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.

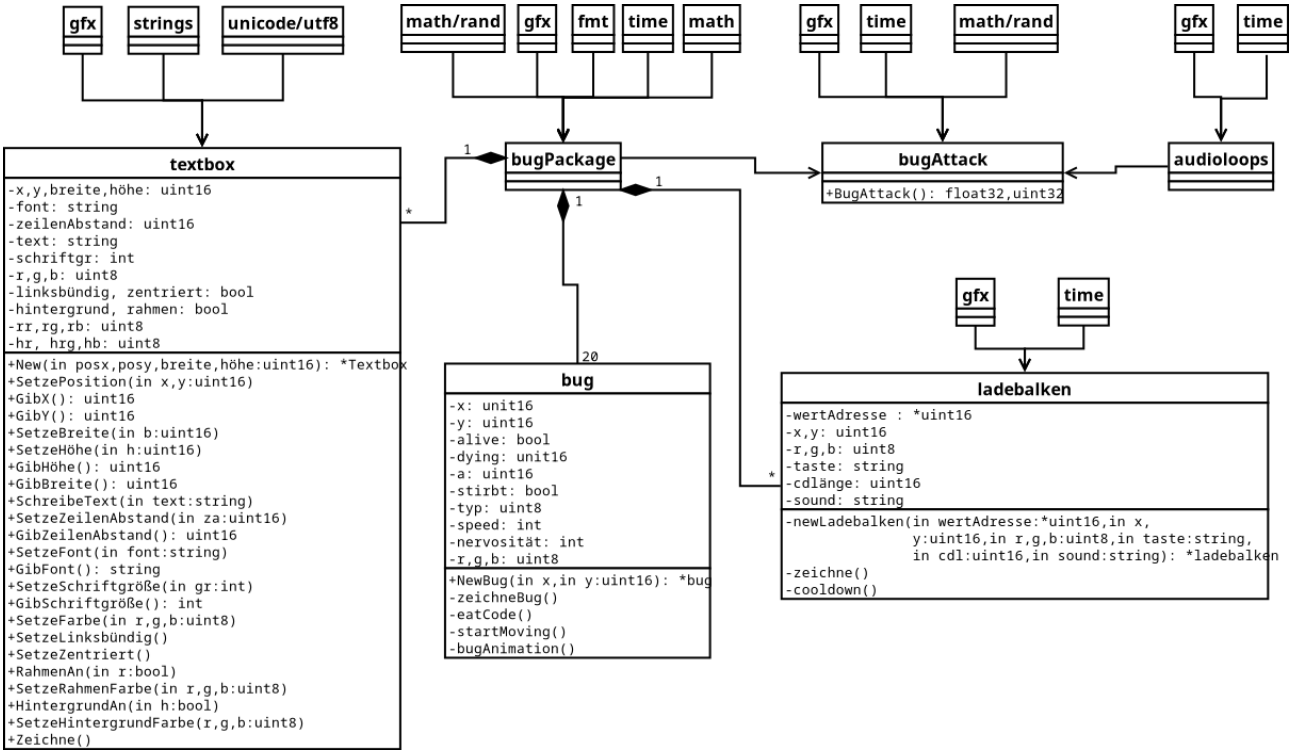
- Die **Klasse „textbox“** wurde so geschrieben, dass sie auch für andere Spiele genutzt werden kann. Hauptziel der Klasse ist es automatische und manuelle Zeilenumbrüche zu ermöglichen, somit muss nicht jedes mal neu mit dem gfx-Paket geschrieben werden. Die Klasse stellt den Text nicht immer nur in 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()