

TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK

INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK

PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG

PROF. DR. STEFAN GUMHOLD

**Profilmodul Forschungsprojekt Anwendung
INF-PM-FPA**

**Prozedurale, parametergesteuerte Streckengenerierung
für ein Rennspiel in der Unreal Engine 4**

**Julien Fischer
(Mat.-Nr.: 4047337)**

Betreuer: Prof. Dr. rer. nat. Stefan Gumhold

Dresden, 31. Mai 2019

Aufgabenstellung

Ziel des Projektes ist die Erweiterung einer bereits bestehenden Anwendung um ein prozedural generiertes Terrain und eines darin eingebetteten Straßenverlaufes. Die Generierung des Terrains und des Straßenverlaufes sollen Parametern unterliegen, die während der Laufzeit der Anwendung geändert werden können, um so den Schwierigkeitsgrad der Anwendung dynamisch variieren zu können.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät einge-reichte Arbeit zum Thema:

Prozedurale, parametergesteuerte Streckengenerierung für ein Rennspiel in der Unreal Engine 4

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel be-nutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 31. Mai 2019

Julien Fischer

Inhaltsverzeichnis

1 Einleitung	3
2 Die zugrunde liegende Anwendung	3
3 Architektureller Überblick	3
4 Details der Implementation	6
4.1 Terrain Tile	6
4.2 Terrain Generator Worker	6
4.3 Terrain Manager	8
4.4 Terrain Tracker Component	11
5 Ausblick	11

1 Einleitung

Im Rahmen des 2. GameJam der Fakultät Informatik an der TU Dresden wurde ein einfaches Rennspiel in der Unreal Engine 4 entwickelt. Für dieses Rennspiel soll nun ein Level entwickelt werden, das nicht im Vorfeld erstellt werden muss, sondern das zur Laufzeit mithilfe von veränderbaren Parametern, mathematischen Funktionen und Zufallswerten erzeugt wird. Dies hat den Vorteil, dass eine potentiell unendlich große Spielwelt implementiert werden kann, ohne dass im Vorfeld aufwendige Modelle von Hand erstellt werden müssen. Da die Parameter die zur Erstellung der Spielwelt herangezogen werden zur Laufzeit veränderbar sind, lassen sich z.B. die Hügeligkeit des Geländes oder die Kurvigkeitsgrad der Strecke anpassen. Dies kann genutzt werden, um den Schwierigkeitsgrad des Spieles dynamisch anpassen zu können.

Wie auch die zugrunde liegende Anwendung wurde das Profilmmodulprojekt in der Unreal Engine 4 (Version 4.20.3) mithilfe von C++ implementiert. Das Projekt kann online unter [1] abgerufen werden.

An dieser Stelle soll noch auf die beiden Hauptquellen eingegangen werden, auf die sich in diesem Projekt gestützt wurde. Zum einen der große Beleg von Michael Franke aus dem Jahr 2011, in dem die Umsetzung eines Fahrbahnsimulators mit dynamischer Streckenerzeugung beschrieben wird [2]. Zum anderen das Projekt *cashgenUE* vom GitHub Benutzer 'midgen', das ebenfalls in der Unreal Engine 4 implementiert wurde und prozedurales Terrain generiert, hier jedoch mit der Hilfe von Heightmaps [3].

Das folgende Kapitel wird zunächst grob auf die zugrunde liegende Anwendung eingehen, während Kapitel 3 einen Überblick über die verwendete Architektur gibt. In Kapitel 4 werden die einzelnen Komponenten dieser Architektur dann genauer beleuchtet. Zum Schluss gibt Kapitel 5 einen Ausblick auf weitere Features die für die Anwendung möglich sind.

2 Die zugrunde liegende Anwendung

Die zugrunde liegende Anwendung *Hoverblocks* wurde beim 2. GameJam der Fakultät Informatik der TU Dresden innerhalb von 48 Stunden von Jonas Schenke, Torsten Mehnert und Julien Fischer entwickelt. Es handelt sich dabei um ein einfaches Rennspiel, in dem der Spieler einen über dem Gelände schwebenden Block, ein sog. *Hovercraft*, steuert und möglichst schnell eine Rennrunde auf einer vorgegebenen Strecke absolvieren muss. Abbildung 1 zeigt einen solchen spielbaren Block und Abbildung 2 zeigt einen Ausschnitt aus einer vorgegebenen Strecke. Die Strecke und das Terrain wurden dabei im Vorfeld von Hand erstellt, zudem wurden manuell Checkpunkte auf dem Straßenverlauf platziert, zu denen sich der Spieler im Falle eines Unfalls zurücksetzen kann, zu sehen in Abbildung 3. Diese manuelle Erstellung hat sich als eine der zeitintensivsten Aufgaben des Projektes herausgestellt, so dass relativ schnell die Idee einer automatisierten Generierung aufkam. Aufgrund der Zeitbeschränkung konnte diese Idee allerdings nicht im ursprünglichen Projekt realisiert werden.

3 Architektureller Überblick

Da die Spielwelt prozedural generiert werden soll wurde der Ansatz gewählt, die Spielwelt in gleich große, quadratische Abschnitte, sog. *Sektoren*, zu unterteilen. Für jeden dieser Sektoren wird dann ein Objekt, ein sog. *Terrain Tile*, erstellt, das sich um die visuelle Repräsentation des Terrains und der Strecke innerhalb des zugewiesenen Sektors kümmert. In Abbildung 4 ist ein einzelnes Tile mit generiertem Terrain und einem Straßenzug zu sehen.

Da die Generierung des Inhaltes eines solchen Tiles eine gewisse Zeit in Anspruch nimmt, muss sie innerhalb eines eigenen Threads, dem sog. *Terrain Generator Worker*, stattfinden um den normalen Spielfluss nicht zu stören. Um diese Aufgabe kümmert sich der sog. *Terrain Manager*. Er ist außerdem dafür

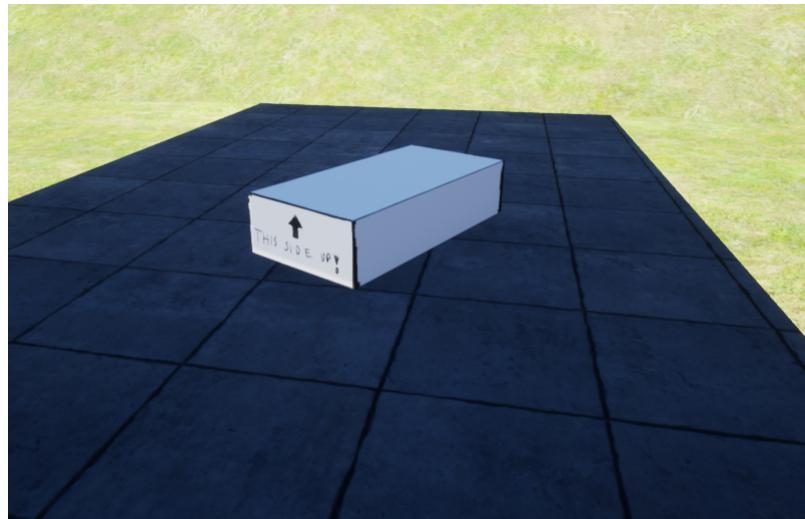


Abbildung 1: Ein vom Spieler steuerbarer Block



Abbildung 2: Ausschnitt aus der Rennstrecke

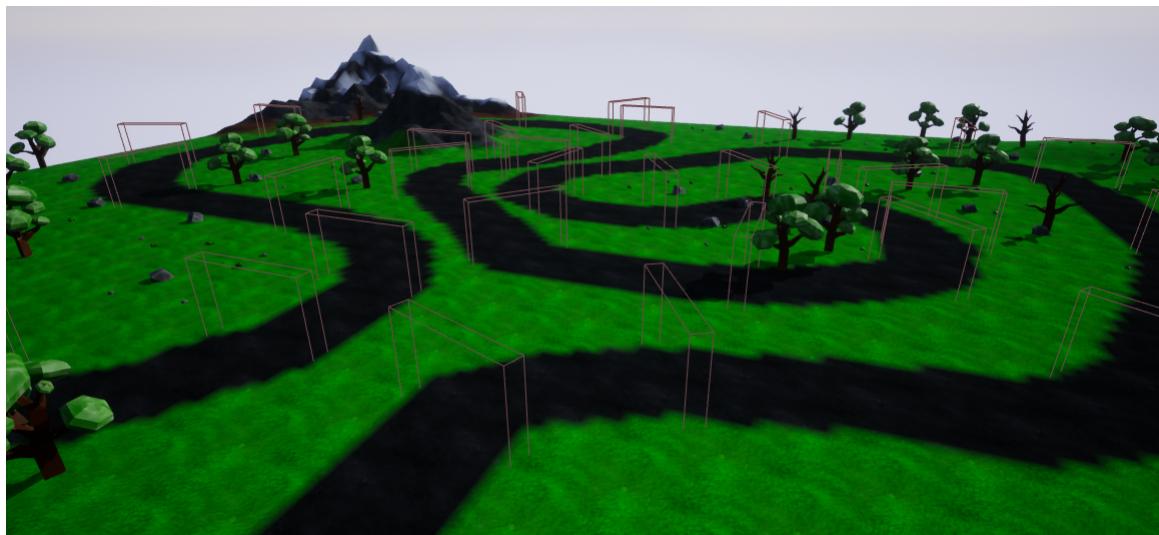


Abbildung 3: Manuell platzierte Checkpunkte auf der Strecke

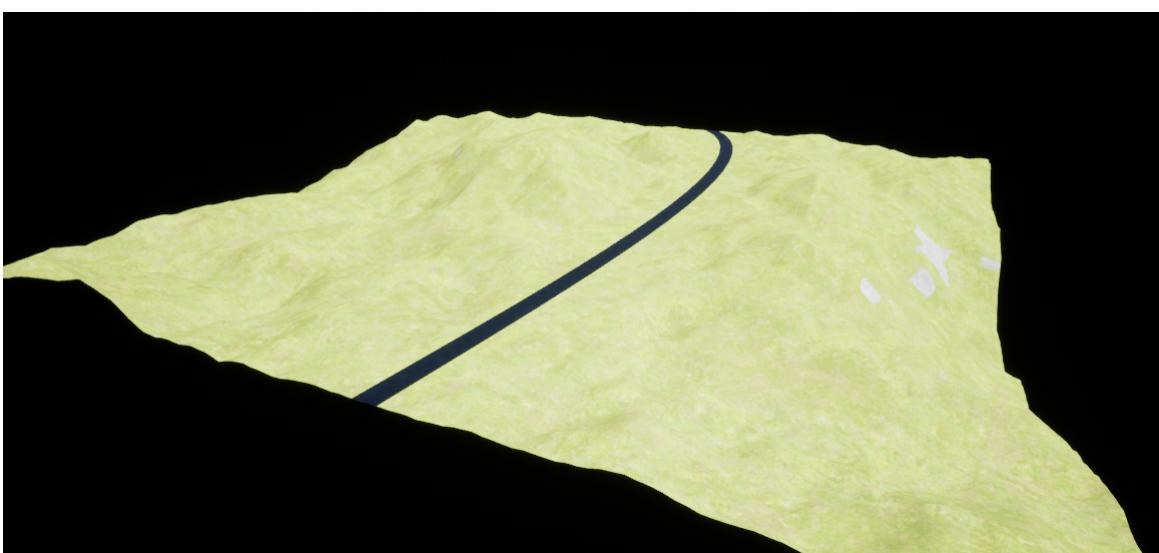


Abbildung 4: Ein einzelnes Tile mit generiertem Terrain und Straßenzug

verantwortlich um jeden relevanten Akteur (also um jedes Hovercraft das am Rennen teilnimmt) eine ausreichende Anzahl an Tiles zu erstellen, so dass dem Spieler der Eindruck einer kontinuierlichen Welt vermittelt wird.

4 Details der Implementation

Die folgenden Sktionen gehen jeweils genauer auf die Implementation eines der Bestandteile der Anwendung ein.

4.1 Terrain Tile

Wie bereits erwähnt wurde ist das Terrain Tile für die visuelle Repräsentation eines Sektors zuständig. Ein Tile wird vom Terrain Manager erzeugt, empfängt dann die vom *Terrain Generator Worker* generierten Geometriedaten (VertexBuffer und Trianglebuffer) und stellt diese mithilfe des *RuntimeMeshComponent* von Koderz [4] dar. Dem RuntimeMeshComponent, einem externen Plugin für die Unreal Engine, wurde als Alternative zum standardmäßig in der Engine enthaltenen ProceduralMeshComponent dabei der Vorzug gegeben, da es ein effektiveres Rendern der Geometrie erlaubt.

Die vom Terrain Generator Worker erzeugte Terrain Geometrie besteht dabei aus einem äquidistanten Punktegitter, dessen Höhenprofil durch die Z Koordinaten der jeweiligen Punkte bestimmt wird. Um einen nahtlosen Übergang zwischen zwei benachbarten Terrain Tiles zu ermöglichen, speichert jedes Tile Informationen über die Höhe der Punkte auf seinen Kanten. Diese Informationen werden dann im Terrain Generator Worker genutzt, wenn benachbarte Terrain Tiles erstellt werden sollen.

Neben der Terrain Geometrie werden dem Terrain Tile ebenfalls die Materialien übergeben, die dann auf die entsprechenden Meshes angewendet werden.

Des weiteren beinhaltet jedes Terrain Tile auch Referenzen auf Objekte, die sich in diesem Tile befinden (z.B. Checkpunkte zum Zurücksetzen des Spielers oder Spieler, die sich gerade in diesem Tile befinden). Mithilfe dieser Referenzen kann dann von außen auf die jeweiligen Objekte zugegriffen werden. So kann unter anderem vom Terrain Manager geprüft werden, ob das Tile noch benötigt wird oder es aus dem Speicher entfernt werden kann.

4.2 Terrain Generator Worker

Der Thread in welchem die Terrain Geometrie erstellt wird nennt sich *Terrain Generator Worker*. Er wird vom Terrain Manager erstellt und mit Arbeitsaufträgen, sog. *Jobs* versorgt. Diese Jobs bestehen zum einen aus dem Terrain Tile, für welches die Geometriedaten erzeugt werden sollen, und einem Array, in das später die fertigen Daten geschrieben werden. Die Jobs werden dem Thread über eine „single-producer single-consumer“ Queue zugeführt. Sobald der Job erledigt wurde wird er dem Terrain Manager über eine „multiple-producer single-consumer“ Queue wieder übergeben.

Der eigentliche Akt der Erzeugung der Terrain Geometrie findet im sog. *Triangle Edge Algorithmus* statt. Hierbei handelt es sich um einen iterativen Algorithmus, der in jeder Iteration die quadratische Grundfläche des Sektors in immer kleinere Quadrate unterteilt, bis schließlich in der finalen Iteration aus jedem so entstandenen Quadrat zwei Dreiecke gebildet werden. Das so entstehende Punktegitter ist äquidistant und hat ein gleichmäßiges Höhenprofil. Um diese Ebenmäßigkeit zu durchbrechen, wird während den Iterationen für jeden erzeugten Punkt die Höhenkoordinate mit einem Wert δ interpoliert. Dieses δ besteht aus einem Zufallswert sowie der aktuellen Iterationstiefe und der fraktalen Dimension. Genaueres zur Berechnung dieses Wertes können die Ausführungen von Miller [5] und Belhadj [6] entnommen werden. Da sich die Implementationen von Miller und Belhadj leicht voneinander unterscheiden sei an

dieser Stelle erwähnt, dass in dieser Arbeit der Implementation von Belhadj der Vorzug gegeben wurde, da sich dort weitere Parameter wie z.B. eine Translation des Zufallswertes einstellen lassen. Abbildung 5 zeigt das Punktegitter eines auf diese Weise generierten Tiles.

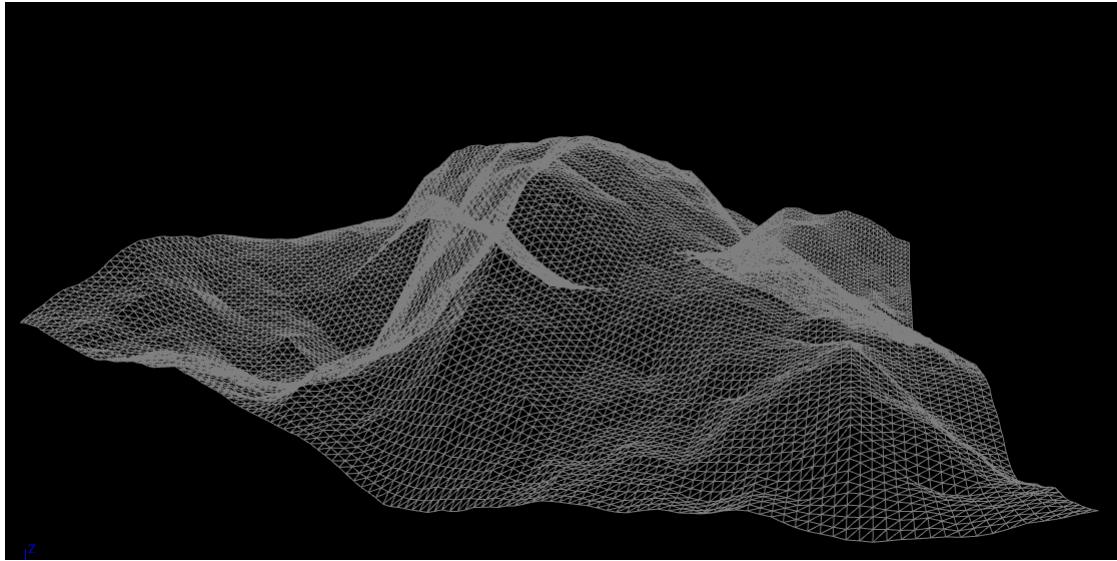


Abbildung 5: Punktenetz eines Tiles

Damit z.B. ein nahtloser Übergang zwischen zwei benachbarten Terrain Tiles möglich ist oder ein flacher Untergrund für die Rennstrecke geschaffen werden kann ist es notwendig, die Höhenkoordinaten einiger Punkte bereits im Vorfeld festzulegen und diese auch im Triangle Edge Algorithmus mehr nicht zu verändern. Solche Einschränkungen werden als *Constraints* bezeichnet und vom Terrain Generator Worker zusammengetragen. Wie Belhadj in [6] festgestellt hat ist es jedoch nicht einfach möglich mit diesen Constraints den Triangle Edge Algorithmus zu starten, da nur genau dann eine artefaktfreie Interpolation möglich ist, wenn die Höhe von Elternpunkten vor oder zeitgleich mit der Höhe der Kindpunkte bekannt sind. Das bedeutet also, dass für einen gegebenen Constraint auch alle Kindpunkte festgesetzt werden müssen. Dies passiert bei Belhadj im sog. *Midpoint Displacement Bottom Up* Prozess. Hierbei wird zuerst ein Simulationsdurchlauf des Triangle Edge Algorithmus ausgeführt, in welchem problematische Constraints erkannt werden (problematische Constraints sind dabei all jene Constraints, deren Elternpunkte keine Constraints sind). Daraufhin werden für jeden solchen Constraint die Elternpunkt Höhenkoordinaten interpoliert und ebenfalls als Constraints gespeichert. Wurden so alle problematischen Constraints behandelt, kann der Triangle Edge Algorithmus mit Constraints gestartet werden.

In der entwickelten Anwendung gibt es zwei Unterschiedliche Arten von Constraints: *Track Constraints* und *Border Constraints*. Track Constraints sind hier all jene Constraints die genutzt werden, um dem Terrain an bestimmten Stellen einen ebenen Untergrund zu geben, so dass eine Straße darüber gelegt werden kann. Die Track Constraints berechnet der Terrain Generator Worker aus den Straßengeometriedaten, die er vom Terrain Manager erhält. Die Border Constraints werden genutzt, um den Übergang zwischen zwei Terrain Tiles zu gestalten. Hierfür fragt der Terrain Generator Worker von allen existierenden Tiles die an das zu erstellende Tile angrenzen die gespeicherten Höheninformationen der Punkte ab, die auf den Kanten des Tiles liegen. So wird erreicht dass in zwei angrenzenden Tiles Punkte, die in beiden Tiles existieren (also Punkte auf der verbindenden Kante zwischen den beiden Tiles) dieselbe Höhenkoordinate besitzen.

4.3 Terrain Manager

Der *Terrain Manager* bildet das Herzstück der Anwendung. Beim Start des Levels erzeugt er einen Thread, den *Terrain Generator Worker*, der für die Generierung der Terrain- und Streckengeometrie verantwortlich ist. Außerdem sorgt der Terrain Manager dafür, dass um jeden bei ihm registrierten Akteur (jedes Hovercraft) die vorgegebene Anzahl an Terrain Tiles erstellt wird. Für jedes dieser Tiles wird dann ein Job erstellt, um vom Terrain Generator Worker die Geometrie für das Tile generieren zu lassen. Nachdem die Geometriegenerierung abgeschlossen ist überträgt der Terrain Manager die Daten dann an das entsprechende Tile, welches sich um die weiteren nötigen Schritte (z.B. Anwendung des Materials) kümmert.

Neben der Anzahl an Tiles, die um ein Hovercraft herum erstellt werden sollen, gibt es noch unzählige andere Parameter, die den Programmablauf beeinflussen. Insbesondere zählen hierzu Parameter die die Generierung des Terrains und der Strecke beeinflussen. All diese Parameter werden zentral im Terrain Manager gespeichert und können dort angepasst werden, sowohl zur Laufzeit als auch zur Entwicklungszeit.

Des Weiteren ist der Terrain Manager dafür verantwortlich frei gewordene Tiles, d.h. Tiles die für keinen registrierten Akteur mehr relevant sind, nach einer gewissen Zeit aus dem Speicher zu entfernen. So wird sichergestellt, dass der Arbeitsspeicher auch bei langer Spielzeit nicht ausgeht. Um dies gewährleisten zu können verwaltet der Terrain Manager zwei Listen. Die eine Liste enthält alle Tiles, die gerade benutzt werden(d.h. die relevant für mindestens einen Akteur sind), während die andere Liste alle Tiles enthält, die zwar momentan erstellt sind, aber für keinen Akteur relevant sind. Wenn ein neuer Sektor dargestellt werden soll wird also zuerst geschaut, ob ein freies Tile verfügbar ist. Wenn ja wird dieses benutzt, wenn nicht wird ein neues Tile erstellt. Wenn ein freies Tile nun eine gewisse Zeit lang nicht für einen neuen Sektor gebraucht wurde, so entfernt der Terrain Manager dieses Tile aus dem Speicher.

Eine weitere Aufgabe für den Terrain Manager ist die Generierung der globalen Streckenführung. Dafür ist es notwendig, für neu darzustellende Sektoren zu überprüfen, ob der Sektor einen Abschnitt der Rennstrecke beinhaltet soll. Dies wird dadurch realisiert, dass der Terrain Manager einen Verweis auf den nächsten Sektor beinhaltet, in dem ein Streckenabschnitt liegen soll, momentan aber noch kein Streckenabschnitt vorhanden ist. Soll nun eben dieser Sektor dargestellt werden, muss der Terrain Manager folgende Aufgaben erledigen:

- Berechnung des nächsten Sektors, der einen Streckenabschnitt beinhalten soll
- Berechnung der Bézierkontrollpunkte
- Berechnung der Punkte auf der Bézierkurve

Berechnung des nächsten Sektors, der einen Streckenabschnitt beinhalten soll: Damit die Streckenführung nicht irgendwann aufhört, muss der Terrain Manager den weiteren globalen Verlauf der Rennstrecke berechnen. Hierfür werden alle an den momentanen letzten Streckensektor angrenzenden Sektoren überprüft, ob sie sich als nächster Streckensektor eignen. Die Anforderungen an den neuen Streckensektor sind dabei, dass er noch nicht erstellt worden sein darf (um zu verhindern dass Terrain Tiles überschrieben werden während sich der Spieler möglicherweise in ihnen befindet) und dass er nicht im sog. *No-go-Quad* enthalten ist. Das No-go-Quad ist ein Viereck, das mehrere Sektoren einschließt, darunter auch alle bisher berechneten Sektoren die einen Streckenabschnitt beinhalten. Dadurch, dass nun verhindert wird, dass der nächste Sektor mit einem Streckenabschnitt in diesem Viereck liegt, wird verhindert, dass sich Schlaufen ähnlich der in Abbildung 6 bilden. Dies ist notwendig, damit die globale Streckenführung unbegrenzt fortgesetzt werden kann.

Aus allen Sektoren die diesen Anforderungen genügen wird dann per Zufall der nächste Sektor der einen Streckenabschnitt beinhalten soll ausgewählt. Das No-go-Quad wird um den momentanen Sektor erweitert.

Berechnung der Bézierkontrollpunkte: Die Generierung des Streckensegmentes für ein Tile folgt ei-

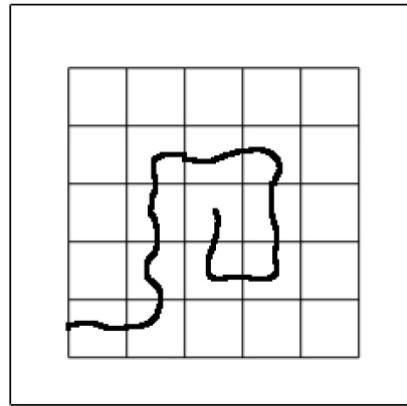


Abbildung 6: Möglicher Verlauf einer Streckenführung der in einer Schlaufe endet. Entnommen aus [2]

ner Bézierkurve, für welche der Terrain Manager die entsprechenden Kontrollpunkte berechnen muss. Während der Anfangspunkt der Bézierkurve der Endpunkt der Bézierkurve des vorangegangenen Sektors ist, ergibt sich der Endpunkt der Bézierkurve entsprechend der Lage des nächsten Sektors, der ein Streckensegment beinhalten soll.

Der erste Kontrollpunkt wird ebenfalls aus der Bézierkurve des vorangegangenen Sektors berechnet. Damit die Orientierung der Strecke am Eingangspunkt des Sektors dieselbe ist wie am Austrittspunkt der Strecke im vorherigen Sektor, wird der Vektor vom zweiten Kontrollpunkt im vorherigen Sektor zum Austrittspunkt im vorherigen Sektor an den Eintrittspunkt im aktuellen Sektor angelegt und der daraus resultierende Punkt ist der erste Kontrollpunkt.

Der zweite Kontrollpunkt ergibt sich aus zwei Normalverteilungen: Die erste Normalverteilung bestimmt die Position eines Zwischenpunktes auf der Linie zwischen Sektor Mittelpunkt und Bézierkurve Austrittspunkt. Die zweite Normalverteilung bestimmt den Winkel einer Rotation dieses Zwischenpunktes um den Austrittspunkt der Bézierkurve. Der rotierte Zwischenpunkt ergibt dann den zweiten Kontrollpunkt.

Diese Regeln zur Erstellung der Bézierkontrollpunkte wurden aus dem großen Beleg von Michael Franke [2] übernommen, mit einer kleinen Anpassung bei der Berechnung der Lage des Zwischenpunktes für den zweiten Kontrollpunkt. Abbildung 7 zeigt eine Grafik aus dem Beleg, die einen Beispielstreckenverlauf mit Kontrollpunkten über mehrere Tiles hinweg schematisch darstellt. Abbildung 8 zeigt einen solchen fertig generierten Verlauf in der Anwendung.

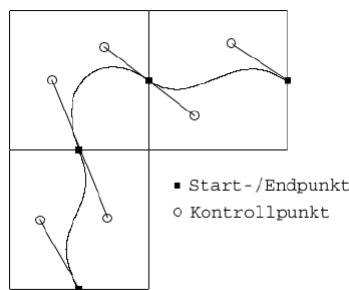


Abbildung 7: Schematische Darstellung eines Streckenverlaufes mit Kontrollpunkten über mehrere Tiles hinweg. Entnommen aus [2]

Die berechneten Kontrollpunkte werden anschließend, zusammen mit anderen für den Sektor relevanten Werten, in einer Hashmap gespeichert, um so, sollte der Sektor in der Zukunft erneut dargestellt wer-

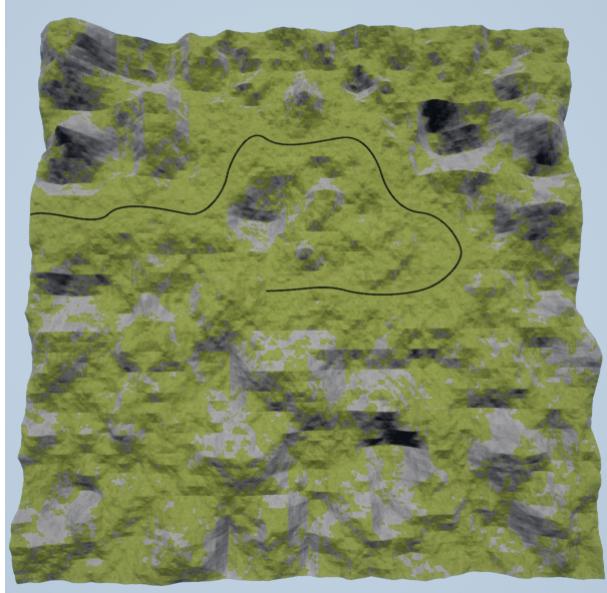


Abbildung 8: Streckenverlauf über mehrere Tiles hinweg mit um den Straßenzug herum generiertem Terrain.

```
static float FVector::EvaluateBezier
(
    const FVector * ControlPoints,
    int32 NumPoints,
    TArray < FVector > & OutPoints
)
```

Abbildung 9: Funktionsdeklaration der EvaluateBezier methode.

den müssen, verwendet werden zu können. Dadurch kann die lokale sowie die globale Streckenführung persistent gehalten werden, auch wenn Tiles in der Zwischenzeit aus dem Speicher entfernt werden.

Berechnung der Punkte auf der Bézierkurve: Damit später eine Streckenführung durch das Tile generiert werden kann müssen Punkte, die auf der durch die Kontrollpunkte definierten Bézierkurve liegen, berechnet werden. Hierfür gibt es in der Unreal Engine bereits eine vorgefertigte Funktion EvaluateBezier(), siehe Abbildung 9. Die so erhaltenen Punkte werden ebenfalls in der Hashmap gespeichert und später bei der Erstellung der Straße genutzt.

Eine weitere wichtige Aufgabe des Terrain Managers ist die Generierung des Straßenmeshes. Zwar wird diese vom Thread des Terrain Generator Worker's ausgeführt, da die Berechnung aber von Variablen abhängt die nur im Terrain Manager zur Verfügung stehen (beispielsweise die Kontrollpunkte für die Bézierkurve die in einer Hashmap abgespeichert sind), stellt der Terrain Manager die nötigen Funktionen bereit, die dann vom Terrain Generator Worker aufgerufen werden. Da die im Vorfeld berechneten Punkte auf der Bézierkurve nur eine Linie bilden, erstellt der Terrain Manager für jeden bereits berechneten Punkt X zwei Punkte Y_0 und Y_1 die in definierten Abständen links und rechts vom Punkt X liegen und so die Breite der Strecke bestimmen. Wurden so für einen Punkt X^t die Seitenpunkte Y_0^t und Y_1^t berechnet und für den Punkt X^{t+1} die Punkte Y_0^{t+1} und Y_1^{t+1} , so erstellt der Terrain Manager ein Streckensegment, indem er zwei Vierecke $(Y_0^t X^t X^{t+1} Y_0^{t+1})$ und $(X^t Y_1^t Y_1^{t+1} X^{t+1})$ erstellt und diese trianguliert. Dies wird nun für alle Punkte X auf der Bézierkurve wiederholt und die so entstehenden Dreiecke und Dreieckspunkte werden in die entsprechenden Triangle- und Vertexbuffer übertragen, welche später zum rendern an das Terrain Tile übergeben werden.

Eine letzte Aufgabe die der Terrain Manager übernimmt ist das Erstellen von Checkpunkten, zu denen

sich der Spieler zurücksetzen kann. Pro Terrain Tile mit Straßenzug wird üblicherweise ein Checkpunkt vom Terrain Manager erstellt und die Referenz zu diesem Checkpunkt wird dem entsprechenden Terrain Tile übergeben, so dass dieses sich um die weitere Verwaltung des Checkpunktes kümmern kann.

4.4 Terrain Tracker Component

In der Unreal Engine werden Objekte, die in der Szene platziert werden können, im allgemeinen als Akteure („Actors“) bezeichnet. An diese Akteure können Komponenten („ActorComponents“) „angeheftet“ werden, die dem Akteur zusätzliche Funktionalitäten bereitstellen. Das Terrain Tracker Component ist eine solche Komponente, die jedem Akteur dem sie hinzugefügt wird die Möglichkeit bietet, sich beim Terrain Manager zu registrieren, so dass fortan Terrain Tiles um den Akteur herum generiert werden. Außerdem ist die Komponente dafür verantwortlich dem Terrain Manager mitzuteilen, wenn der zugehörige Akteur einen Sektor verlassen und einen neuen betreten hat, so dass der Terrain Manager neu benötigte Tiles erstellen und nicht mehr benötigte Tiles löschen kann.

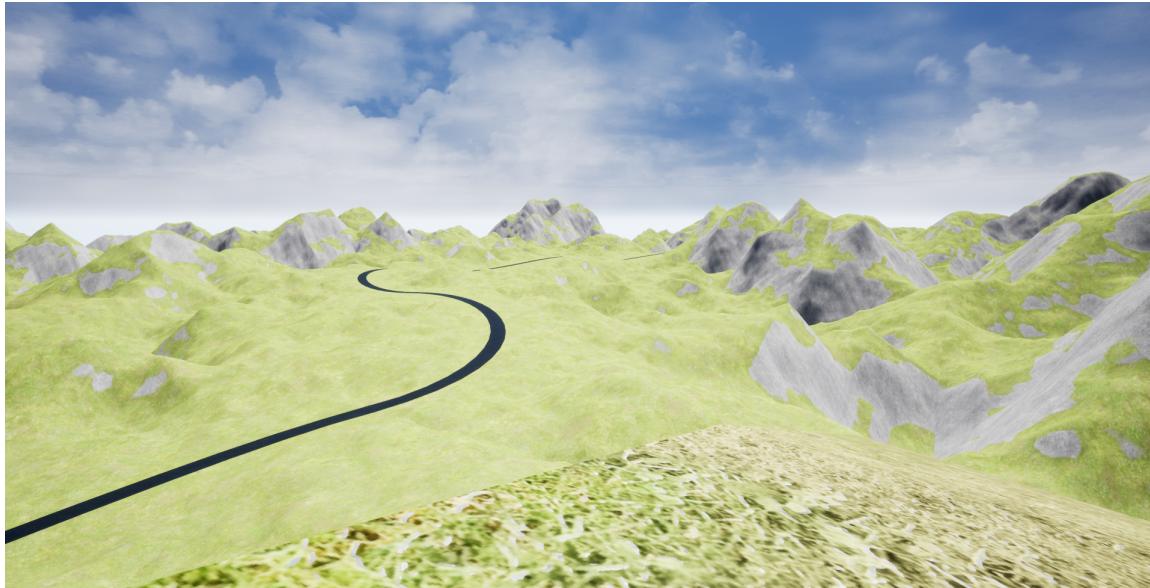
5 Ausblick

Dieses Kapitel soll einen kurzen Überblick über Funktionalitäten und Features liefern, die zwar für die Anwendung angedacht sind, aber im aktuellen Stand des Projektes noch nicht implementiert wurden.

Als erstes sei hier ein Ziel für den Spieler zu nennen, denn bisher kann sich der Spieler zwar frei in der Spielwelt bewegen, jedoch besitzt er neben dem freien Erkunden keinen Anreiz dafür. Dies soll geändert werden, indem immer wieder Punkte auf der Streckenführung erstellt werden, die in einer gewissen Zeit erreicht werden müssen. Erreicht der Spieler einen solchen Punkt, bekommt er Zeit auf seinem Zeitkonto gutgeschrieben und muss den nächsten Punkt erreichen. Die Distanz zwischen den zu erreichenden Punkten wird dabei immer größer, während die zur Verfügung stehende Zeit immer geringer wird. Schafft der Spieler es irgendwann nicht mehr rechtzeitig den Punkt zu erreichen, ist das Spiel vorbei und ein Highscore wird aus seiner insgesamt zurückgelegten Strecke berechnet. Anhand dieses Highscores kann der Spieler seine Leistung mit der von anderen Spielern vergleichen.

Um den Spieler dazu zu bringen während der Fahrt auf der Rennstrecke zu bleiben und nicht Kurven über das Terrain abzukürzen, soll zudem die Möglichkeit eingeführt werden, die maximale Beschleunigung des Hovercrafts abhängig vom Untergrund zu machen.

Zuletzt soll noch die Möglichkeit genannt werden, Power-ups auf dem Streckenverlauf zu platzieren. Diese könnten dem Spieler beim Aufsammeln für eine gewisse Zeit einen Bonus geben, z.B. eine erhöhte Beschleunigung.



References

- [1] Julien Fischer Jonas Schenke Torsten Mehnert. *Hoverblocks by Brigade Vollrausch*. 2018. URL: https://github.com/jufi2112/GameJam2018_Brigade_Vollrausch.
- [2] Michael Franke. *Dynamische Streckengenerierung und deren Einbettung in ein Terrain*. Mar. 2011.
- [3] midgen. *cashgenUE*. URL: <https://github.com/midgen/cashgenUE>.
- [4] Koderz. *RuntimeMeshComponent*. URL: <https://github.com/Koderz/RuntimeMeshComponent>.
- [5] Gavin S P Miller. “The Definition and Rendering of Terrain Maps”. In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’86. New York, NY, USA: ACM, 1986, pp. 39–48. ISBN: 0-89791-196-2. DOI: 10.1145/15922.15890. URL: <http://doi.acm.org/10.1145/15922.15890>.
- [6] Farès Belhadj. “Terrain Modeling: A Constrained Fractal Model”. In: *Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*. AFRIGRAPH ’07. Grahamstown, South Africa: ACM, 2007, pp. 197–204. ISBN: 978-1-59593-906-7. DOI: 10.1145/1294685.1294717. URL: <http://doi.acm.org/10.1145/1294685.1294717>.