



TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG

Die Ressourcenuniversität. Seit 1765.

Fakultät für Mathematik und Informatik
Institut für Informatik

Studienarbeit

OpenGL Game: Achtung, die Kurve!

Simon Al Nomer und Hernán Felipe Valdés González

Bachelor Angewandte Informatik

Matrikel: 64 082
63 952

19. Oktober 2020

Betreuer/1. Korrektor:
M.Sc. Jonas Treumer

2. Korrektor:
M.Sc. Ben Lorenz

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

Simon Al Nomer und Hernán Felipe Valdés González

Zusammenfassung

Implementierung des Spieles "Achtung, die Kurve" in C++ mit OpenGL für die Lehrveranstaltung "Multimedia".

Inhaltsverzeichnis

Zusammenfassung	3
1. Einführung	4
1.1. Spielmechanik	4
2. Softwaredokumentation	5
2.1. user_data_t	5
2.2. player_info	5
2.3. Game	5
2.4. GameState	6
2.5. Model und Mesh	7
2.6. Display	7
2.7. Font	8
3. Zeichnungen	8
3.1. Player	8
3.2. Linien	9
4. Bewegung	12
4.1. Startgeschwindigkeit	12
4.2. Im laufenden Spiel	12
5. Kollisionen	12
6. Projektdokumentation	13
7. Benutzerhandbuch	13
7.1. Installation	13
7.2. Spielanweisungen	14
Literatur	14
A. Screenshots	15

1. Einführung

Achtung, Die Kurve ist ein Multiplayer-Spiel, welches im Jahr 1995 von Filip Oščádal und Kamil Doležal in DOS entwickelt wurde. Im Jahr 2010 wurde eine neue Version des Spiels unter dem Namen “Achtung, die Kurve! Flash Remake” veröffentlicht. Diese Version ist mit Adobe Flash von Geert van den Burg entwickelt worden. Das Spiel kann von mehreren Spielern an einem Computer online gespielt werden.

Nachdem das Spiel großen Zuspruch fand, beschloss van den Burg, sich mit Robin Brouns zusammenzuschließen und eine Fortsetzung zu entwickeln, welche 2011 unter den Namen Curve Fever (Originaltitel “Achtung, die Kurve! 2”) veröffentlicht. Es kann online und mit Spielern aus dem Netz gespielt werden.

In den folgenden Jahren wurden dem Spiel immer weiter neue Features hinzugefügt wie zum Beispiel verschiedene Boni, die Möglichkeit, in einem Team zu spielen, oder das Bestehen einer Rangliste.

2015 sammelte van den Burg ein Team um sich, um eine neue Version des Spiels in HTML5 zu implementieren, welche im September 2016 auf dem Markt kam.

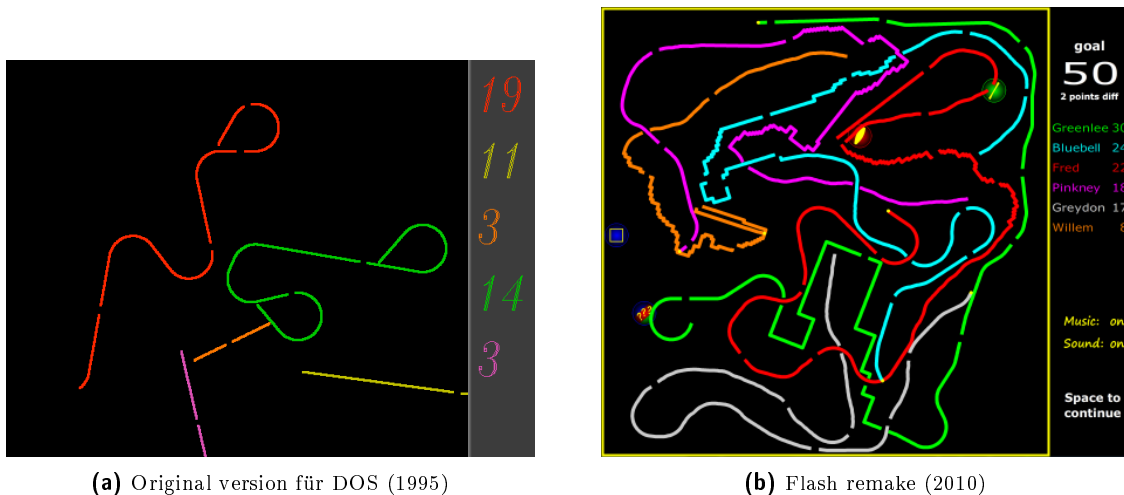


Abb. 1: 2 Versionen, die als Basis für das Spiel genommen wurden

Die Implementierung unseres Spiels basiert auf die Version von 1995 in Kombination mit dem Stil der Flash Version.

1.1. Spielmechanik

Das Spiel können bis zu 6 Spieler zusammen an einem Bildschirm und mit einer Tastatur spielen. Jeder Spieler besitzt zwei vorbestimmten Tasten, die rechts und links symbolisieren. Um das Spiel zu starten, müssen mindestens 2 Spieler spielen. Das Ziel des Spiels ist es, dass ein Spieler solange wie möglich am Leben bleibt.

Jeder Spieler wird durch einen Kreis dargestellt, der mit jeder Bewegung einen Pfad mit seiner Farbe hinterlässt. Der Spieler kann sich nur nach rechts oder links drehen und damit die Richtung seiner Bewegung ändern. Wenn der Spieler nicht reagiert, bewegt er sich weiter in die gleiche Richtung.

Ein Spieler verliert, wenn er gegen seine eigene Linie, die Linie anderen Spieler oder den Rand stößt. Das Spiel ist zu Ende, wenn nur noch ein Spieler am Leben ist.

2. Softwaredokumentation

Unseres Projekt hat den Code als Basis, der zusammen während der Vorlesung erstellt wurde. Dieser Code wurde in C++ migriert. Die Entscheidung, das Spiel in C++ statt in C zu programmieren, lag insbesondere an der Standardbibliothek von C++ und deren implementierten Daten-Strukturen sowie die Eigenschaft mit Klassen zu programmieren.

Für die Softwarearchitektur wurde das State Pattern implementiert, um einen endlichen Automaten zu simulieren. Die Änderungen der Zuständen sind in der Klasse *Game* durchgeführt.

Das Spiel wurde im Sinne von Szenen programmiert. Die Klasse *Game* verhält sich als eine Szene sowie als eine Szene-Manager. Die anderen zwei Szenen sind *Menu* und *GameOver*.

In dem Spiel wird zwischen User und Player differenziert. Das Spiel besitzt nur ein User, der die Kontrolle über den Computer besitzt und mehrere Player, welche die Kreise kontrollieren.

2.1. user_data_t

Die Daten des laufenden Spiels, sowie die Anzahl an Spieler und ihren Punkten werden in einem Struct gespeichert, der dann in GLFW zur Verfügung gestellt wird. Dieser Struct heißt *user_data_t*.

```
1 typedef struct {
2     int window_width;
3     int window_height;
4     GameState game_state;
5     std::vector<player_info_t>* player_info;
6 } user_data_t;
```

2.2. player_info

Alle Player werden am Anfang des Programmes in main.cpp addiert und dann im Menü als aktiv gesetzt.

```
1 typedef struct {
2     bool is_active;
3     int id;
4     std::string name;
5     std::string menu_text;
6     Control control;
7     std::array<GLubyte, 3> color;
8     glm::vec3 menu_color;
9     int score;
10 } player_info_t;
```

2.3. Game

Die zentrale Klasse des Projektes ist *Game*. In dieser Klasse werden die Zustände des Automaten geändert und unterschiedliche Klassen initialisiert und geteilt. In der Abb. 2 steht die Klassen, die anschließend im *Game* initialisiert werden. Ein Beispiel ist die Klasse *Font*, die für die Erzeugung und für die Zeichnung von Texten verantwortlich ist. Sie wird nur einmal in *Game* initialisiert und im Folgenden mit anderen Klassen geteilt.

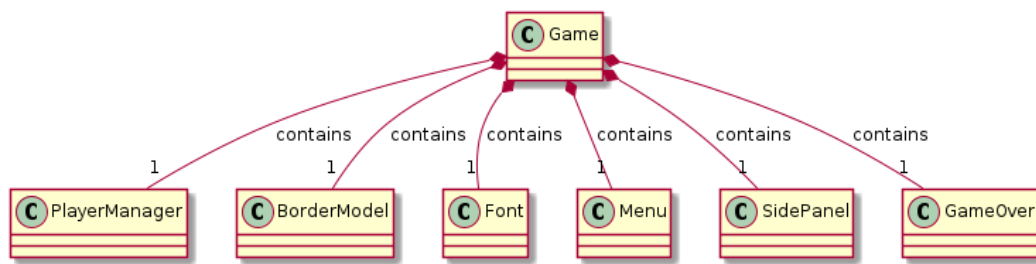


Abb. 2: UML Diagramm der Klasse Game

2.4. GameState

In der Abb. 3 wird der Automat des Spieles dargestellt. Der Startzustand des Automaten ist **GAME_MENU**.

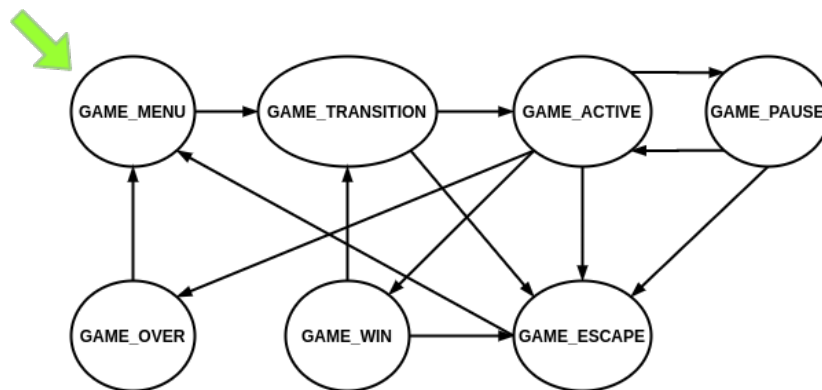


Abb. 3: Darstellung des Zustandsautomaten

GAME_MENU Es wird das Menü für den Auswahl von Spielern gezeigt. Es kann nur zu dem nächsten Zustand übergegangen werden (**GAME_TRANSITION**), in dem mindestens 2 Spieler ihre Anwesenheit bestätigen und dann die SPACE-Taste gedrückt wird.

GAME_TRANSITION Das Spiel ist bereit, zu starten. Alle Spieler sind auf ihren Plätzen und ihre Positionen werden mit einem gefärbten Kreis gezeichnet. Um das Spiel zu starten, muss die SPACE-Taste gedrückt werden und der neuer Zustand wird **GAME_ACTIVE**. Um zurück ins Menü zu gehen, muss die ESCAPE-Taste gedrückt werden und der neuer Zustand wird **GAME_ESCAPE**.

GAME_ACTIVE Das Spiel läuft und die Spieler dürfen anderen umschließen und in die Irre führen. Es kann in jedem Moment die SPACE-Taste gedrückt werden, um das Spiel zu pausieren und der neuer Zustand wird **GAME_PAUSE**. Es kann auch mit der ESCAPE-Taste das Spiel abgebrochen und zurück ins Menü gelangt werden. Dann wird der neue Zustand **GAME_ESCAPE**. Wenn nur ein Spieler auf dem Feld übrig ist, wird der Zustand automatisch zu **GAME_WIN** oder **GAME_OVER** übergehen, abhängig ob die maximale Punktzahl erreicht wurde oder nicht.

GAME_PAUSE Das Spiel ist pausiert. Mit der SPACE-Taste kann zurück zu GAME_ACTIVE und mit der ESCAPE-Taste zu GAME_ESCAPE übergegangen werden.

GAME_ESCAPE ist ein Zwischenzustand, welcher die Daten wieder ins Default bringt. Es wird direkt zu GAME_MENU gegangen.

GAME_WIN ist ein Zwischenzustand, der den Gewinner der Runde anzeigt (mit einem blinkenden Namen). Nach ein paar Frames wird direkt zu GAME_TRANSITION gegangen, um eine neue Runde zu starten.

GAME_OVER Es wird eine Liste mit der gesamten Punktzahl angezeigt. Wird die SPACE-Taste gedrückt, erscheint das GAME_MENU und alles startet von vorne.

2.5. Model und Mesh

Die Klassen *Model* und *Mesh* sind zwei abstrakte Klassen, welche als Basisklasse angewandt werden.

Mesh Ein *Mesh* enthält die Punkte, die gezeichnet werden, das Vertex Array Object (VAO) und das Vertex Buffer Object (VBO). Die Klassen *BorderMesh*, *PlayerMesh* und *LineMesh* erben vom *Mesh*. Diese Klassen müssen die Funktion "draw" deklarieren, um die Punkte zu zeichnen.

Model Ein *Model* enthält ein *Mesh* und die ID des *Shader*, welches angewandt wird. Ein *Model* interagiert mit *Game*. In dieser Klasse werden die Positionen und Transformation durch die Funktion "update" informiert.

2.6. Display

Die Klasse *Display* initialisiert und besitzt eine Instanz von GLFWwindow. Ein Pointer des Fensters wird dann an die Klasse *Game* übergeben. Anschließend schließt *Display* am Ende des Programms auch das Fenster (beenden).

Das Fenster hat eine ursprüngliche Ratio von 1:1 mit einer Größe von 700x700 Pixel. Das Fenster behält sein Aspect Ratio, indem es immer den kleinsten Wert zwischen Höhe und Breite nimmt und sich dann in der Mitte positioniert. Ist zum Beispiel die Höhe größer als die Breite, zentriert das Fenster sich in der Mitte von der Höhe mit der Größe von der Breite. Der Rest wird mit einem schwarzen Padding gefüllt.

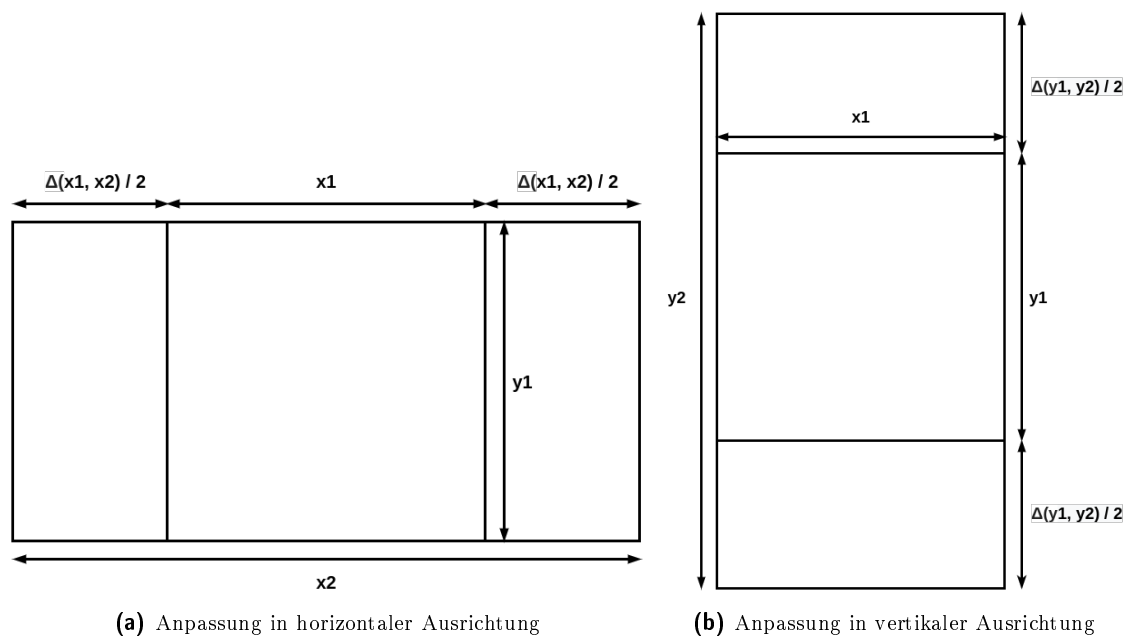


Abb. 4: Anpassungen nach Größenänderung des Fensters

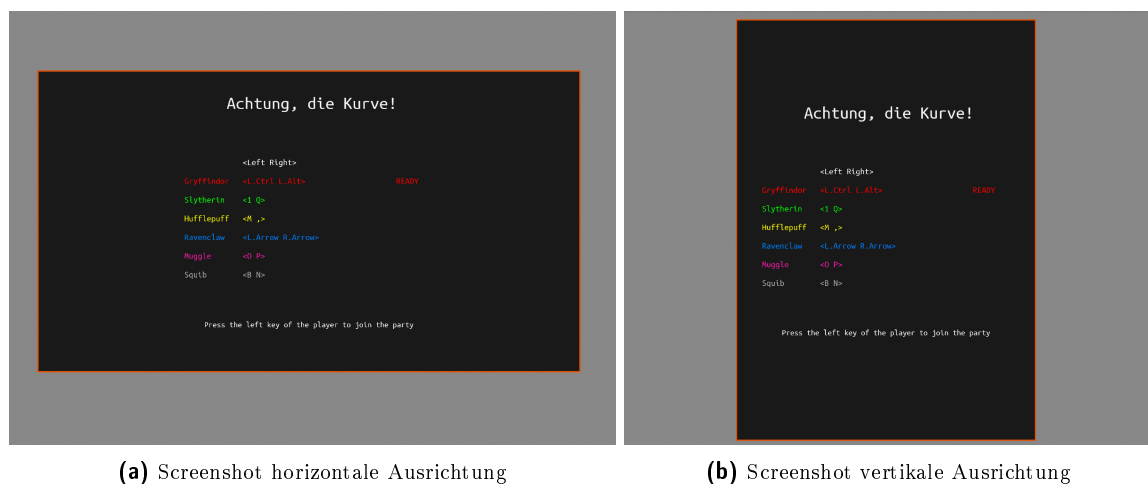


Abb. 5: Screenshots von dem Beispiel der Ausrichtung

2.7. Font

Für die Erstellung von Texten ist in Anlehnung an einem Tutorial [1]. Der Code wurde für das Projekt adaptiert. Die Lösung basiert auf die Bibliothek FreeType. FreeType ist in der Lage, Schriftarten zu laden, Bitmap zu rendern und unterstützt mehrere Text-Operationen.

3. Zeichnungen

3.1. Player

Wie bereits anfangs erwähnt, wird jeder Spieler durch einen Kreis dargestellt. Die Erstellung der Kreise erfolgt durch Randpunkte des Kreises, welche den gleichen Radius zum Mittelpunkt haben. Diese Randpunkte werden durch die Funktion OpenGL Flag `GL_TRIANGLE_STRIP`

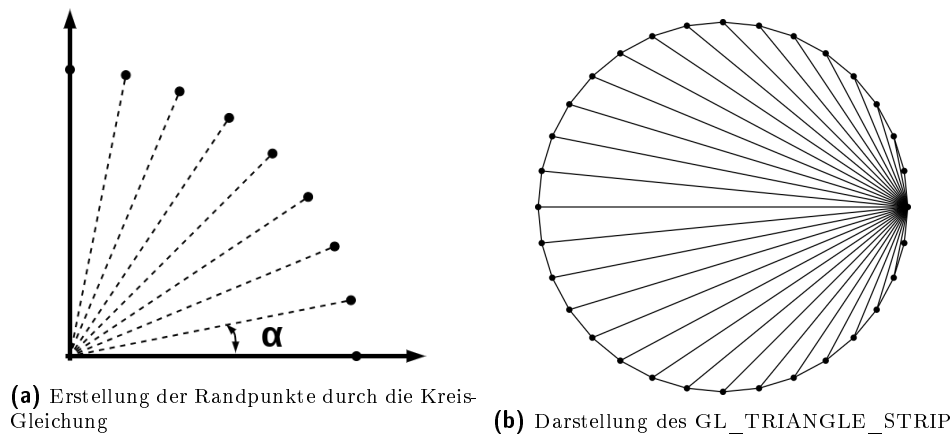


Abb. 6: Erstellung eines Kreises

verbunden. Eine Darstellung des Prozesses ist in der Abb. 6 gezeigt.

3.2. Linien

Für das Zeichnen dicker Linien wurde ein Algorithmus entwickelt, der in der Abb. 7 dargestellt ist.

Um den Winkel zwischen zwei Vektoren zu berechnen wurde die folgende trigonometrische Eigenschaft angewandt:

$$\cos(\theta) = \frac{\vec{v}_1 \cdot \vec{v}_2}{\|\vec{v}_1\| \|\vec{v}_2\|} \quad (1)$$

Für die Drehrichtung der Linie wurde die z-Komponente des Kreuzproduktes zwischen \vec{v}_1 und \vec{v}_2 analysiert.

$$\vec{v}_3 = \vec{v}_1 \times \vec{v}_2 \quad (2)$$

Es gibt 3 Fälle:

1. $\vec{v}_{3z} > 0$: Die Linie dreht sich nach links
2. $\vec{v}_{3z} < 0$: Die Linie dreht sich nach rechts
3. $\vec{v}_{3z} = 0$: Die Linie ist gerade

Wenn die neuen Punkte L und R berechnet wurden, können sie in dem LineMesh hinzugefügt werden. In der Abb. 8 ist eine graphische Darstellung des Algorithmus dargestellt.

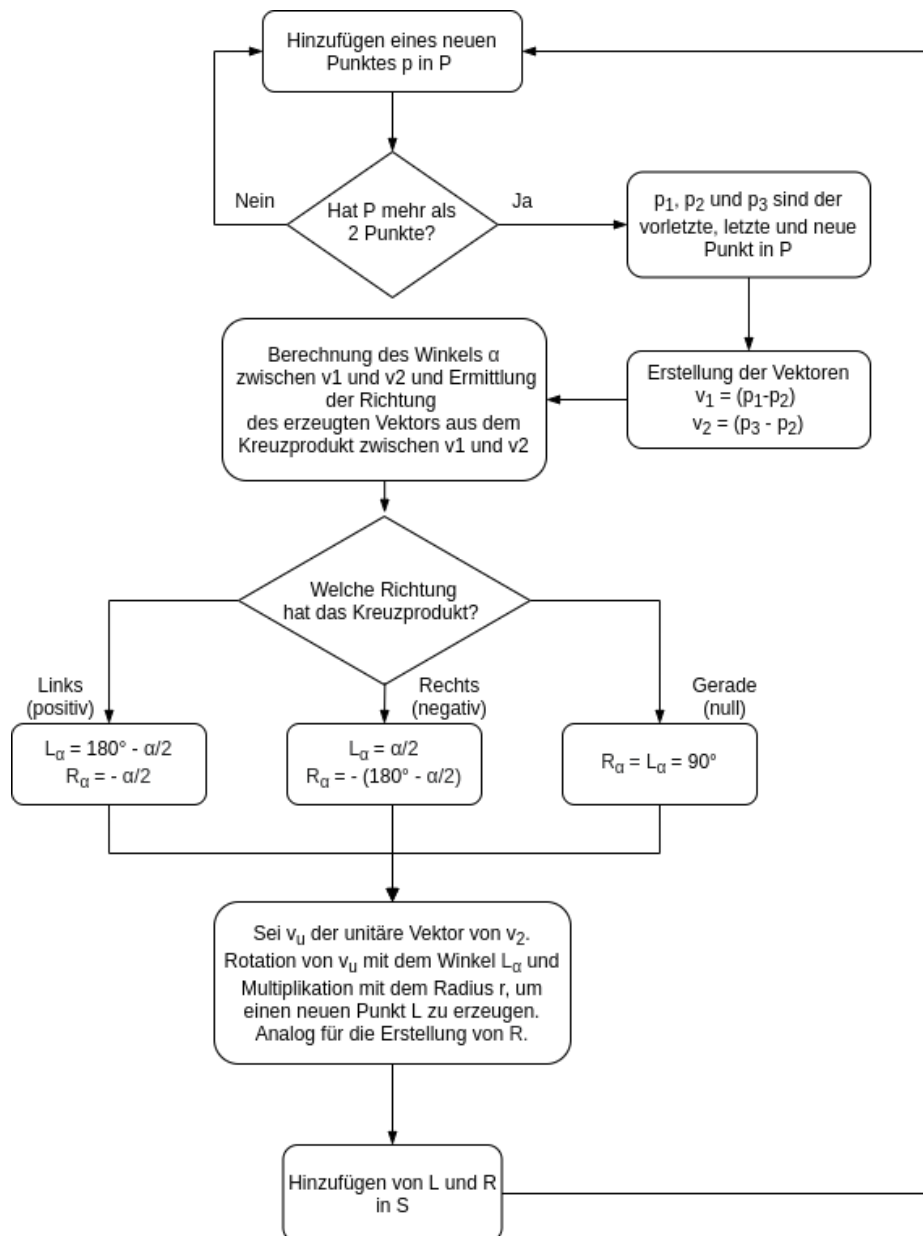


Abb. 7: Flow-Diagramm des Algorithmus zur Linienerzeugung

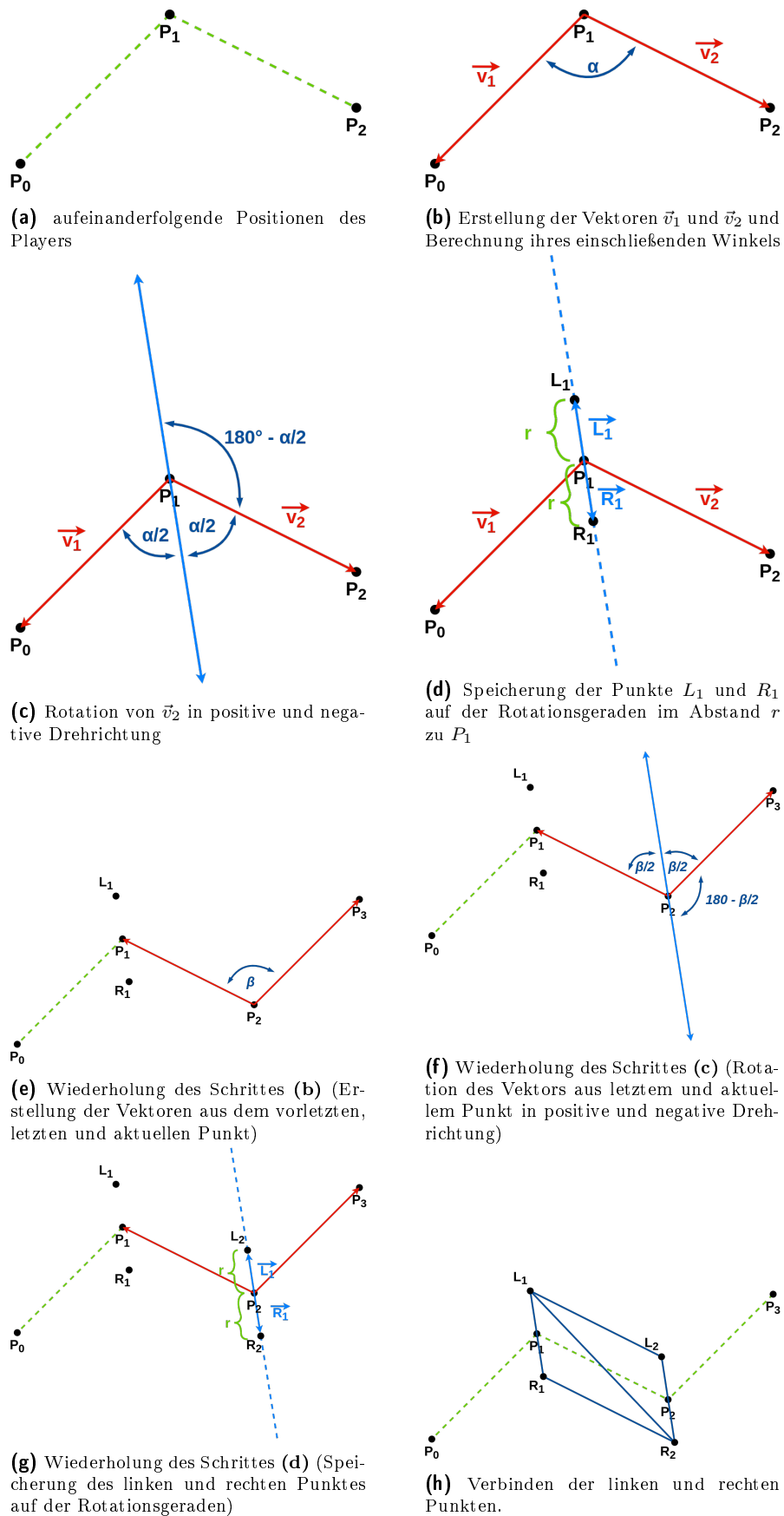


Abb. 8: Beispiel der Erzeugung eines Linien-Segments

4. Bewegung

4.1. Startgeschwindigkeit

Alle *PlayerModels* starten mit dem gleichen Geschwindigkeitsvektor v .

$$v = \begin{bmatrix} 0.09 \\ 0 \end{bmatrix} \quad (3)$$

Wenn ein Player initialisiert ist, wird ein randomisierter Winkel θ erzeugt. Anschließend wird der Vektor v in z rotiert, um einen neuen Vektor zu erzeugen.

$$\begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} \quad (4)$$

Der Vektor v wird durch v' ersetzt.

4.2. Im laufenden Spiel

In jedem Frame gibt es 3 Fälle für alle aktiven Player:

Der Player drückt keine Steuerungstaste Der Geschwindigkeitsvektor v bleibt gleich und der Player bewegt sich in die gleiche Richtung.

Der Player drückt die linke/rechte Taste Der Geschwindigkeitsvektor v wird wie in der Gleichung 4 rotiert.

Der Player ist tot Der Player wird nicht mehr geupdated, der Geschwindigkeitsvektor bleibt gleich aber er wird sich nicht bewegen.

5. Kollisionen

Vereinfachend werden alle Punkte der Linien als Kreisen betrachtet. Da alle Elementen im Spiel als Kreisen betrachtet werden, kann man eine Kollision detektieren, wenn der Abstand zwischen beiden Kreisen kleiner als die Summe ihrer Radien sind.

Seien C_1 und C_2 die Zentren zweier Kreise. Der Abstand zwischen den Zentren wird mit der folgenden Gleichung berechnet:

$$d = \sqrt{(C_{1x} - C_{2x})^2 + (C_{1y} - C_{2y})^2} \quad (5)$$

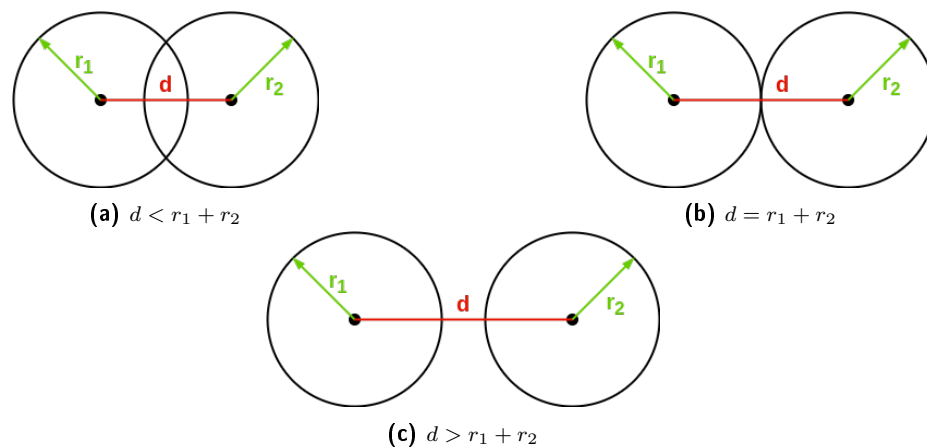


Abb. 9: Kollisionsfälle

In jedem Frame wird der Abstand zwischen jedem Player und jedem Punkt, sowie der Abstand zwischen jedem Player und dem Border berechnet. Der Prozess ist in der Abb. 10 dargestellt.

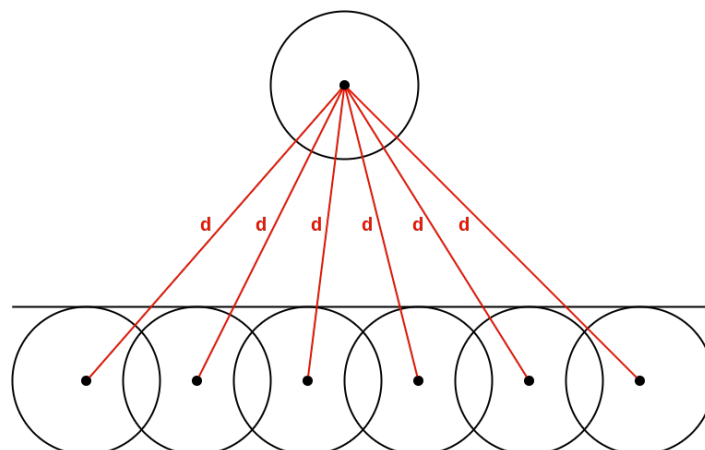


Abb. 10: Der Player überprüft seinen Abstand zu allen Punkten der Linie

6. Projektdokumentation

Für die Entwicklung des Projektes wurde die IDE Visual Studio Code (VSCode) ausgewählt. VSCode besitzt ein kostenloses Plugging, welches das Pair Programming erlaubt, das LiveShare heißt. Im Kombination mit Skype konnte das Projekt gleichzeitig entwickelt werden.

Für den Code wurde ein Github Repository erstellt, welches es erlaubt hat, asynchron zu arbeiten.

7. Benutzerhandbuch

7.1. Installation

Das Projekt befindet sich in GitHub auf dem Repository <https://github.com/h-valdes/kuve>.

Installieren Sie die Dependencies:

1. GLFW
2. FreeType2

Clonen Sie das Repository:

```
$ git clone https://github.com/h-valdes/kurve.git
```

Erstellen Sie einen neuen Build Ordner auf dem Ordner des Repository und kompilieren Sie das Projekt:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Um das Projekt zu starten:

```
$ ./kurve
```

7.2. Spielanweisungen

Es gibt 6 mögliche Spieler. Jeder Spieler hat 2 vordefinierten Steuerungstasten:

Name	Links	Rechts
Gryffindor	L.Ctrl	L.Alt
Slythering	1	Q
Hufflepuff	M	,
Ravenclaw	L.Arrow	R.Arrow
Muggle	O	P
Squib	B	N

Tab. 1: Steuerungstasten der Spieler

Um das Spiel zu starten, müssen mindestens 2 Spieler ihre Anwesenheit bestätigen. Die Bestätigung wird mit ihrer respektiven linken Steuerungstaste durchgeführt. Zum abbrechen muss die rechte Steuerungstaste gedrückt werden.

Wenn nur ein Spieler übrig ist, wird die Runde beendet. Der Name des Gewinners blinkt kurz und nach dem Blinken kann die nächste Runde mit der SPACE-Taste gestartet werden.

Wenn ein Spieler die maximale Punktzahl erreicht hat, ist das Spiel zu Ende.

Literatur

- [1] Joey de Vries. *Learn OpenGL*. URL: <https://learnopengl.com/> (besucht am 10.09.2020).

A. Screenshots



Abb. 11: Menü



Abb. 12: Bestätigung der Anwesenheit der Spieler

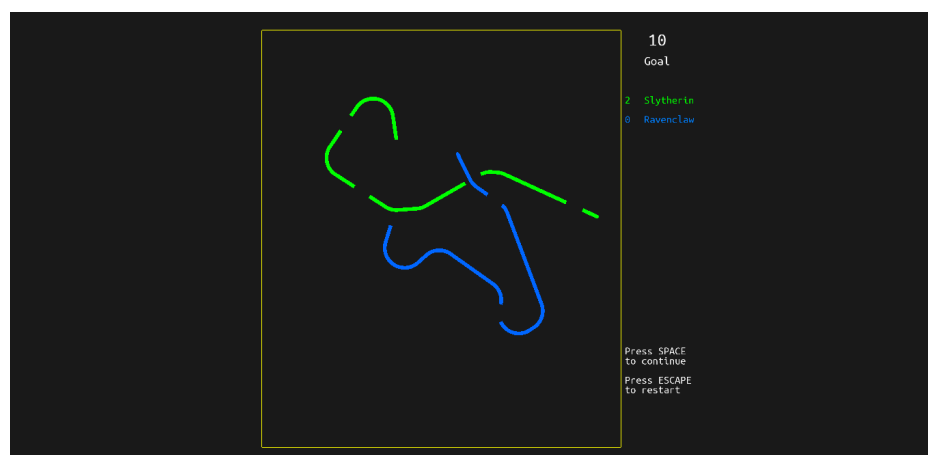


Abb. 13: Ein Spiel pausiert