Entwicklung eines 2D Multiplayer Space Shooter in C++

Development of a 2D Multiplayer Space Shooter in C++

Bachelor-Teamprojekt

Teilnehmer : Wichter, David, 955941

Simon, Jan, 955029

Assert, Yannick, 954998

Betreuer : Prof. Dr. Christof Rezk-Salama

Semester : Wintersemester 2014 / 2015

Gliederung

1. Vorstellung des Projekts

1.1. Allgemeine Beschreibung

“Alpha Strike” ist ein 2D Multiplayer Space-Shooter, in dem bis zu vier Spieler gegeneinander antreten können.

1.2. Spielablauf

Zunächst muss der Server gestartet und entsprechend des folgenden Spiels eingestellt werden. Nachdem der Server gestartet ist, können sich die einzelnen Spieler über ein Interface mit diesem verbinden. Sobald alle Spieler erfolgreich mit dem Server verbunden sind, startet das Spiel. Nun muss jeder Spieler versuchen, die Raumschiffe der anderen Spieler mit Raketen abzuschießen. Es ist dabei in einem gewissen Maß möglich, die Raketen selbst zu steuern. Sobald ein Spieler getroffen wurde, scheidet dieser bis zum Ende der Runde aus. Derjenige, der am Ende als einziger übrig ist, gewinnt die Runde und erhält einen Punkt. Daraufhin startet die nächste Runde. Das Spiel ist beendet, wenn ein Spieler mehr als die Hälfte der zu erreichenden Punkte gesammelt (“Best of”-Methode) und in Folge dessen gewonnen hat.

1.3. Steuerung

Die Steuerung des Raumschiff ist relativ einfach gehalten. Mit den Tasten “A” und “D” wird das Raumschiff um die eigene Achse gedreht und mit der Taste “W” kann das Raumschiff beschleunigt werden. Möchte der Spieler seine Geschwindigkeit verringern, so muss dieser sein Raumschiff in die entgegengesetzte Flugrichtung drehen und wieder beschleunigen.

Raketen können mit der Leertaste abgefeuert werden. Solange die Leertaste gedrückt ist, kann die Rakete mit den Tasten “A” und “D” nach links oder rechts gelenkt werden. Während dieser Zeit ist es jedoch nicht möglich das Raumschiff zu steuern; erst wenn die Leertaste losgelassen wird kann das Raumschiff wieder gesteuert werden.

2. Netzwerkkommunikation

2.1. Allgemeines und Konzept

Um den Multiplayer-Aspekt von Alpha-Strike umzusetzen, baut die Anwendung auf eine traditionelle Server-Client Architektur auf der Basis von UDP auf, bei der jeder Teilnehmer das Spiel über eine eigene Instanz des Clients erlebt und steuert. Die Daten jedes Clients werden in regelmäßigen Abständen an die Server-Applikation geschickt, die auf der Maschine des sogenannten Hosts ausgeführt wird. Dabei ist es möglich und durchaus gebräuchlich, dass ein Spieler sowohl Client als auch Host betreibt und sich mit über seine Client-Instanz mit seiner eigenen Maschine verbindet. Der Server hat die Aufgabe die gesammelten Information der Spielteilnehmer aufzubereiten und die Spieldaten aller Clients untereinander zu verteilen. Dabei werden lediglich solche Daten verschickt, die sich zwischen Clients unterscheiden.

Wir unterscheiden zwischen Reliable und Non-Reliable Data. Letztere Daten werden einfach so über das Netzwerk geschickt und finden Verwendung während das eigentliche Spiel selber läuft, wenn der Client in jedem Frame seine Position zum Beispiel zum Server schickt. Reliable Data hingegen ist jede Information, die nur einmal verschickt wird (als Event) und auf jeden Fall ankommen muss. Hierfür haben wir ein einfaches Acknowledgement-System verwendet, da UDP selber im Gegensatz zu TCP keine eingebaute Möglichkeit besitzt um sicherzugehen, dass verschickte Pakete auch wirklich ankommen. Dabei verschickt Spieler A sein Reliable-Data-Packet immer wieder so lange, bis er von Spieler B ein entsprechendes Acknowledgement-Packet erhält. Da dieses Acknowledgement-Packet natürlich genauso wie das ursprüngliche verlorengehen kann, muss Spieler B auf jede eingehende Kopie des Packets mit einem Acknowledgement reagieren, bis Spieler A die Bestätigung erhalten hat und mit dem Versenden seines Packets aufhören kann.

2.2. Client

Im Client findet das eigentliche Spielgeschehen statt. Das bedeutet, dass jede Art von Spiellogik und Spielfluss für jeden Teilnehmer der Partie individuell in seiner eigenen Clientinstanz verarbeitet wird. Einmal pro Spielschleife wird der Netzwerkteil des Programms auf neue Daten vom Server abgefragt, in denen die Spieldaten der anderen Teilnehmer enthalten sind. Das Verhalten der jeweils anderen Spieler wird durch den sogenannten Netplayer dargestellt und alle Teile der Spiellogik, die Informationen von und über die anderen Clients benötigen, arbeiten mit diesen Repräsentationen. Falls das Serverpacket neue Daten enthält, werden diese an die Netplayer-Instanzen weitergeleitet, welche daraufhin ihre Spieldaten auf den neuesten Stand bringen und somit die anderen Teilnehmer des Spiels repräsentieren. Die Daten, die von jedem Client an den Server geschickt werden, sind folgende:

- Die Position des Spielers
- Der Richtung, in die der Spieler schaut
- Der Winkel der Rotation des Spielers
- Die Position der Rakete des Spielers
- Die Richtung, in die die Rakete des Spielers schaut
- Ob der Spieler gestorben ist oder nicht

Diese Daten werden als 32 Bit große Gleitkommazahlen verschickt (der letzte Wert ist als boolean mit 1,0 = true und 0,0 = false codiert). Zusammenfassend schickt also jeder Client die Daten seiner Spielerinstanz an den Server und erhält im Gegenzug die äquivalenten Daten der jeweils anderen Clients.
Da der Server die Informationen in einem langsameren Takt verteilt als nötig wäre damit jeder Netplayer in jeder Spielschleife aktuelle Informationen hat, wird zwischen den einzelnen Serverpaketen interpoliert. Dazu speichert sich der Client immer das Serverpaket, das vor dem aktuellsten angekommen ist. In den Spielschleifen-Zyklen, in denen keine neuen Informationen vom Server kommen, findet eine lineare Interpolation

zwischen dem vorletzten und dem letzten erhaltenen Paket statt. Auf diese Weise wird eine stockende und flackernde Darstellung der Netplayer verhindert.

2.3. Server

Der Server trägt die Daten aller Clients zusammen und verteilt diese wiederum unter den Teilnehmern des Spiels. Dabei müssen einige Parameter der Partie im Voraus bekannt sein: Neben dem Port, auf dem der Server ansprechbar sein soll, muss die Tickrate bekannt sein mit der neue Daten an die Clients verteilt werden soll. Außerdem muss bekannt sein, wie viele Spieler an der Partie teilnehmen werden und wie viele Runden gespielt wird. Zuletzt muss dem Server noch mitgeteilt werden, welche Karte gespielt wird.

Sind alle diese Parameter eingestellt, startet der Server die Partie und wartet darauf, dass sich Clients mit einem “Willkommenspaket” als neue Spieler anmelden (siehe Abbildung 1 unten). Der Inhalt dieses Pakets besteht aus dem Namen des Spielers und enthält auch die nötigen Verbindungsinformationen des Clients. Der Server antwortet mit einer “gameinfo”-Nachricht, die als Acknowledgement-Paket für das Willkommenspaket fungiert und in der dem Teilnehmer die Anzahl an Mitspielern, die Rundenanzahl der Partie und die vom Server gewünschte Map (als Index) geschickt wird. Anschließend wird dem Client noch in einem Start-Paket seine eigene Client-ID beim Server und die Namen aller anderen Spieler mitgeteilt. Nachdem der Empfänger dieses Paket erneut mit einem “Acknowledged”-Paket bestätigt, ist die Verbindungsaufnahme (auch: “Handshake”) mit diesem Client beendet. Wenn die Verbindungsaufnahme mit allen Spielern beendet ist, wechselt der Server in den Gameplay-Zustand und die erste Runde der Partie kann starten.

In diesem Zustand wartet der Server auf neue Datenpakete der Spieler (siehe auch Kapitel 2.2) und sammelt diese. Wenn der nächste Takt erreicht wird, schickt er die Daten der Clients zurück, wobei er jedem Spieler allerdings nur die Daten der anderen Teilnehmer und niemals die eigenen Informationen zusendet.

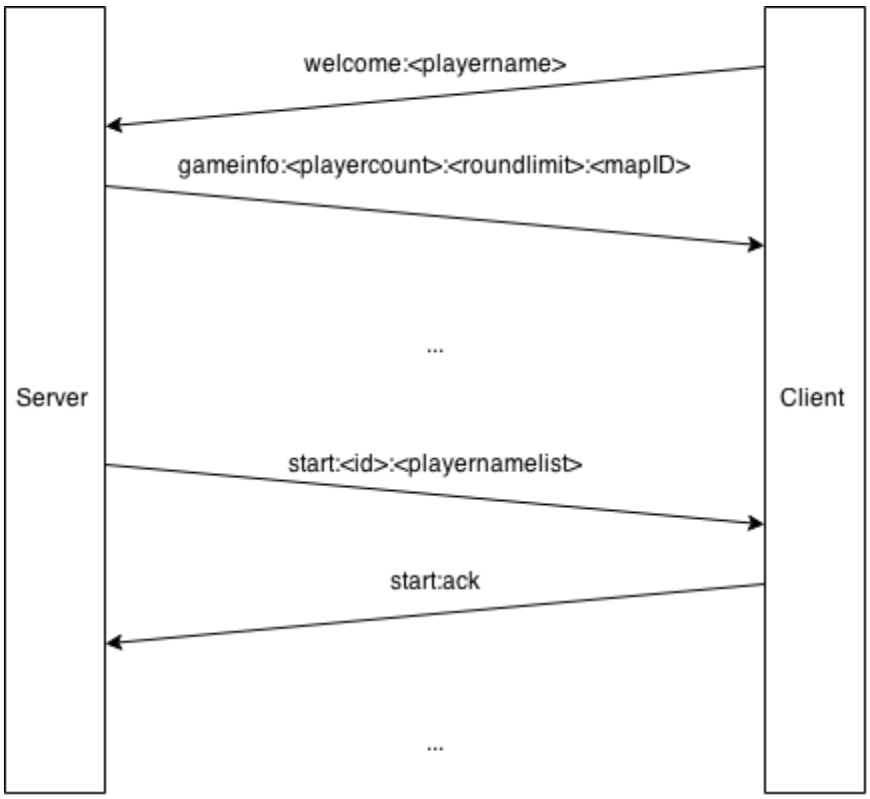

Abb. 1: Der Handshake zwischen Server und Client

3. Codestruktur

Das Projekt ist in zwei Unterprogramme eingeteilt, die unabhängig voneinender programmiert, compiliert und betrieben werden: Den Server und den Client. Die Aufgaben der beiden Programme wurden in den Kapiteln 2.2 und 2.3 erläutert, daher soll im Folgenden nur auf die Struktur des Codes eingegangen werden.

Die meisten Klassen des Clients lassen sich in zwei Unterkategorien sortieren, und zwar in Module und in Entitäten. Die Module sind Teile des Programms die eine bestimmte Aufgabe der Spiellogik und/oder -darstellung erfüllen. Sie existieren die meiste Zeit nur mit einer einzigen Instanz während das Spiel läuft und sind daher zum größten Teil als Singletons implementiert. Beispiele für Module sind der Spriterenderer,

der alle Sprites auf dem Bildschirm zeichnet, oder der InputObserver, der gemäß des Observer-Modells empfangene Eingaben an seine Listener weiterleitet.
Instanzen, die aktiv an der Simulation teilnehmen, sind Entitäten. Sie haben eine explizite Präsenz in der Spielwelt, die mit einer Position in der Spielwelt einhergeht. Daher sind die meisten Entitäten, wie zum Beispiel die Spielfigur oder ihre Raketen, Ableitungen der Klasse Transform, welche die Basis für Bewegliche Objekte darstellt.
Zuletzt gibt es noch Hilfsklassen, wie die Klasse Countdown oder Vector2, die es ermöglichen gewisse Aufgaben einfacher zu bewältigen.
Alle Module und Entitäten werden in einem Zustandsautomaten verwaltet, der das Grundgerüst für die Applikation bietet und die Spielschleife implementiert. Die State Machine beseht aus der Klasse Game, der abstrakten Klasse Gamestate und den Ableitungen von Gamestate. Die Kernschleife findet in Game statt und besteht aus den Aufrufen von update() und render(), welche an eine Instanz einer Ableitung von Gamestate delegiert werden. Soll der Zustand gewechselt werden, so wird die Gamestate-Instanz von Game polymorphisch mit einem anderen abgeleiteten Gamestate ausgetauscht.
Der Server arbeitet ebenfalls mit einem Zustandsautomaten, allerdings ohne entsprechende Klassenhierarchien. Stattdessen wird der Zustand anhand eines Enums unterschieden, der Teil der Hauptklasse Server ist. In dieser Klasse findet auch die Kernschleife statt, die zwischen dem Empfangen von Informationen und dem Verteilen der entsprechenden Daten an die Clients wechselt (siehe Kapitel 2.3). Die Daten, die der Server verwalten muss, werden in Instanzen der Klasse Playerinfo gespeichert, welche als Container für alle spielerspezifischen Informationen dient. Schlussendlich gibt es noch die Klasse Main, die den Server initialisiert, die Kernschleife einleitet, und schließlich beendet.

In den Verzeichnissen *Documentation/Codestructure/Client/html/index.html* und *Documentation/Codestructure/Server/html/index.html* kann eine detaillierte Übersicht der Codestruktur, sowie der Aufbau der einzelnen Klassen eingesehen werden.