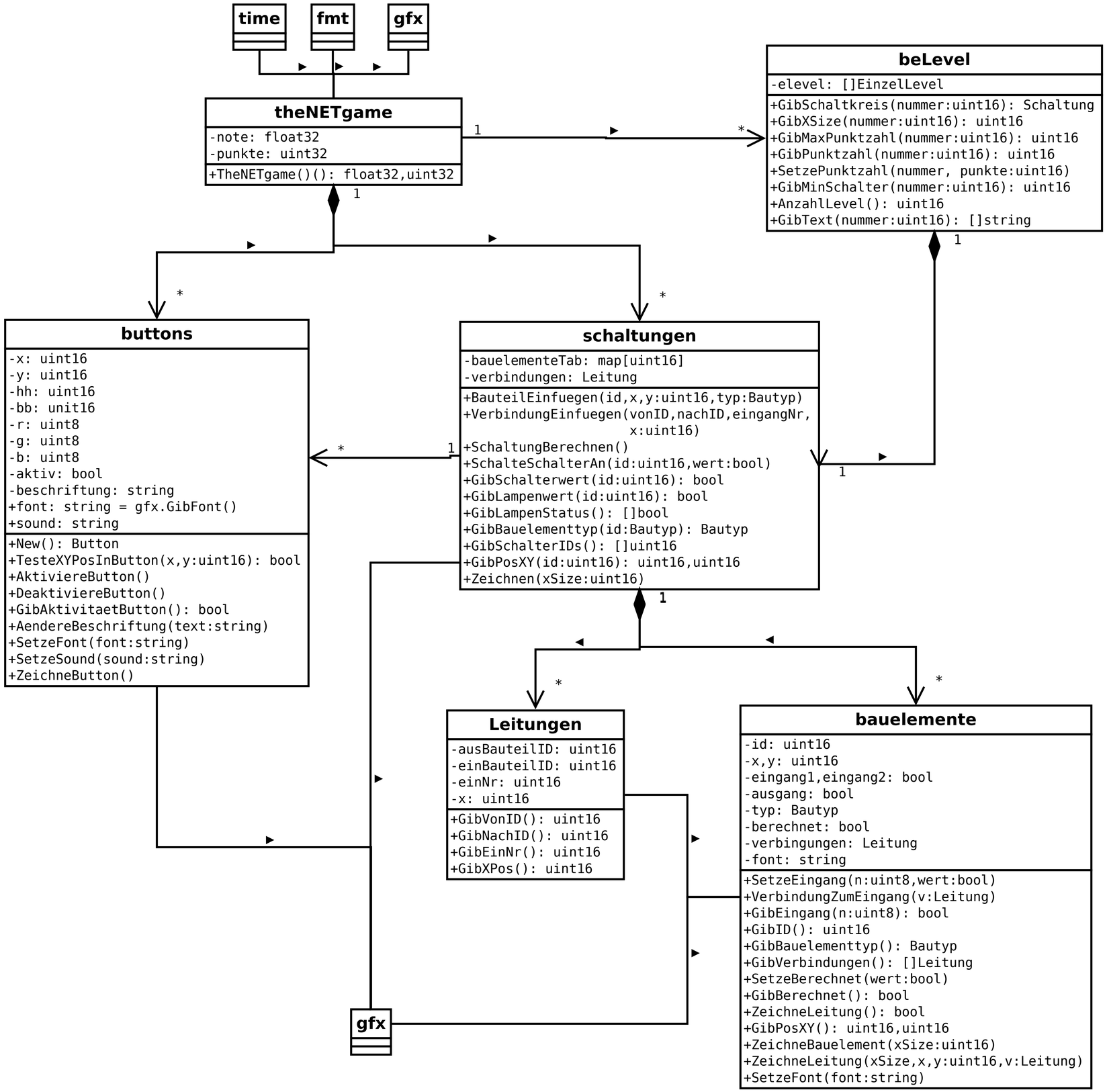
- Die **Klasse „objekte“** wurde so geschrieben, dass sie diverse Funktionen vereint. Sie ist sehr umfangreich und flexibel für andere Spiele nutzbar. Beinhaltet sind die Disigns aller dargestellten Objekte, die als Bilder geladen oder über gfx gezeichnet werden. Über die Methoden der Klassen sollten sämtliche Operationen der Objekte gesteuert werden können. Dies beinhaltet ihre Lage (Koordinaten) im gfx-Fenster, die Aktivität, ihre dargestellte Größe, ihren Typ und ihre Erstellungszeit, einen optionalen Text (string) zur Beschriftung und eine Bedingung für eingegebene Koordinaten, sodass das entsprechende Objekt „getroffen“ worden ist.

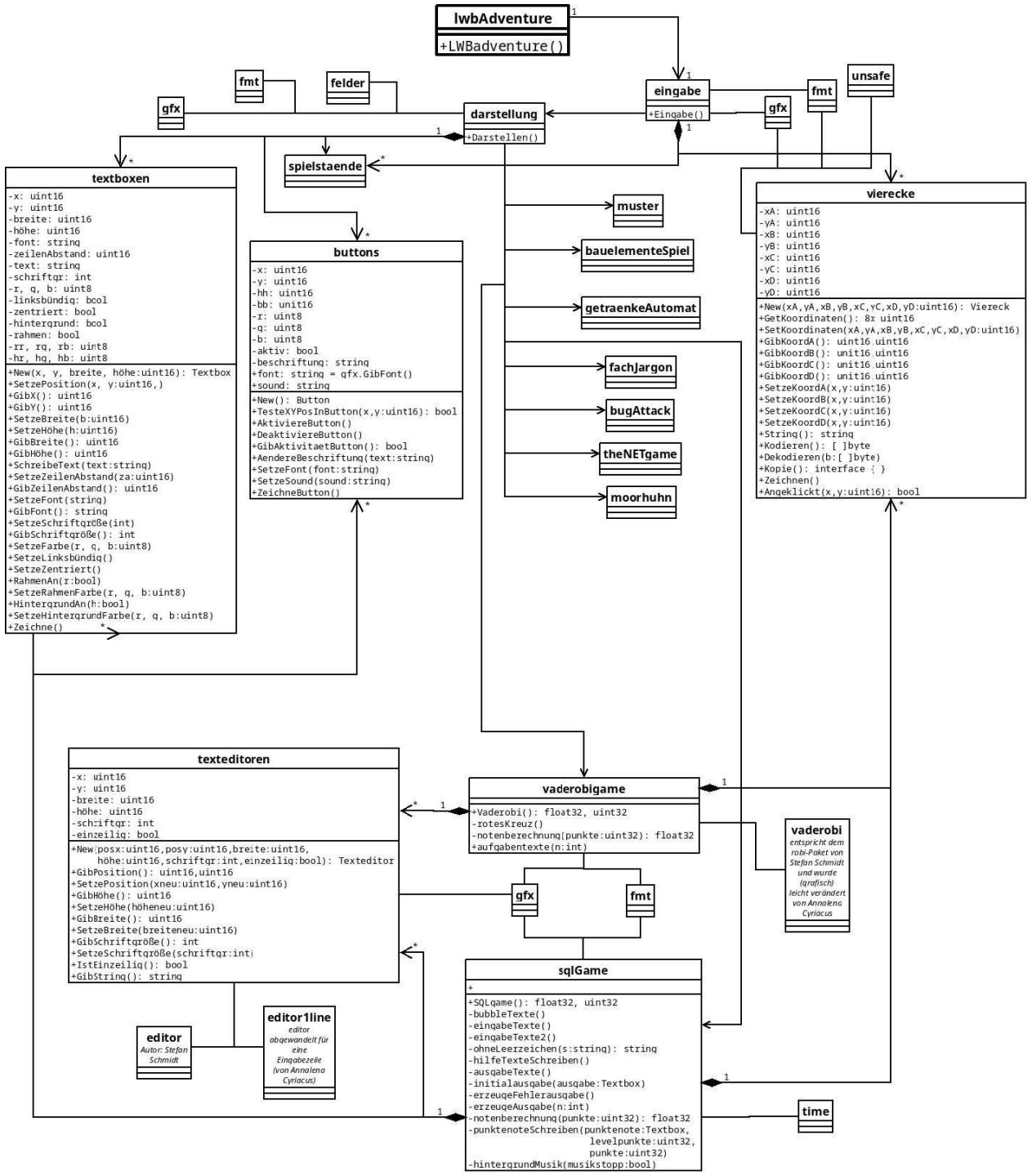
- Die Klasse **„texte“** enthält als zu exportierende Variablen Strings und Arrays/Slices von Strings, die im Hauptprogramm die Übersichtlichkeit verbessern sollen.

- Weitere Klassen (für z. B. Zwischenanzeigen, Berechnung der Endnote oder die Anzege des Enbildschirms) hätten separat erzeugt werden können. Da aber von diesen **Zusatzfunktionen** auf diverse geteilte Variablen zugegriffen wird und diverse Interaktionen innerhalb stattfinden sollen, wurde von einer Auslagerung abgesehen.

UML-Diagramm zum muster-Paket:




time
fmt
gfx
theNETgame
-note: float32
-punkte: uint32
+TheNETgame()(): float32,uint32
beLevel
-elevel: []EinzelLevel
+GibSchaltkreis(nummer:uint16): Schaltung
+GibXSize(nummer:uint16): uint16
+GibMaxPunktzahl(nummer:uint16): uint16
+GibPunktzahl(nummer:uint16): uint16
+SetzePunktzahl(nummer, punkte:uint16)
+GibMinSchalter(nummer:uint16): uint16
+AnzahlLevel(): uint16
+GibText(nummer:uint16): []string
buttons
-x: uint16
-y: uint16
-hh: uint16
-bb: unit16
-r: uint8
-g: uint8
-b: uint8
-aktiv: bool
-beschriftung: string
+font: string = gfx.GibFont()
+sound: string
+New(): Button
+TesteXYPosInButton(x,y:uint16): bool
+AktiviereButton()
+DeaktiviereButton()
+GibAktivitaetButton(): bool
+AendereBeschriftung(text:string)
+SetzeFont(font:string)
+SetzeSound(sound:string)
+ZeichneButton()
schaltungen
-bauelementeTab: map[uint16]
-verbindungen: Leitung
+BauteilEinfuegen(id,x,y:uint16,typ:Bautyp)
+VerbindungEinfuegen(vonID,nachID,eingangNr,
x:uint16)
+SchaltungBerechnen()
+SchalteSchalterAn(id:uint16,wert:bool)
+GibSchalterwert(id:uint16): bool
+GibLampenwert(id:uint16): bool
+GibLampenStatus(): []bool
+GibBauelementtyp(id:Bautyp): Bautyp
+GibSchalterIDs(): []uint16
+GibPosXY(id:uint16): uint16,uint16
+Zeichnen(xSize:uint16)
Leitungen
-ausBauteilID: uint16
-einBauteilID: uint16
-einNr: uint16
-x: uint16
+GibVonID(): uint16
+GibNachID(): uint16
+GibEinNr(): uint16
+GibXPos(): uint16
bauelemente
-id: uint16
-x,y: uint16
-eingang1,eingang2: bool
-ausgang: bool
-typ: Bautyp
-berechnet: bool
-verbingungen: Leitung
-font: string
+SetzeEingang(n:uint8,wert:bool)
+VerbindungZumEingang(v:Leitung)
+GibEingang(n:uint8): bool
+GibID(): uint16
+GibBauelementtyp(): Bautyp
+GibVerbindungen(): []Leitung
+SetzeBerechnet(wert:bool)
+GibBerechnet(): bool
+ZeichneLeitung(): bool
+GibPosXY(): uint16,uint16
+ZeichneBauelement(xSize:uint16)
+ZeichneLeitung(xSize,x,y:uint16,v:Leitung)
+SetzeFont(font:string)
gfx
1
*


lwbAdventure
+LWBadventure()
fmt
felder
gfx
darstellung
+Darstellen()
eingabe
+Eingabe()
fmt
gfx
unsafe
spielstaende
textboxen
-x: uint16
-y: uint16
-breite: uint16
-höhe: uint16
-font: string
-zeilenAbstand: uint16
-text: string
-schriftgr: int
-r, g, b: uint8
-linksbündig: bool
-zentriert: bool
-hintergrund: bool
-rahmen: bool
-rr, rg, rb: uint8
-hr, hg, hb: uint8
+New(x, y, breite, höhe:uint16): Textbox
+SetzePosition(x, y:uint16,)
+GibX(): uint16
+GibY(): uint16
+SetzeBreite(b:uint16)
+SetzeHöhe(h:uint16)
+GibBreite(): uint16
+GibHöhe(): uint16
+SchreibeText(text:string)
+SetzeZeilenAbstand(za:uint16)
+GibZeilenAbstand(): uint16
+SetzeFont(string)
+GibFont(): string
+SetzeSchriftgröße(int)
+GibSchriftgröße(): int
+SetzeFarbe(r, g, b:uint8)
+SetzeLinksbündig()
+SetzeZentriert()
+RahmenAn(r:bool)
+SetzeRahmenFarbe(r, g, b:uint8)
+HintergrundAn(h:bool)
+SetzeHintergrundFarbe(r, g, b:uint8)
+Zeichne()
buttons
-x: uint16
-y: uint16
-hh: uint16
-bb: unit16
-r: uint8
-g: uint8
-b: uint8
-aktiv: bool
-beschriftung: string
+font: string = gfx.GibFont()
+sound: string
+New(): Button
+TesteXYPosInButton(x,y:uint16): bool
+AktiviereButton()
+DeaktiviereButton()
+GibAktivitaetButton(): bool
+AendereBeschriftung(text:string)
+SetzeFont(font:string)
+SetzeSound(sound:string)
+ZeichneButton()
muster
bauelementeSpiel
getraenkeAutomat
fachJargon
bugAttack
theNETgame
moorhuhn
vierecke
-xA: uint16
-yA: uint16
-xB: uint16
-yB: uint16
-xC: uint16
-yC: uint16
-xD: uint16
-yD: uint16
+New(xA,yA,xB,yB,xC,yC,xD,yD:uint16): Viereck
+GetKoordinaten(): 8x uint16
+SetKoordinaten(xA,yA,xB,yB,xC,yC,xD,yD:uint16)
+GibKoordA(): uint16,uint16
+GibKoordB(): uint16,uint16
+GibKoordC(): uint16,uint16
+GibKoordD(): uint16,uint16
+SetzeKoordA(x,y:uint16)
+SetzeKoordB(x,y:uint16)
+SetzeKoordC(x,y:uint16)
+SetzeKoordD(x,y:uint16)
+String(): string
+Kodieren(): [ ]byte
+Dekodieren(b:[ ]byte)
+Kopie(): interface { }
+Zeichnen()
+Angeklickt(x,y:uint16): bool
texteditoren
-x: uint16
-y: uint16
-breite: uint16
-höhe: uint16
-schriftgr: int
-einzeilig: bool
+New(posx:uint16,posy:uint16,breite:uint16,
höhe:uint16,schriftgr:int,einzeilig:bool): Texteditor
+GibPosition(): uint16,uint16
+SetzePosition(xneu:uint16,yneu:uint16)
+GibHöhe(): uint16
+SetzeHöhe(höheneu:uint16)
+GibBreite(): uint16
+SetzeBreite(breiteneu:uint16)
+GibSchriftgröße(): int
+SetzeSchriftgröße(schriftgr:int)
+IstEinzeilig(): bool
+GibString(): string
vaderobigame
+Vaderobi(): float32, uint32
-rotesKreuz()
-notenberechnung(punkte:uint32): float32
+aufgabentexte(n:int)
gfx
fmt
vaderobi
entspricht dem robi-Paket von Stefan Schmidt und wurde (grafisch) leicht verändert von Annalena Cyriacus
sqlGame
+
+SQLgame(): float32, uint32
-bubbleTexte()
-eingabeTexte()
-eingabeTexte2()
-ohneLeerzeichen(s:string): string
-hilfeTexteSchreiben()
-ausgabeTexte()
-initialausgabe(ausgabe:Textbox)
-erzeugeFehlerausgabe()
-erzeugeAusgabe(n:int)
-notenberechnung(punkte:uint32): float32
-punktenotenSchreiben(punktenote:Textbox,
levelpunkte:uint32,
punkte:uint32)
-hintergrundMusik(musikstopp:bool)
time
editor
Autor: Stefan Schmidt
editor1line
editor abgewandelt für eine Eingabezeile (von Annalena Cyriacus)

Entwurfsentscheidungen

vaderobigame	sqlGame
- Grundlage des Spiels ist das **robi-Paket** von Stefan Schmidt, das von mir leicht verändert wurde. Aus dem robi ist vaderobi geworden und die vorhandenen Grafik-Elemente und Methoden sind teilweise leicht verändert, um auch die sonstige Darstellung der Welt etwas an den vaderobi anzupassen. - Neben der vaderobi-Welt gibt es **Text-Elemente**, die das Spiel beschreiben bzw. erläutern und einen **Texteditor**, in den Befehle zur Steuerung des vaderobis eingegeben werden können. - Anstelle des Texteditors hätte auch die Klasse „Felder“ genutzt werden können, die Entscheidung ist aber auf den im Rahmen der Veranstaltung ALP3 entwickelten Texteditor gefallen, da durch die gemeinsame schrittweise Erarbeitung eine Anpassung der editor-Datei und deren Einbindung in eine Klasse „texteditoren“ recht niederschwellig möglich war.	- Die grafische Oberfläche des Spiels besteht aus Bildern und **Text-Elementen**, die das Spiel beschreiben bzw. erläutern und durch die Level führen. - Die Spiel-Steuerung läuft über einen next-Button und einen **Texteditor**, in den SQL-Anfragen eingegeben werden sollen. - Da eine Anbindung an eine tatsächliche Datenbank in Rahmen des SWP-Projekts zu aufwendig und vor allem zu zeitintensiv gewesen wäre, sind nur die zu den richtigen SQL-Anfragen passenden **Ausgaben** implementiert und werden bei richtiger Eingabe angezeigt. - Bei falscher Eingabe erfolgt keine Ausgabe der Anfrage-Ergebnisse, sondern eine **Fehlermeldung** mit Hinweisen zu Syntax und Schreibweise sowie jeweils **gestaffelte Hilfen** für den 3. und 4. Versuch. - Bei vier Falscheingaben wird die richtige Lösung angezeigt und es geht ohne Punkte für das nicht geschaffte Level über den next-Button ins nächste Level.
- Da **beide Spiel mithilfe von textuellen Benutzer-Eingaben gesteuert** werden, wurde hierfür ein Klasse „texteditoren“ entworfen. - Die **Klasse „texteditoren“** basiert auf dem editor08-Quelltext von Stefan Schmidt aus der ALP3-Veranstaltung im drittem Semester, der benutzt wurde, um die Pakete editor und editor1line zu erstellen, die zur Erzeugung der Instanzen der Klasse „texteditoren“ genutzt werden. Die Klasse kommt in beiden Spielen zur Anwendung und kann (ggf. mit leichten Anpassungen/Ergänzungen) auch in anderen Zusammenhängen genutzt werden. Neben Position, Abmessung und Schriftgröße kann bei der Initialisierung auch entschieden werden, ob der Texteditor ein- oder mehrzeilig sein soll. Beim einzeiligen Editor ist die ENTER-Taste so belegt, dass damit statt einem Zeilenumbruch (wie im mehrzeiligen Editor) die Eingabe bestätigt wird. Die Darstellung des Texteditors ist für die Nutzung in den beiden Mini-Games ausgelegt, die Darstellungs-Implementierung ließe sich sicher noch optimieren und könnte/müsste für die Nutzung in anderen Zusammenhängen ggf. angepasst werden. - Die **Klasse „vierecke“** wird in beiden Spielen sowie im Main-Game benutzt. Sie dient der Erstellung von allgemeinen Vierecken, um entsprechend geformte grafische Elemente in einem gfx-Fenster (un-/sichtbar) nachzuzeichnen und anklickbar zu machen. Bei der Initialisierung werden die Koordinaten der vier Ecken übergeben, dadurch entsteht eine größere Variabilität als bei Quadraten oder Rechtecken. In den beiden Mini-Games wird mithilfe dieser Klasse das Exit-Symbol anklickbar gemacht, im Main-Game die Türen im Mainfloor sowie das Info-Symbol an der Pinnwand und die Verlassen-Symbole – also alle klickbaren Elemente, die nicht durch Buttons dargestellt sind. - Beide Spiele können mithilfe der Texteingabe **„exit“** in den Texteditor vorzeitig beendet werden (kein klickbarer Button, da das Spiel ansonsten auch nur über Tastatur-Eingaben gesteuert wird). Gespeichert und zurückgegeben wird die bis dahin erreichte aktuelle Gesamtpunktzahl und Gesamtnote.	

Designentscheidungen für das Minigame *BugAttack* *(Philipp Liehm)*

Das Minigame basiert auf einem **Paket „bugPackage“** damit innerhalb des Spiels alle Funktionen vor dem Hauptspiel versteckt sind und nur die eigentlich Spiel-Funktion exportiert wird.

- **Tastatureingabe:** eigenständige Go-Routine / Funktion die über das ganze Spiel hinweg über funktioniert.

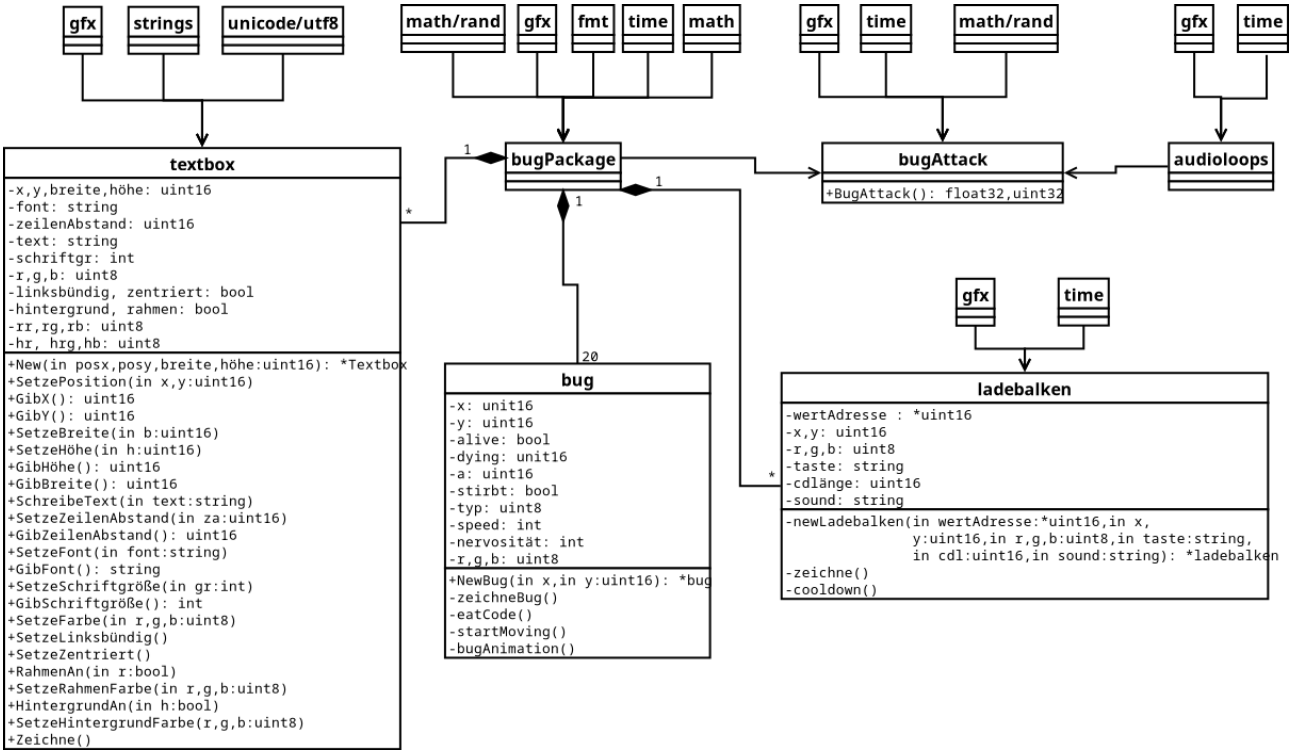
- Die **Klasse „textbox“** wurde so geschrieben, dass sie auch für andere Spiele genutzt werden kann. Hauptziel der Klasse ist es automatische und manuelle Zeilenumbrüche zu ermöglichen, somit muss nicht jedes mal neu mit dem gfx-Paket geschrieben werden. Die Klasse stellt den Text nicht immer nur in der Textbox da, machmal auch darüber hinaus. Dies liegt an der Schwierigkeit die tatsächliche Breite eines Worts/Zeichens abzuschätzen, da diese von Zeichen zu Zeichen unterschiedliche ist und sogar zwischen einzelnen Schriftarten nicht gleich ist. Daher wird die Breite nur in Bezug auf die Schriftgröße geschätzt, dies ist aber für unsere Anwendung ausreichend.

- Die **Klasse „bugs“** ist nur für das „bugPackage“ implementiert, da die Bugs so spezifisch auf dieses Minigame angepasst sind. Die „bugs“ verwenden ein bestimmtes Raster der Welt im „bugPackage“ und greifen auf globale Variablen des Pakets zu.
→ Die Klasse wurde mit verschiedenen Attributen wie „alive“, „dying“, „stirbt“ konstruiert. Diese scheinen ähnlich, sind aber sinnvoll, da parallel zur Bewegung der „bugs“ auch eine Animation der „bugs“ abläuft.
→ Die Anzahl der „bugs“ ist auf 20 begrenzt, da es sonst wegen der Größe der „bugs“ zu viele werden.

- Die **Klasse „ladebalken“** ist auch eng mit dem „bugPackage“ verknüpft. Das Grundprinzip ist, dass eine global definierte Zählvariable beobachtet wird und abhängig von der Größe ein Rechteck gezeichnet sowie die Zählvariable selbst verändert (hochgezählt) wird. Wichtig war hier, dass die zugehörige Taste und die Geschwindigkeit mit der hochgezählt wird, übergeben wird.

- Es gibt verschiedene **Zusatzfunktionen**, wesentlich sind z.B.:
→ Das Feld in dem die Zeiger der „bugs“ gespeichert sind wird regelmäßig aufgeräumt, d.h. „bugs“ die nicht mehr leben werden gelöscht
→ Dieses Feld ist durch ein Schloss geschützt. Das ist notwendig, da parallel gelesen und geschrieben wird und es zu Fehlern kommen kann wenn z.B. gerade von einem nicht mehr existenten „bug“ gelesen werden soll.
→ Die Level sind in einzelne Funktionen ausgelagert. Es gibt allerdings eine Funktion zum Start der Level, hier werden immer gleich ablaufende Routinen zusammengefasst. Somit muss in den einzelnen Level-Funktionen nur die Anzahl der „bugs“ und deren Parameter sowie „ladebalken“ definiert werden.

- Ein kurzes **Paket „audioloops“** ermöglicht Audiodateien wiederkehrend abzuspielen. Dafür muss allerdings die Audiolänge bekannt sein. Diese Pakte kann unabhängig vom „bugPackage“ verwendet werden.


gfx
strings
unicode/utf8
math/rand
gfx
fmt
time
math
gfx
time
math/rand
gfx
time
textbox
-x,y,breite,höhe: uint16
-font: string
-zeilenAbstand: uint16
-text: string
-schriftgr: int
-r,g,b: uint8
-linksbündig, zentriert: bool
-hintergrund, rahmen: bool
-rr,rg,rb: uint8
-hr, hrg,hb: uint8
+New(in posx,posy,breite,höhe:uint16): *Textbox
+SetzePosition(in x,y:uint16)
+GibX(): uint16
+GibY(): uint16
+SetzeBreite(in b:uint16)
+SetzeHöhe(in h:uint16)
+GibHöhe(): uint16
+GibBreite(): uint16
+SchreibeText(in text:string)
+SetzeZeilenAbstand(in za:uint16)
+GibZeilenAbstand(): uint16
+SetzeFont(in font:string)
+GibFont(): string
+SetzeSchriftgröße(in gr:int)
+GibSchriftgröße(): int
+SetzeFarbe(in r,g,b:uint8)
+SetzeLinksbündig()
+SetzeZentriert()
+RahmenAn(in r:bool)
+SetzeRahmenFarbe(in r,g,b:uint8)
+HintergrundAn(in h:bool)
+SetzeHintergrundFarbe(r,g,b:uint8)
+Zeichne()
1
*
bugPackage
1
1
bugAttack
+BugAttack(): float32,uint32
audioloops
20
bug
-x: unit16
-y: uint16
-alive: bool
-dying: unit16
-a: uint16
-stirbt: bool
-typ: uint8
-speed: int
-nervosität: int
-r,g,b: uint8
+NewBug(in x,in y:uint16): *bug
-zeichneBug()
-eatCode()
-startMoving()
-bugAnimation()
*
gfx
time
ladebalken
-wertAdresse : *uint16
-x,y: uint16
-r,g,b: uint8
-taste: string
-cdlänge: uint16
-sound: string
-newLadebalken(in wertAdresse:*uint16,in x,
y:uint16,in r,g,b:uint8,in taste:string,
in cdl:uint16,in sound:string): *ladebalken
-zeichne()
-cooldown()