Sebastian Kriege - 5994360
Kai Brobeil - 6000693

[CG] Aufgabenblatt 6 - S

Konzeption	Unser Spiel ist stark an das als Beispiel gegebene Spiel von Albinoblacksheep angelehnt. Es gibt Löcher im Boden, durch die man maximal zweimal fallen darf. Die Steuerung erfolgt nach links und rechts, außerdem kann man über Löcher springen (dabei wird man wird kurz unverwundbar). Genauso ist man eine Sekunde (60 Frames) lang unverwundbar nachdem man gefallen ist. Das Programm funktioniert folgendermaßen: Die Hauptklasse, Kubuz, führt den main loop aus und hält Referenzen auf alle Objekte. Außerdem handelt sie die Tastatureingaben. Alle darstellbaren Objekte erben Ihren render-Code von der abstrakten Klasse Renderable. Diese stellt schon alle notwendigen GL-Operationen für ein generisches Objekt bereit (Buffer initialisieren, Texturen laden, Daten erneuern, etc). Die Klasse Playable macht das gleiche für sound-Objekte, allerdings muss sie nicht geerbt, sondern nur instanziiert werden. Renderable wird von dem Hintergrund (backdrop), dem Insect, dem Tubus, der GUI und der LevelGUI erweitert. GUI und LevelGUI benutzen außerdem einen alternativen Shader, der keine Matrizenumformungen macht und stattdessen direkt auf screen coordinates schreibt, was für die GUI vollständig ausreichend ist. Die GUI handelt den Pause- und GameOver-Screen im Wechsel mit der Anzeige der aktuellen Anzahl der Leben. Die LevelGUI stellt eine vierstellige Zahl dar, die das aktuelle Level angibt. Dazu rendert sie 4 Renderable Objekte, jeweils eine Ziffer. Wir starten mit dem Pause-Screen, damit der Spieler Zeit hat, sich vorzubereiten. Für die Umsetzung der Sound-Objekte benutzen wir eine Klasse aus LWJGL 2, die im Header der Klasse auch so gekennzeichnet ist. Leider wurde diese in LWJGL 3 nicht mehr mitgeliefert. Den PNG-Loader aus dem Beispielprojekt in Moodle übernehmen wir ebenso. Wir setzen uns jeweils am Wochenende zusammen und planen die Aufgaben für die nächste Woche (Rhythmus Samstag bis Samstag).

Milestone I – erste Woche		
Klasse	**Funktion**	**Bearbeiter**
Tubus	Erstellen eines Tubus mit x Ecken und y Segmenten pro Seite, muss eine Funktion enthalten um den Tubus in Richtung Kamera zu bewegen (moveZ)	Sebastian Kriege
simplePrimitives (author Thorsten Gattinger)	Hauptklasse zum Anzeigen von Objekten, Erstellen einer Instanz des Tubus, Aufruf der moveZ Funktion im Loop	Kai Brobeil

Milestone II – zweite Woche		
Klasse	**Funktion**	**Bearbeiter**
Kubuz	Aufsplitten der Klasse simplePrimitives in eine Hauptklasse und eine Klasse Renderable. Die Hauptklasse wird in Kubuz umbenannt. Die Hauptklasse verwaltet den Overhead von Opengl und den Loop.	Kai Brobeil
Renderable	Erstellen einer Basisklasse für alle Objekte, die gerendert werden müssen. Diese Klasse übernimmt viele Funktionen von Exampleprimitives. Somit kann jedes Objekt einen anderen Shader und eine eigene Textur verwenden.	Sebastian Kriege
Insect (extends Renderable)	Geometrie, Animation (Laufen, Springen) und Textur erstellen.	Sebastian Kriege
Level	Eine Klasse um Level zu erstellen. Als Datenstruktur ein 2d-Array von Booleans verwenden, das angibt an welcher Stelle ein Loch im Tubus sein soll.	Sebastian Kriege
GUI (extends Renderable)	Eine Klasse zum Erstellen von 2D-Overlays. (Pause, Anzahl Leben, GameOver)	Kai Brobeil
Tubus (extends Renderable)	Abändern vom Tubus, so dass er auch über Renderable verwaltet wird und die Level angezeigt werden.	Sebastian Kriege
Backdrop (extends Renderable)	Hintergrund implementieren	Kai Borbeil

Milestone III – dritte Woche		
Klasse	**Funktion**	**Bearbeiter**
Renderable	Erweitern der Klasse Renderable um eine Funktion, um die Translation und Rotationsmatrizen zu verwenden.	Sebastian Kriege
Kubuz	Tastensteuerung ohne Verzögerung implementieren, Bewegung des Läufers per Tastendruck und Abfrage zum Herunterfallen einbauen. Zusätzlich soll die Anzahl an Seiten per Knopfdruck im Pause-Screen steuerbar sein.	Kai Brobeil
LevelGUI	Eine Fortschrittsanzeige für Level implementieren.	Kai Brobeil
Playable	Eine Klasse zum Abspielen von Sounds implementieren.	Kai Brobeil

Klasse		**Kubuz**
Funktion	**Parameter**	**Beschreibung**
loop	-	Hier findet unsere Mechanik statt. Wann soll sich das Insekt bewegen, fällt es als nächstes durch ein Loch oder wurde gerade der Pause-Knopf gedrückt. Folgende Tastenbelegungen verwenden wir: • backspace: (nur im Pause-Screen!) mehr Kanten, das Spiel wird zurückgesetzt • b: pause und weiter • <- und ->: Steuerung • esc: (nur im GameOver-Screen) reset, das Spiel startet erneut
pause, resume	-	Regelt das Pause-Verhalten. Die paused-Variable wird auf true/false gesetzt und der Sound wird aus/an geschaltet.
restart	-	Resettet alle Spielbestandteile.

Klasse		**Renderable**
Funktion	**Parameter**	**Beschreibung**
createGeometry	-	Erstellt VertexArray, textureArray und IndexArray für die Darstellung des Objektes
init	-	Muss im Konstruktor jeder Subklasse aufgerufen werden. Regelt die Erstellung der Buffer, der Shader und der Translationsmatrix.
modifyModel	float setMX, float setMY, float setMZ, float setTX, float setTY, float setTZ	Diese Funktion erlaubt die additive Veränderung der Translationsmatrix. Dabei sind die ersten drei Werte für die Rotation um die jeweilige Achse und die letzten drei Werte für die Translation. Mit der Funktion resetTranslationMatrix können diese wieder auf den default-Wert zurückgesetzt werden.
initBuffers / **updateBuffers**	-	Um unnötige Erstellung von Bufferobjekten im Arbeitsspeicher zu vermeiden haben wir die Erstellung und das Updaten der Buffer in zwei Funktionen aufgeteilt.
render	-	Diese Funktion führt den Draw-Aufruf durch. Da jedes Objekt eine eigene Textur hat wollten wir für jedes Objekt einen separaten Aufruf haben.

Klasse		**Insect**
Funktion	**Parameter**	**Beschreibung**
createLegAnimation	int number, float scaleX, float[] root	Number ist der Index des Beines, insgesamt gibt es 6 Beine. ScaleX ist der Abstand in X-Richtung zwischen dem Körper und dem Punkt, wo der Fuß den Boden berührt. Root ist ein Punkt im 3-dimensionalen Raum, von der aus das Bein erstellt werden soll. Diese Funktion erstellt eine Animationsliste für ein Bein. Das Bein selbst ist Flach und besteht nur aus 4 Punkten.
animateLeg	int number, int step	Number ist wieder Index des Beines, Step bestimmt welcher Animationsschritt verwendet werden soll. Die Funktion schreibt den nächsten Animationsschritt in den vertexArray. IndexArray und textureArray bleiben unangetastet.
animate	-	Regelt die Animation aller Beine.
jump	-	Setzt die Variable für den Zustand springen (jumpingSince) auf 1. Somit kann die Funktion doJumpStep aufgerufen werden und die Sprunganimation schrittweise durchgeführt werden.
fall	-	Führt die Fallanimation schrittweise durch.

Klasse		**Tubus**
Funktion	**Parameter**	**Beschreibung**
createVertexArray	-	Erstellt ein Segment (Ring) vom Tubus an Position 1f, schreibt ihn in den vertexArray, verschiebt das Segment um segmentSizeZ und schreibt es wieder in den vertexArray. Dieser Vorgang wird renderDepth-mal wiederholt.
createIndexArray	-	Um zu überprüfen, an welchen Stellen sich ein Loch befindet ist hier eine Abfrage von curLevel (2d-Array von Booleans). Falls die nächste Position true ergibt wird diese Position übersprungen und das Viereck wird nicht gezeichnet.
moveZ	offsetZ	Diese Funktion sorgt für die Bewegung des Tubus in z-Richtung. Falls der Tubus zu weit nach vorne bewegt werden soll wird er um segmentSizeZ nach hinten verschoben und die Variable position, welche die Position im Level darstellt wird erhöht.

turn	boolean direction	Anstatt den kompletten Tubus zu drehen bleiben die Vertexe unangetastet. Lediglich die Variable rotation wird angepasst. Diese sorgt dafür, dass bei der Erzeugung vom IndexArray das Segment vom 2d-LevelArray um die Variable rotation verschoben wird. Kurz gesagt es werden nur die Löcher gedreht, nicht der Tubus.
isHole	float posX, float posZ	Überprüft, ob sich an der Position posX, posZ ein Loch befindet und gibt dann true zurück.
progress	-	Regelt den Übergang zum nächsten Level.

Klasse		**GUI**
Funktion	**Parameter**	**Beschreibung**
reduceLife	-	Wenn der Läufer ein Leben verliert wird diese Funktion aufgerufen und reduziert die Anzahl an angezeigten Herzen. Gibt true zurück falls der Läufer keine Leben mehr hat und false falls das Spiel weitergehen soll.

Klasse		**LevelGUI**
Funktion	**Parameter**	**Beschreibung**
increase	-	Verwaltet die 4 Zählerstellen und erhöht den Zähler um 1.

Klasse		**Playable**
Funktion	**Parameter**	**Beschreibung**
play	-	Die Klasse regelt die Wiedergabe von Soundfiles. Die Funktion play startet die Wiedergabe.

Quellen	
Hintergrund	http://opengameart.org/node/25677
Weitere Bilder	Alle weiteren Bilder wurden selbst in GIMP2 erstellt.
Soundfiles	http://www.freesound.org/
Wavedata.java	https://github.com/LWJGL/lwjgl/blob/master/src/java/org/lwjgl/util/WaveData.java