TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Fakultät für Mathematik und Informatik
Institut für Informatik

Seminararbeit

Rocketdodge

Asteroids Revamp

Lukas Hein,
Dennys-Daniel Vogt

Angewandte Informatik
Matrikel: 63543,
63411

19. Oktober 2020

Betreuer/1. Korrektor:
Ben Lorenz, Jonas Träumer

2. Korrektor:
Prof. Dr. Konrad Froitzheim

Eidesstattliche Erklärung

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Diese Versicherung bezieht sich auch auf die bildlichen Darstellungen.

19. Oktober 2020

Lukas Hein,
Dennys-Daniel Vogt

Inhaltsverzeichnis

1 Einleitung **4**

2 Das Originalspiel **5**

3 Codedokumentation **6**
3.1 Bibliotheken 6
3.2 Programmablauf 6
3.3 Spiellogik 7
3.4 Darstellung 8
3.5 Grafiklisten 8

4 Projektdokumentation **9**
4.1 Genutzte Hilfsmittel 9
4.2 Arbeitsaufteilung 9
4.3 Zeitplanung und Meilensteine 10
4.4 Arbeitsweise 10

5 Benutzerhandbuch **12**
5.1 Vor dem Spiel 12
5.2 Spielprinzip und Aufbau 12
5.2.1 Overlay 12
5.2.2 Objekte 12
5.2.3 Levels 14
5.2.4 Schwierigkeitsgrad 15
5.3 Steuerung 16
5.3.1 Raumschiff 16
5.3.2 Spielmenü 16

6 Zusammenfassung **17**

7 Literaturverzeichnis **20**

1 Einleitung

Rocket Dodge (siehe Abb. 1) ist ein erweitertes Spiel aus 2008. Der ursprüngliche Name war Spaceshooter. Dieser Name war nach Spielweise unangemessen und wurde deshalb umbenannt. Der ursprüngliche Autor war Samuel Simeonov (Originalspiel anbei). Das Spiel wurde als Demo im Rahmen einer Bewerbung genutzt. Es wurde im Rahmen des Multimedia-Projekts an der TU Freiberg als Prüfungsleistung erweitert und verbessert. Jegliche Neuerungen wurden von Lukas Hein und Dennys-Daniel Vogt im Jahre 2020 erarbeitet.
Bei dem Spiel handelt es sich um ein 3D-Ausweich-Spiel im Weltraum-Stil. Angezeigt wird es in einer 2D-Weise. Dabei kommen Raketen von verschiedenen Positionen auf ein Raumschiff zugeflogen. Das Ziel ist es den Raketen erfolgreich auszuweichen, um so lange wie möglich zu überleben.

Diese Arbeit ist gegliedert in vier Abschnitte. Im ersten Teil wird das Originalspiel kurz gezeigt und erläutert (vgl. Kapitel 2). Im zweiten Abschnitt wird auf den erarbeiteten Sourcecode eingegangen. Es wird der allgemeine Programmablauf gezeigt. Ebenfalls wird auf die Spiellogik, die verschiedenen Darstellungsmöglichkeiten und die implementierten 3D-Modelle eingegangen (vgl. Kapitel 3). Im dritten Teil wird die Projekterarbeitung beleuchtet. Es wird gezeigt, wie das Projekt strukturiert wurde, welche Hilfsmittel genutzt wurden und wie die Arbeitsweise jedes Einzelnen und im Team war (vgl. Kapitel 4). Im letzten Abschnitt wird ein kleines benutzerfreundliches Handbuch präsentiert, in dem die Steuerung und das Spiel im Detail erklärt ist (vgl. Kapitel 5). Zum Schluss werden die wichtigsten Daten im Kapitel 6 zusammengefasst.

Abb. 1: Rocket Dodge Cover

2 Das Originalspiel

Das Spiel wurde von Samuel Simeonov im Jahre 2008 als Demo im Rahmen einer Bewerbung geschrieben. Es nutzt die Bibliothek GL_freeglut.h (OpenGL Utility Tool) zur Koordinierung des Spielablaufs und zur Darstellung bestimmter Körper. Beim Start des Projekts war das gegebene Spiel weder kompilierfähig, noch gab es ein Buildingtool.
Das Spiel wurde simpel gehalten. Es gab 3 verschiedene 3D-Modelle:

- Das Raumschiff
- Der Planet
- Die Raketen

Die Objekte wurden mithilfe bestimmter geometrischer Objekte (Quader, Dreiecke, Zylinder etc.) erstellt. Diese konnten implementiert werden durch die Bibliothekfunktionen für Körper und Flächen. Ein Grundverständnis von Vektoren und Vektorrechnung wird vorausgesetzt. Durch die Aneinanderreihung dieser Objekte wurden Listen erstellt, die am Ende unsere 3D-Modelle darstellen. Weitere Informationen zur Darstellung und deren Listen können im Kapitel 3 nachgelesen werden.
In der Übersicht sind folgende Werte und Lichtquellen enthalten:

Werte:

- Score
- Hits

Lichtquellen:

- Spotlicht in weiß von unten rechts
- Spotlicht in grün von oben rechts
- Spotlicht in blau von unten links
- Spotlicht in rot von oben links

Der Score zeigt die bereits erspielten Punkte an. Die Hits geben an wie viel Leben noch zur Verfügung stehen.
Die Lichtquellen sind auf den Planeten in der Mitte (siehe Abb. 2) mit einem Scheinwinkel von 25 Grad gerichtet.

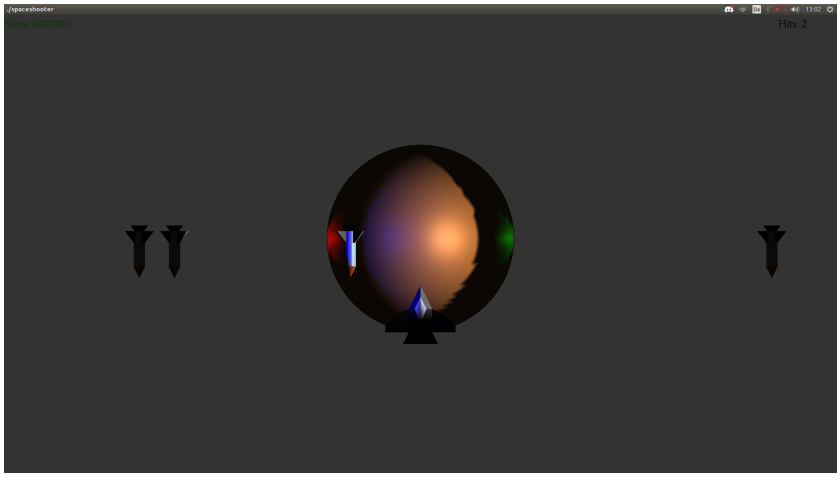


Abb. 2: Originalspiel aus 2008

3 Codedokumentation

In diesem Kapitel wird auf den Code im spezifischen eingegangen. Es werden die Bibliotheken, der allgemeine Programmablauf, die Spiellogik, die Darstellung und die Grafiklisten erklärt. Allgemein ist zu erwähnen, dass das Spiel in einem prozeduralen Stil in der Sprache C++ geschrieben wurde. Dieser Stil wurde vom ursprünglichen Autor (siehe Kapitel 2) übernommen.

3.1 Bibliotheken

In der Tabelle 1 werden die genutzten Bibliotheken aufgelistet und deren Hauptfunktionen innerhalb des Codes erläutert.

Tab. 1: Bibliotheken und deren Funktionen

Bibliothek	Funktion
stdio.h	Standard In- und Output/genutzt zum Debugging
stdlib.h	Enthält Standardmakros
GL/freeglut.h	genutzt zum Programmablauf/neuere Version von GLUT, da GLUT seit 1998 nicht weiter unterstützt wurde
math.h	sin/cos Funktionen zur Berechnung von Kreisbahnen
limits.h	Maximalwerte von Integer
time.h	genutzt um einen Seed für srand() zu erhalten der immer anders ist
string.h	Kommandozeilen Input

3.2 Programmablauf

Das Programm beginnt mit einer Evaluation der Kommandozeilenparameter. Dort wird geprüft ob ein weiterer Parameter (argv[1]) gesetzt wurde und ob dieser gültig ist. Falls dies der Fall ist, wird der Spielmodus auf den gewünschten Modus geändert und das Spiel gestartet. Folgend darauf wird eine zufällige Zahl bestimmt, die auf den Lichtwechsel in Level neun Einfluss hat.
Mit den folgenden GLUT-Funktionen wird das Programm initialisiert und läuft dann in einer Schleife:

1. glutInit
2. glutInitDisplayMode
3. glutInitWindowSize
4. glutInitWindowPosition
5. glutCreateWindow
6. glutFullScreen
7. glutReshapeFunc
8. glutDisplayFunc
9. glutMouseFunc
10. glutKeyboardFunc
11. glutSpecialFunc
12. glutTimerFunc
13. init
14. glutMainLoop

Zusammengefasst wird folgendes ausgeführt: GLUT wird initialisiert. Daraufhin werden die Fenstergröße und die Fensterposition bestimmt und im Anschluss das Fenster erstellt. Um das

beste Spielerlebnis zu bieten, wird ebenfalls der Vollbildmodus eingeschaltet. Die Callback-Funktionen zum Anzeigen und Neuzeichen der Spielwelt werden gesetzt. Darauf folgend werden Callback-Funktionen für Maus und Tastatur gesetzt, um die Steuerung des Raumschiffs zu ermöglichen. Um ein gleichbleibendes Spielerlebnis und die gleiche Update-Rate zu haben wird die Timer-Funktion gesetzt. Trotzdem müssen jegliche 3D-Modelle erstmals geladen werden und dafür sorgt die init() Funktion. Um die Spielschleife zu starten wird darauf hin der glutMainLoop() gerufen.

3.3 Spiellogik

Mittelpunkt der Spielmechaniken ist die Erzeugung immer zahlreicher werdender Raketen an zufälligen Positionen. Das Originalspiel (siehe Kapitel 2) verfügt über eine solche Funktion, jedoch wurde jene in diesem Projekt sowohl verbessert als auch erweitert. Den Grundbaustein bildet ein Array namens is_occupied, welches eine bestimmte Menge von potentiellen Raketenbahnen repräsentiert. Mithilfe dieses Arrays werden zu Beginn jeder Raketenwelle diese Bahnen in einer zufälligen Reihenfolge in die x-Ordinaten der Raketen umgerechnet und in eine Matrix, als object_properties bezeichnet, übernommen. Mittels der x-Ordinaten und der in timer() inkrementierten Variable rockety, wird die Position jeder Rakete festgelegt. Die Anzahl der Raketen, die tatsächlich gezeichnet werden, hängt von der Variable difficulty ab, welche, mit einigen Ausnahmen, nach jeder Welle erhöht wird. Da die Raketen im Originalspiel nur von oben nach unten fliegen, reicht die horizontale Position der Bahnen aus um deren Koordinaten zu bestimmen. Allerdings entwickelt sich dadurch als effizienteste Ausweichtaktik horizontale Bewegungen am unteren Bildschirmrand. Dadurch wurden die vier Bewegungsmöglichkeiten des Raumschiffs nicht ausgenutzt, da es keinen Grund gab den Raketen entgegenzufliegen. Daher fliegen in dem Spiel Rocket Dodge nur Raketen aus Levels eins bis drei wie im Original von oben nach unten. In den Levels vier bis sechs fliegen sie von unten nach oben und in den Folgenden in beide Richtungen. Deshalb muss die Orientierung jeder einzelnen Rakete in die Matrix (is_occupied) übergeben werden. Somit kann sowohl das Modell als auch die y-Ordinate des Objektes entsprechend dieses Parameters gedreht werden. Die Kollision von Raketen und Raumschiff werden 60 mal in der Sekunde in timer() mittels einer Boundingbox überprüft. Das bedeutet, dass alle Objekte vereinfacht durch Rechtecke repräsentiert werden. Berühren sich diese wird eine Kollision registriert. Wird ein Treffer ermittelt, so wird diese Funktion übersprungen, bis eine neue Welle beginnt. Diese Logik war, mit Ausnahme der Betrachtung des Richtungsparameters, bereits Teil des originalen Spiels. Des weiteren wurden Extra-Leben hinzugefügt, wobei diese als Repräsentation die Rakete aus der object_properties Matrix mit dem Index difficulty+1 nutzt. Dafür wird zu Beginn jeder Welle per Zufallsfunktion mit einer Wahrscheinlichkeit von 20% festgelegt ob genau ein Extra-Leben erscheint oder nicht. Diese Wahrscheinlichkeit gilt für den Schwierigkeitsgrad Mittel und ändert sich für andere Schwierigkeitsgrade. Die Logik zum Aufsammeln der Leben basiert auf der Boundingbox der Raketen,jedoch mit dem Unterschied, dass die Extra-Leben durch ein kleines Quadrat dargestellt werden.
Nach jeder Welle wird nicht nur der Integer difficulty, sondern auch die Anzahl der überstandenen Wellen innerhalb eines Levels (level_index) erhöht. Erreicht dieser nach Ende einer Welle seinen Maximalwert, festgelegt durch MAX_WAVE, so wird das Level inkrementiert und level_index zurückgesetzt. Somit hat der Spieler das nächste Level erreicht. Die Erhöhung von difficulty findet nicht linear statt, sondern flacht innerhalb jedes Dreierblocks (1 – 3, 4 – 6, 7 – 9) ab. Des Weiteren wird difficulty bei jedem Übergang von einem Level-Block in den nächsten verringert. Dies erfolgt aus der Absicht den Spieler mit der Veränderung des Raketenverhaltens nicht zu überfordern.

3.4 Darstellung

Die Bilddarstellung wird mithilfe der GLUT-Funktion glutDisplayFunc() dargestellt. Als Übergabeparameter wird eine Callback-Funktion benötigt, die hier display() heißt. Der Aufruf dieser Funktion geschieht 60 Mal pro Sekunde um konstante 60 Bilder pro Sekunde (engl. frames per second (FPS)) zu haben und somit ein gleichbleibendes Spielerlebnis auf unterschiedlicher Hardware zu gewährleisten. Für weitere Informationen zum Programmablauf vergleiche Kapitel 3.2.

Die display() Funktion besteht aus folgenden Bestandteilen:

- Lichtquellen einschalten
- Overlay (Score, Hits etc.) anzeigen
- You Won! Schriftzug, wenn gewonnen
- Pauseanzeige bei Pause
- Sternenhintergrund
- jedes einzelne Level (vgl. Kapitel 5.2.3)
- das Raumschiff
- die Raketen
- das Leben

Die Evaluation des Levels ist im Kapitel 3.3 nachzulesen. Dennoch wird mithilfe eines switch-case Statements die Darstellung der Level geändert.

3.5 Grafiklisten

Unter Grafiklisten versteht man die Zusammensetzung mehrerer Vektoren zu Flächen bzw. Körpern. Gruppiert werden sie zu einer Liste, die daraufhin gerufen werden können um diese Fläche bzw. Körper darzustellen. Folgende Listen existieren in Rocket Dodge:

- Planet
- Rakete
- Leben
- Raumschiff

All diese Objekte wurden fest implementiert und nicht als externe Ressource geladen. Dies wäre durchaus möglich, aber die Zeit zur Erarbeitung hätte nicht gereicht. Um die Erstellung eines solchen 3D-Modells zu realisieren ist ein räumliches Vorstellungsvermögen hilfreich.
Der Planet ist eine Kugel erstellt durch die Glut-Funktion glutSolidSphere(). Neben dem Radius kann auch die Farbe des Planeten und deren Farbeigenschaften geändert werden.
Die Raketen bestehen aus mehreren Körpern. U.a. werden Körper wie Zylinder (gluCylinder()) oder Kegel (glutSolidCone()) zur Darstellung genutzt.
Das Leben Besteht aus zwei Quadern die fest mithilfe von GL_Quads implementiert wurden. Die einzelnen Seiten der Quader wurden verschiedenfarbig markiert. Somit kann ein Kontrast entstehen, wenn sich der Körper dreht. Um auch ohne Lichtquelle das Leben optimal sehen zu können, wurde die Beleuchtung für den Körper ausgeschaltet. Damit wird eine maximale Helligkeit für das Objekt erreicht.
Das Raumschiff ist das komplexeste Modell. Es wird aus mehreren Flächen zusammengesetzt. Dreiecke (GL_TRIANGLES), Vierecke (GL_QUADS) und GL_QUAD_STRIPS (eine Darstellungsmöglichkeit von mehreren zusammenhängenden Vierecken) werden genutzt um das Raumschiff zu modellieren.

4 Projektdokumentation

In diesem Teil wird auf die Erarbeitung des Projekts eingegangen. Auf den Aufbau, die Meilensteine, die zeitliche Planung, genutzte Hilfsmittel und die Arbeitsweise in der selbstständigen Arbeit und im Team wird eingegangen.

4.1 Genutzte Hilfsmittel

Für die Erarbeitung dieses Projekts wurden folgende vier primären Hilfsmittel genutzt:

1. Git
2. GitHub
3. Discord
4. Whatsapp

Git ist eine Software zur verteilten Versionsverwaltung von Dateien (vgl. [git]) und wird meistens in Debian Systemen bereits vorinstalliert geliefert. Die Nutzung von Git ist äußerst wichtig um ein paralleles Arbeiten von Mitgliedern an verschiedenen Themen zu ermöglichen. Damit können Zweige (engl. branches) erstellt werden und jeder kann für sich arbeiten. Zum Schluss kann die Arbeit fusioniert (engl. merged) werden um das finale Produkt zu erhalten.
GitHub ist ein netzbasierter Dienst zur Versionsverwaltung für Software-Entwicklungsprojekte (vgl. [github]). GitHub basiert auf git und wird genutzt um die Versionsverwaltung Betriebs- und Standortunabhängig durchzuführen. Genutzt wurden u.a. folgende Features:

- Versionskontrolle
- Merges
- Branches

Discord ist ein kostenloses Kommunikations- und Streaming-Portal für Gamer. Es wurde zur internen Kommunikation und für Peerprogramming oder Issuesolving via Streaming genutzt.
Whatsapp ist ein online Messager und wurde zur Organisation und internen Kommunikation genutzt.

4.2 Arbeitsaufteilung

Um die Arbeit aufteilen zu können, muss man die Ziele klar definieren. Die in Tabelle 2 aufgezählten Ziele wurden gesetzt um das Spiel erfolgreich zu vollenden.

Tab. 2: Ziele und deren Aufteilung

Ziel	Bearbeitet von
1-Kreieren eines Sternenhintergrunds	Dennys-Daniel Vogt
2-Animation bei Kollision	Dennys-Daniel Vogt
3-Raketen aus anderen Richtungen	Lukas Hein
4-Erstellen von 3D-Modell „Leben“	Dennys-Daniel Vogt
5-Erstellen der Logik beim Einsammeln des Lebens	Lukas Hein
6-Animation bei Erhaltung eines Lebens	Dennys-Daniel Vogt

7-Erstellen von 10 verschiedenen Levels	Lukas Hein, Dennys-Daniel Vogt
8-Erstellen von Lichtquellen und deren Eigenschaften in den verschiedenen Levels	Lukas Hein, Dennys-Daniel Vogt
9-Erstellen von „You Won!“ Schriftzug, und definieren wann gewonnen	Dennys-Daniel Vogt
10-Implementierung von WASD zur Kontrolle des Schiffs	Dennys-Daniel Vogt
11-Implementierung einer Pause-Funktion	Lukas Hein, Dennys-Daniel Vogt
12-Implementierung von verschiedenen Schwierigkeitsstufen	Lukas Hein
13-Terminalinteraktion beim Start des Spiels zum setzen eines Schwierigkeitsgrades	Dennys-Daniel Vogt
14-Ingame-Option zum Wechseln der Schwierigkeit	Lukas Hein
15-Integrierung eines Textur-Loaders	noch offen
16-Integrierung eines OBJ-Loaders	noch offen

4.3 Zeitplanung und Meilensteine

Folgend wurden die Ziele aus Tabelle 2 zu Meilensteinen zusammengefasst, dessen Zieldatum definiert und das Enddatum (vgl. Tab. 3). Das Zieldatum jeden Meilensteins definiert das geplante Abschlussdatum, wohingegen das Enddatum das wahre Datum des Abschlusses ist.

Tab. 3: Zeitplanung und Meilensteine

Meilenstein	Ziele	Zieldatum	Enddatum
Hintergrund	1	25.08.20	03.09.20
Extra-Leben	4, 5, 6	20.09.20	22.09.20
Level	7,8,9	15.09.20	10.09.20
Raketen	2,3	20.09.20	17.09.20
Steuerung	10,11	10.09.20	17.09.20
Schwierigkeitsgrade	12,13,14	25.09.20	22.09.20

Das Projekt wurde am 19.08.2020 begonnen mit dem ersten Ziel des Hintergrunds und der Einarbeitung in den bereits bestehenden Code. Das Zieldatum für das Spiel wurde auf den 20.09.2020 gesetzt. Dies wurde um zwei Tagen überschritten (vgl. Tab. 3). Die restliche Zeit sollte für die Ausarbeitung dieser Arbeit und des Vortrages genutzt werden. Bedauerlicherweise wurden die Ziele 15 und 16 (vgl. Tab. 2) aus zeitlichen Gründen nicht integriert. Dennoch haben diese nach dem Abschluss des Projekts eine hohe Priorität.

4.4 Arbeitsweise

Die Erarbeitung des Spiels sollte größtenteils selbstständig erfolgen. Jeder sollte sich Ziele heraus nehmen und diese bearbeiten. Durchaus kam es vor, dass man sich gegenseitig geholfen hat um diverse Fehler zu vermeiden und das bestmögliche Resultat zu erlangen. Nachdem ein Ziel bearbeitet wurde, wurde es auf den jeweiligen Branch in GitHub gepushed. Dieser Code wurde von Teammitgliedern kontrolliert bevor es auf den Master-Branch gemerged wurde (zur besseren Illustration vgl. Abb. 3).

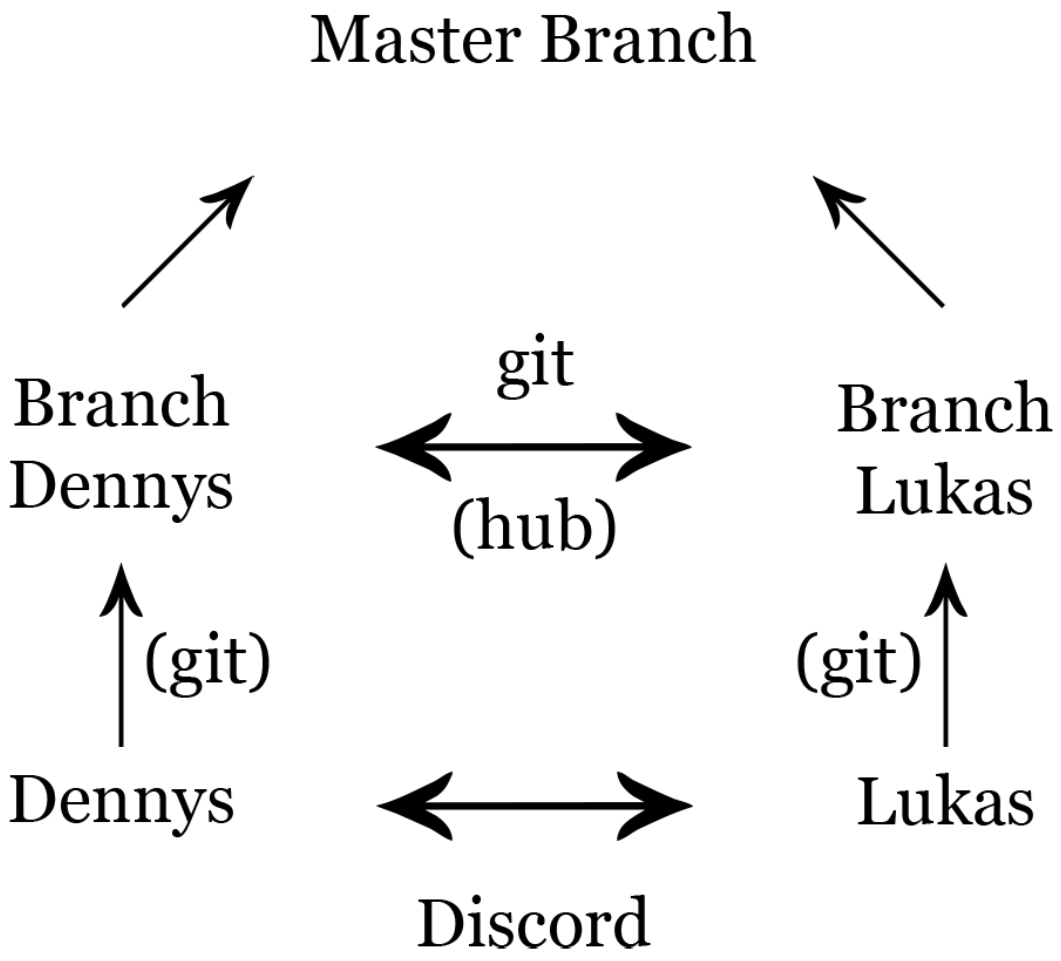


Abb. 3: Arbeitsweise im Team

5 Benutzerhandbuch

5.1 Vor dem Spiel

Falls noch nicht kompiliert, muss der Programmcode des Spiels zunächst über die enthaltende Makefile übersetzt werden. Dafür werden der C++ Compiler g++ und das Build-Management-Tool make, sowie die genutzte Bibliothek GL/freeglut.h benötigt. Es ist ebenfalls möglich ohne das Build-Management-Tool make das Spiel zu kompilieren. Dafür werden folgende Terminaleingaben im Spielordner benötigt:

1. g++ -Wall -g -c rocket_dodge.cpp
2. g++ -Wall -g rocket_dodge.o -o rocketdodge -lGL -lGLU -lglut

Man kann das Spiel nun entweder über die Konsole, oder über die ausführbare Datei starten. Um das Programm über die Konsole zu starten, navigiert man in das Verzeichnis in der die Datei liegt und führt den Befehl ./rocketdodge aus. In beiden Fällen wird Rocket Dodge im Standardschwierigkeitsgrad Mittel gestartet. Beim Aufruf des Spiels über die Konsole kann man außerdem über einen Zusatzparameter bestimmen welcher Schwierigkeitsgrad gewählt wird. Folgende Eingaben sind möglich:

- Mittel: ./rocket-dodge -m
- Einfach: ./rocket-dodge -e
- Schwer: ./rocket-dodge -h
- Extrem: ./rocket-dodge -k

5.2 Spielprinzip und Aufbau

Ziel des Spielers in Rocket Dodge ist es mit seinem Raumschiff so lange wie möglich Raketen auszuweichen, um so eine höchstmögliche Punktzahl zu erreichen. Hat der Spieler die maximale Wellenanzahl in Level zehn überstanden, so hat er das Spiel gewonnen (siehe Abb. 4). Fallen die Trefferpunkte seines Raumschiffs auf null hat er verloren (siehe Abb. 5). Nachdem das Spiel gewonnen wurde, kann jedoch weitergespielt werden um eine persönliche Bestleistung zu erreichen.

5.2.1 Overlay

Zum Overlay gehören das Scoreboard (siehe Abb. 6 (1)), welches (von oben) die Punkte , die übrigen Trefferpunkte und das momentane Level anzeigt, in dem sich der Spieler befindet, sowie der Schwierigkeitsgrad (siehe Abb. 6 (2)), auf dem gespielt wird. Nach jeder Welle werden die Punkte um 100 * Anzahl der Raketen (siehe Abb. 6 (4)) erhöht.

5.2.2 Objekte

Das Raumschiff (siehe Abb. 6 (3)) wird durch den Spieler gesteuert. Er muss versuchen den Raketen (siehe Abb. 6 (4)) auszuweichen und sollte, wenn möglich, Extra-Leben (siehe Abb. 6 (5)) einsammeln um seine Überlebenschancen zu erhöhen. Kollidiert man mit einer Rakete, so erscheint ein rotes Blinken und das Raumschiff verliert einen Trefferpunkt (siehe Abb. 7a). Es wird pro Welle maximal eine Kollision registriert. Wird ein Extra-Leben eingesammelt, so

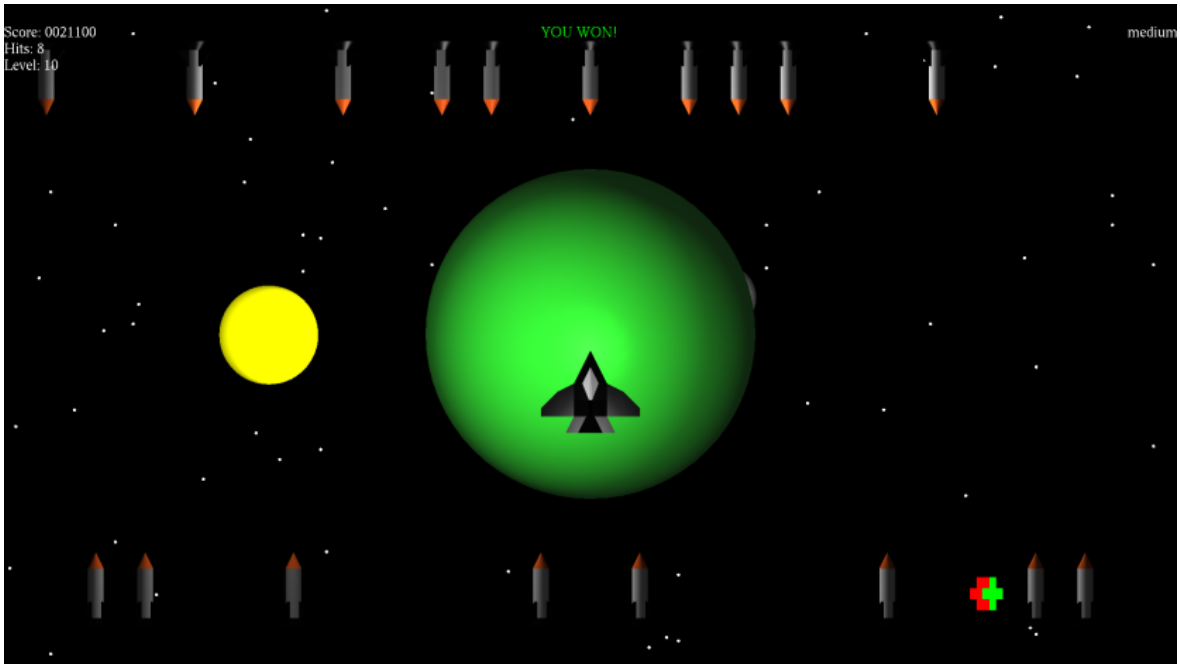


Abb. 4: Siegesanzeige

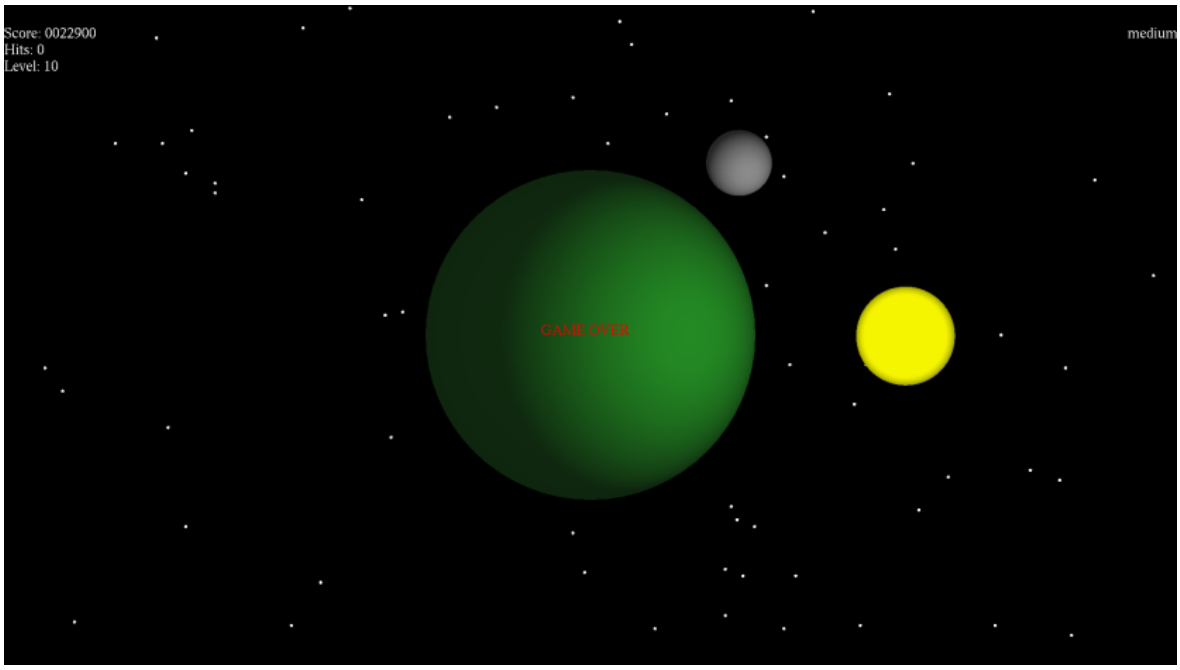


Abb. 5: Game Over Anzeige

blinkt ein grünes Licht und die Trefferpunkte werden um eins erhöht (siehe Abb. 7b). Man kann ein Maximum von neun Leben ansammeln.

Die Objekte im Hintergrund, wie z.B. Planeten (siehe Abb. 6 (6)) beeinflussen das Spielgeschehen nicht. Die einzige Ausnahme stellt hierbei der große rote Planet in Level acht dar, welcher am linken oberen Bildschirmrand Raketen verdeckt.

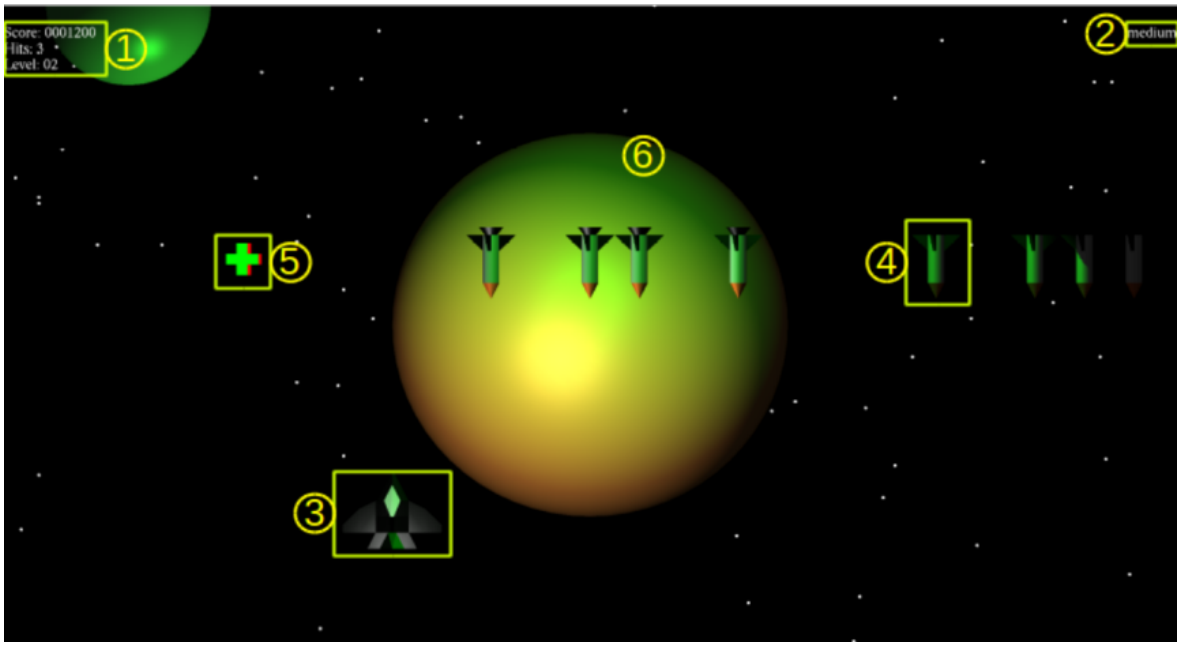


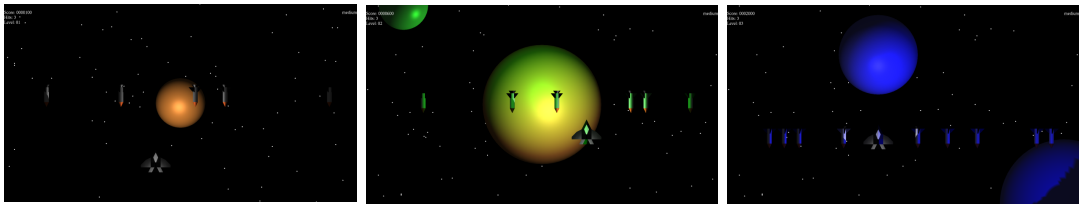
Abb. 6: Overlay

(a) Kollision **(b)** Extra-Leben

Abb. 7: Animation

5.2.3 Levels

In den Levels 1–3 fliegen die Raketen von oben auf den Spieler zu (siehe Abb. 8).

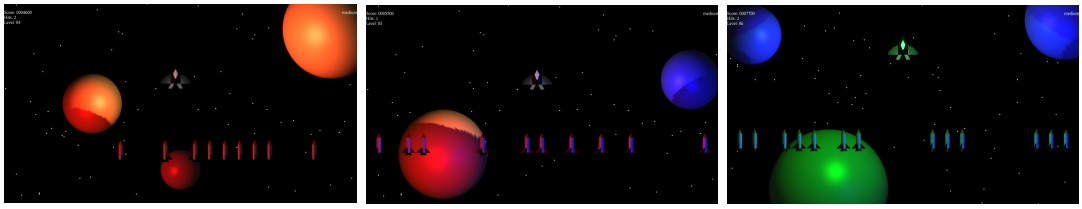


(a) Level 1 **(b)** Level 2 **(c)** Level 3

Abb. 8: Levels 1–3

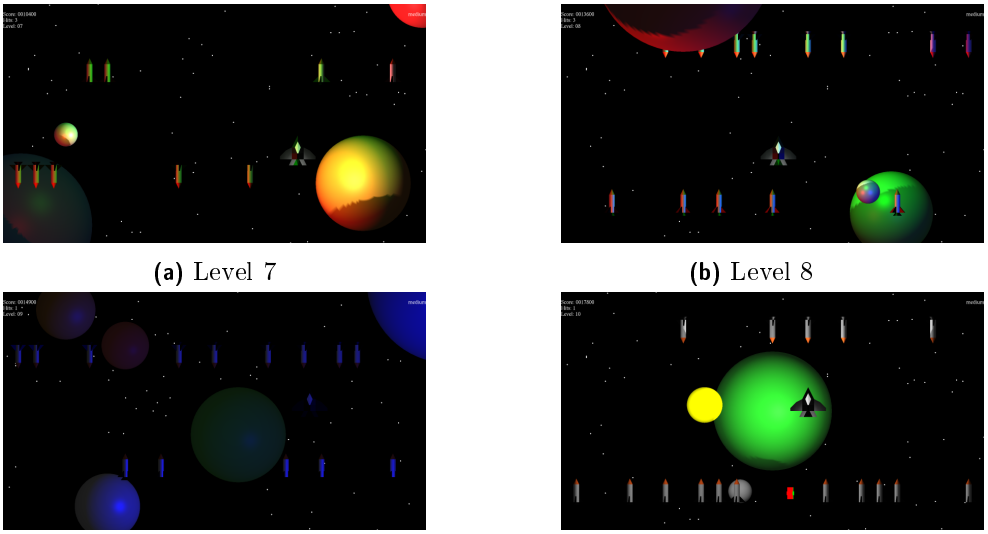
In den Levels 4–6 fliegen die Raketen von unten auf den Spieler zu (siehe Abb. 9).

In den Levels 7–10 fliegen die Raketen von beiden Seiten auf den Spieler zu (siehe Abb. 10).



(a) Level 4 **(b)** Level 5 **(c)** Level 6

Abb. 9: Levels 4–6



(a) Level 7 **(b)** Level 8

(c) Level 9 **(d)** Level 10

Abb. 10: Levels 7–10

5.2.4 Schwierigkeitsgrad

Die Schwierigkeitsgrade unterscheiden sich in Einfach, Mittel, Schwer und Extrem. Sie beeinflussen die Startanzahl und Geschwindigkeit der Raketen sowie deren Wachstum und das Erscheinen von Extra-Leben. Der Schwierigkeitsgrad kann bei Start des Spiels über die Konsole festgelegt werden, Der Standard-Schwierigkeitsgrad ist Mittel. Im Schwierigkeitsgrad Einfach steigt die Anzahl der Raketen, sowie die Geschwindigkeit dieser langsamer, während sich diese Werte im Modus Schwer schneller vergrößern. Außerdem startet das Spiel im Schwierigkeitsgrad Schwer mit fünf statt einer Rakete. Auf der Stufe Extrem bleiben Anzahl und Geschwindigkeit der Raketen konstant auf dem Maximum. Außerdem erscheinen keine Extra-Leben.

5.3 Steuerung

5.3.1 Raumschiff

Die Steuerung des Raumschiffs erfolgt über die Pfeiltasten bzw. WASD, wobei gilt:

- Norden: oben / W
- Westen: links / A
- Süden: unten / S
- Osten: rechts / D

Des weiteren ist das Raumschiff um seine Längsachse drehbar. Dazu klickt man die linke Maustaste für eine Links- bzw. die rechte Maustaste für eine Rechtsdrehung. Dies ist notwendig, um Raketen auszuweichen, die nur eine Raketenbreite Platz bieten. Der Drehwinkel wird nach jeder überstandenen Welle zurückgesetzt. Dies kann auch durch drücken der mittleren Maustaste manuell geschehen. Die weiteren Steuermöglichkeiten werden folgend aufgelistet:

- Drehung auf die linke Seite: linke Maustaste
- Drehung auf die rechte Seite: rechte Maustaste
- Ausgangsposition: mittlere Maustaste

5.3.2 Spielmenü

Folgend wird die Steuerung des Spielmenüs aufgelistet:

- Pause / Weiter: P
- Neues Spiel: Enter (Während des Spiels oder im Game-Over-Bildschirm)
- Spiel Verlassen: Esc

Die Änderung des Schwierigkeitsgrades im Spiel ist nur im Game-Over-Bildschirm möglich. Mit den folgenden Tasten kann man die Schwierigkeit ändern:

- Einfach: E
- Mittel: M
- Schwer: H
- Extrem: K

6 Zusammenfassung

Das Spiel „Rocket Dodge“ ist ein überarbeitetes Spiel aus 2008. Es handelt sich um ein 3D Weltall-Ausweichspiel im 2D Stil. Das Originalspiel wurde von Samuel Simeonov im Rahmen einer Bewerbung geschrieben.
Der Code wurde in der Sprache C++ im prozeduralen Stil geschrieben. Die externe Bibliothek, die am meisten genutzt wurde ist die GL/freeglut.h Bibliothek. Diese ermöglicht und vereinfacht viele Funktionen für das Erstellen eines Spiels.
Das Spiel besteht aus mehreren Objekten, wobei nur das Raumschiff, das Hauptobjekt, vom Spieler bewegt werden kann. Das Ziel des Spiels ist es den aus verschiedenen Richtungen kommenden Raketen auszuweichen. Auf diesem Weg werden auch Leben zum einsammeln verfügbar sein. Nach Abschluss der zehn verschiedenen Levels und Überleben weiterer Wellen, ist das Spiel gewonnen. Es handelt sich aber um ein open-end-type Spiel, bei dem man bis zu der maximal erreichbaren Punktzahl spielen kann um seinen eigenen Rekord zu brechen. Beim Start des Spiels oder während des Spiels kann der Schwierigkeitsgrad geändert werden. Zu wählen ist dabei zwischen 4 verschiedenen Graden, und zwar Leicht, Mittel, Schwer, und Extrem Schwer.
Innerhalb von ca. 30 Tagen wurde das Spiel aufgearbeitet und verbessert. Dabei wurden Hilfsmittel wie GitHub und Discord genutzt. In GitHub wurden diverse Ziele definiert, die zum erfolgreichen Abschluss des Spiels dienen sollten. Erarbeitet wurden diese Ziele in selbstständiger Arbeitsweise. Falls Hilfe benötigt wurde, hat sich das Team getroffen um das Problem schnellstmöglich effizient zu lösen. Die Erarbeitung des Spiels sollte exakt 30 Tage dauern, leider wurde dies nicht eingehalten und mit einer kleinen Verspätung vollendet. Die verbliebene Zeit wurde zur Erarbeitung der Arbeit und des Vortrags genutzt.
Im Allgemeinen würden wir das Spiel als Erfolg zählen. Bei der Ausarbeitung wurde viel gelernt sowohl über Git/GitHub als auch im Programmiertechnischen. Vor allem das Arbeiten und die Einarbeitung in die externe Bibliothek GL/freeglut.h und den bereits bestehenden Code war interessant und informativ. Persönlich gesehen war das Spiel eine Bereicherung und hat zur Bildung beigetragen.

Abbildungsverzeichnis

1 Rocket Dodge Cover 4
2 Originalspiel aus 2008 5
3 Arbeitsweise im Team 11
4 Siegesanzeige 13
5 Game Over Anzeige 13
6 Overlay 14
7 Animation 14
8 Levels 1–3 14
9 Levels 4–6 15
10 Levels 7–10 15

Tabellenverzeichnis

1 Bibliotheken und deren Funktionen 6
2 Ziele und deren Aufteilung 9
3 Zeitplanung und Meilensteine 10

7 Literaturverzeichnis

git Wikimedia Foundation Inc. (2020, Oktober 7). Git. https://de.wikipedia.org/wiki/Git.

github Wikimedia Foundation Inc. (2020, Oktober 7). GitHub. https://de.wikipedia.org/wiki/GitHub.