SOFTWAREPROJEKT – 3D GAME

Dokumentation

Team:
Lena Spitz (Teamleiterin)
Lars Haider
Mareen Allgaier

Dozent:
Prof. Holger Theisel
Gerd Schmidt (Seminarleitung)

Projektdauer:
04.04.2016 – 30.09.2016

Inhaltsverzeichnis

Einführung

Im Rahmen des Softwareprojekts 3D-Game-Development an der Fakultät für Informatik der Otto-von-Guericke Universität war die Aufgabe, ein 3D Spiel zu entwickeln. Dafür hatten Teams von drei bis vier Studenten sechs Monate Zeit. In diesen gab es regelmäßige Treffen mit dem Seminarleiter und den anderen Entwicklungsteams, um den aktuellen Stand, Erfolge, Probleme und Vorgehensweisen zu diskutieren.

Das Team um Lena Spitz, Lars Haider und Mareen Allgaier entschied sich dazu, ein Spiel namens „Volamus“ zu entwickeln. Bei dem Zwei-Spieler-Spiel „Volamus“ handelt es sich um ein Sportspiel, in dem Tiere eine vereinfachte Version von Volleyball spielen. Es geht darum, dass zwei Charaktere einen Ball über ein Netz spielen und versuchen, Punkte zu bekommen, indem sie den Ball in das Feld des Gegners Schlagen.

Der Name des Spiels ergibt sich aus dem Sport Volleyball. Dieser wird auch Volley genannt und kommt vom Lateinischen volare, was fliegen bedeutet. „Volamus“ ist eine Konjugation von volare, und heißt übersetzt „wir fliegen“.

Prototyp

Bevor es an die Ausarbeitung eines Konzeptes für das endgültige Spiel ging, entwickelte jede Gruppe ein kurzes, kleines 3D-Spiel. Dieses diente dem Kennenlernen der Entwicklungsumgebung. Bereits hier arbeiteten wir, das spätere Team von „Volamus“, zusammen. Da kein Teammitglied über Vorwissen in der Spieleprogrammierung verfügte, hatten wir uns mehrmals getroffen, um gemeinsam zu programmieren. Hierbei wurde die Entwicklungsumgebung zusammen kennen gelernt und erste Erfahrungen gesammelt.

Unser Prototyp war ein Weltraumspiel, bei dem ein Raumschiff Kometen ausweichen oder diese mithilfe von mit Soundeffekten ausgestatteten Projektilen abschießen musste. Wurde das Raumschiff des Spielers von einem Kometen getroffen, war das Spiel verloren. Dies wurde durch Aufblinken von „Game Over“ und ein akustisches Feedback deutlich. Es gab außerdem Hintergrundmusik und eine statische Textur eines Sternenhimmels als Hintergrund.

Es wurde einige Zeit zum Einarbeiten benötigt und um herauszufinden, wie das Grundgerüst eines 3D-Spiels überhaupt aussieht und was für das Spielen benötigt wird. Eigene Modelle wurden für den

Prototypen noch nicht erstellt, man bediente sich an frei-verwendbaren Beispielmodellen aus Online-Tutorials. Außerdem wurde auf Texturen komplett verzichtet, da wir uns hauptsächlich auf die Spielmechanik konzentrierten. Zeitintensiv waren hierbei vor allem die Kollisionen, aber auch das Konzept der Kameras und das Einbinden von Modellen in die Umgebung.

Die Entwicklung des Prototyps hat es ermöglicht, bereits vor dem Projekt alle Teammitglieder kennen zu lernen. So wurde auch festgestellt, dass eine gute Zusammenarbeit möglich ist.

Außerdem bot der Prototyp die Chance, das Programm frei auszuprobieren. Dabei wurde deutlich, dass ein Spiel auch einfach sein kann, es aber trotzdem Spaß bereitet, solange es flüssig spielbar ist. Wichtig für den Spaßfaktor und das Verständnis sind sowohl akustische als auch optische Feedbacks.

Das Spiel

Die Geschichte von Volamus

Um den Spieler den Einstieg zu erleichtern und ihn emotional in das Spiel zu involvieren, besitzt „Volamus“ eine Geschichte:

„Schon seit vielen vielen Jahren gibt es eine Legende von einer goldenen Wolke. Niemand weiß genau, wann und wie die Wolke entstanden ist und welche Geheimnisse sie birgt.

Aber eines wird in allen Versionen der Legende klar – die Wolke ist sehr mächtig und verspricht unfassbare Kräfte.

Unter den Acagamics Männchen gib es eine Gruppe alter weiser Gelehrter, die an den Wahrheitsgehalt dieser Legende glauben. Deshalb machten sie es sich zur Aufgabe, die Wolke zu finden und zu studieren.

Für dieses gewagte Unternehmen suchen sie nun eine Gruppe aus fähigen Gefährten.

Um für alle Gefahren gewappnet zu sein und erfolgreich wiederzukehren, sollen sie über Geschicklichkeit, Stärke, Anpassungsfähigkeit und Mut verfügen.

Ein Turnier soll sicherstellen, dass wirklich nur die Besten der Besten in dieses Abenteuer geschickt werden.

Hierbei müssen sich die Kandidaten in der Königsdisziplin beweisen, denn so können sie ihre Fähigkeiten ideal zur Schau stellen…“

Basierend auf diesem Hintergrund treten in diesem Spiel Pinguine auf einer Eisscholle zum Match gegeneinander an.

Spielmenü

Wird das Spiel gestartet, muss ein Spieler Enter auf der Tastatur oder A auf dem Kontroller drücken, um den Startbildschirm zu verlassen. Anschließend wird die Geschichte des Spiels eingeblendet. Diese kann übersprungen werden, wodurch man direkt in das Hauptmenü gelangt.

Im Hauptmenü gibt es drei Möglichkeiten: Das Spiel zu verlassen, ein Spiel zu beginnen oder auf die Optionen zu gehen.

In den Optionen kann die Geschichte noch einmal angeschaut werden. Unter „Credits“ befinden sich die Quellen der verwendeten Töne und Texturen, sowie die Angabe der Entwickler. Bei „Settings“ kann die Musiklautstärke und der Vollbildmodus ein- beziehungsweise ausgeschaltet werden. „Controls“ bietet den Spielern die Möglichkeit, die Tastenbelegung für die Steuerung mit der Tastatur und dem Kontroller nachzuschauen.

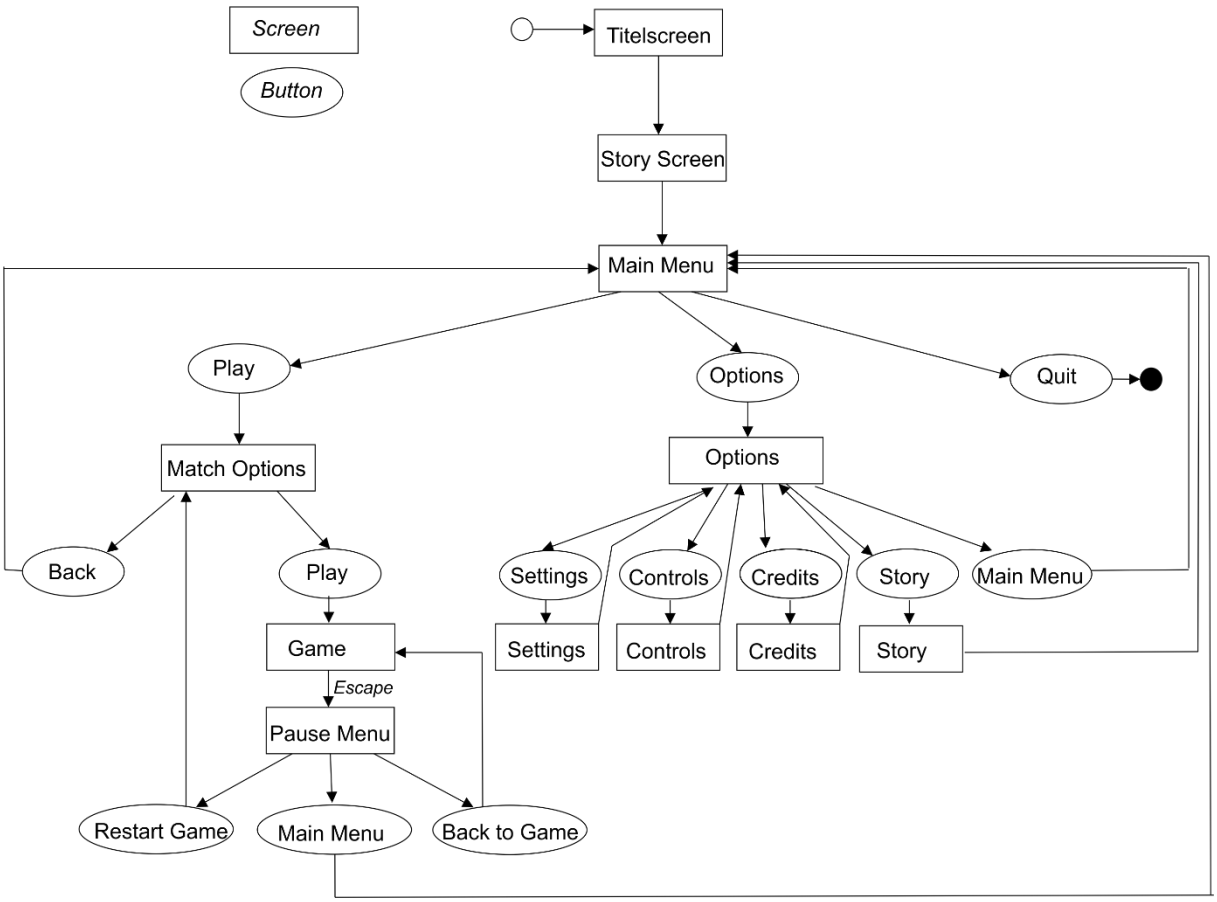
Über „Main Menu“ gelangt man zurück in das Hauptmenü. Startet man von dort ein Spiel, können Matchoptionen eingestellt werden. Dazu zählen die zu erreichende Gesamtpunktzahl, der Wind und die einzusammelnden Gegenstände.

Bestätigt man seine Auswahl, kommt vor Spielbeginn eine Aufforderung zum Schere-Stein-Papier. Dies ist ein einfaches und faires Konzept, um zu bestimmen, welcher Spieler Aufschlag hat. Aufschlag zu haben bedeutet immer einen Vorteil. Schere-Stein-Papier stellt ein randomisiertes, intransparentes und somit auch frustfreies Entscheidungssystem dar. Auch dient es als Einstieg in die kompetitive Atmosphäre eines Matches.

Nachdem beide Spieler ihre Eingabe getätigt haben, öffnet sich das zentrale Spiel. Der Bildschirm teilt sich in einen Split-Screen, auf welchem die Spieler ihren Pinguin-Charakter isometrisch von hinten sehen.

Das Spiel kann durch Drücken der „Escape“-Taste angehalten werden. Jederzeit kann man zurück in das aktuelle Spiel gelangen, alternativ ein neues Spiel starten oder auch das Hauptmenü aufrufen.

Verknüpfungsübersicht der Menüs und Screens:



Spielwelt

Das Spielfeld ist der zentrale Ereignisort des Spiels.

Es handelt sich um eine Eisscholle unter Sternenhimmel, welcher durch eine Textur auf einem sich langsam rotierendem Skydome realisiert wurde. Auf der Eisscholle sind rot die Spielfeldbegrenzungen eingezeichnet, und auf der Mittellinie steht das ebenfalls aus Eis gemachte Netz über welches der Ball geschlagen werden muss.

Am Rande des Spielfelds, mittig und nahe am Netz stehen auf beiden Seiten Fahnen, welche die Windrichtung signalisieren.

Neben dem Netz und auf Höhe der Mittellinie steht ein Acagamics-Männchen mit einer Trillerpfeife, welches den Schiedsrichter des Matches darstellt. An den Rändern der Spielhälften stehen weitere Pinguine. Diese sind das Publikum und jubeln für ihren jeweiligen Spieler.

Zum Spielstart steht der Pinguin, welcher Aufschlag hat, an der hinteren Spielfeldbegrenzung. Der gegnerische Spieler startet in der Mitte seiner Spielfeldhälfte. Beide sind Richtung Netz gewandt.

Die Spieler können sich drehen um die Schlagrichtung zu beeinflussen, und es gibt einen Pfeil auf dem Boden vor jedem Spieler, welcher sichtbar ist, sobald man sich dreht. Die Farbe des Pfeils gibt außerdem an, wann ein Schlag möglich ist.

Wenn man nicht gerade Aufschlag hat, kann der Pinguin-Charakter frei durch das Spielfeld auf seiner Seite des Netzes watscheln.

Kernmechanik und Spielhandlung

Die Charaktere der Spieler können sich während dem Spiel nur innerhalb ihrer Feldbegrenzung bewegen. Innerhalb des Feldes stehen vier Bewegungsrichtungen zur Verfügung: links, rechts, vor und zurück. Dies erfolgt entweder über den linken Thumbstick oder über die Pfeiltasten der Tastatur. Zusätzlich haben die Spieler mit A (bei Xbox Kontrollern) oder X (bei Playstation Kontrollern), beziehungsweise der Leertaste, die Möglichkeit zu springen.

Erreicht ein Spieler den Ball, muss er den Ball bevor dieser den Boden berührt zurück in die Hälfte seines Gegners schlagen. Berührt der Ball den Boden innerhalb des eigenen Feldes, bekommt der gegnerische Spieler einen Punkt. Berührt der Ball den Boden außerhalb des Feldes, ist dies ein Fehler des Spielers mit dem letzten Ballkontakt. Der Gegner bekommt einen Punkt.

Erlangt ein Spieler die zu Beginn eingestellte Gesamtpunktzahl des Matches, so hat dieser gewonnen.

Um den Ball erfolgreich in die gegnerische Hälfte zu schlagen, stehen den Spielern zwei Möglichkeiten zur Verfügung. Sie können mit dem left Bumper und right Bumper, beziehungsweise E und Q sowohl leicht, als auch stark schlagen. Mit dem rechten Thumbstick oder den Pfeiltasten könne sie die Flugrichtung des Balls einstellen. Dabei ist eine Rotation um 45° in beide Richtungen möglich.

Bildschirmdarstellung

Wird ein Match begonnen, teilt sich der Bildschirm. So sieht jeder Spieler seinen Charakter von hinten aus der isometrischen Perspektive. Dadurch wird für jeden Spieler eine gute Sicht gewährleistet, denn beide Spieler haben den gleichen Blickwinkel und sehen ihr jeweiliges Feld auf die gleiche Weise, wodurch Fairness erhalten bleibt. Während dem Spiel ist es nicht möglich, die Kamera zu drehen oder zu zoomen. Ist das Match beendet und die „Winner“ / „Loser“ Anzeige verschwunden, wird der Split -screen aufgehoben. Die Kamera bewegt sich nun um das Spielfeld.

Zusätzliche Spielfunktionen

Vor Beginn eines Matches haben die Spieler die Gelegenheit, das Spiel spannender und abwechslungsreicher zu gestalten, indem sie drei zusätzliche Funktionen einschalten.

Zu dem normalen Spiel kann man einen Wind einschalten, dessen Richtung im Spiel die Ballflugbahn beeinflusst und den Spielern durch Fahnen am Feldrand angezeigt wird.

Des Weiteren können vier Arten von Gegenständen eingesammelt werden. Zwei vergrößern oder verkleinern den Ball. Die anderen zwei beziehen sich auf die Laufgeschwindigkeit der Charaktere. Eines verringert die Geschwindigkeit des Gegners, das andere erhöht die eigene Geschwindigkeit.

Feedback

Um dem Spieler Rückmeldung über seine Aktionen zu geben, beziehungsweise um ihm seinen Spielfortschritt zu verdeutlichen, gibt es in „Volamus“ ein optisches und auditives Feedback.

In den Menüs gibt es neben der Hintergrundmusik auch ein Klickgeräusch, welches jedes Mal ertönt, wenn der Spieler einen anderen Button auswählt. Um zu signalisieren, welcher Button momentan aktiv ist, haben diese einen Fading-Effekt, der den Button subtil blinken lässt. Wo Spieleinstellungen vorgenommen werden können, ist ein kleiner Pfeil vorhanden, der die momentane Auswahl anzeigt.

Im Spiel selbst gibt es verschiedenste Geräusche. Beim Schlagen des Balls erklingt ein Schlaggeräusch. Jubel und der Pfiff des Schiedsrichters ertönen beim Erhalten eines Punktes. Am oberen Bildschirmrand gibt es außerdem Punktetafeln, welche den momentanen Punktestand anzeigen.

Während dem Spiel sorgen zusätzlich Animationen für ein visuelles Feedback. Nach vorne rotierende Flügel dienen dabei als Schlaganimation. Wird der Spielcharakter bewegt, sorgt eine minimale

Rotation nach links und rechts dafür, dass der Charakter einen watschelnden Eindruck macht. Auch der Ball ist durch eine Rotation um sich selbst animiert. Fahnen an der Mittellinie verdeutlichen dem Spieler den Einfluss des Windes. Dabei drehen sie sich in die jeweilige Windrichtung.

Damit der Spieler weiß, in welche Richtung er beim nächsten Schlag werfen wird und wann er überhaupt schlagen kann, gibt es zusätzlich zur Rotation des Pinguin Modells in die entsprechende Richtung eine Pfeilanzeige. Dieser Pfeil befindet sich auf dem Boden vor dem Modell, welchen man bei Rotation sehen kann. Der Pfeil ist standardmäßig rot, wird jedoch grün wenn ein Schlag möglich ist. Der Pfeil des gegnerischen Pinguins auf der anderen Seite des Spielfelds ist dabei nicht sichtbar.

Wenn ein Spieler nur noch einen Punkt vom Sieg entfernt ist, erscheint ein großer Text, der den Matchball verkündet. Außerdem ändert die Punktanzeige ihre Farbe von weiß nach gelb.

Ist das Spiel beendet, taucht in der Bildschirmhälfte des Siegers ein Banner mit „WINNER“ und in der Hälfte des Verlierers ein Banner mit „LOSER“ auf. Zusätzlich hüpfen der Pinguin des Gewinners sowie die Zuschauer auf seiner Seite vor Freude auf der Stelle. Am Rand seiner Spielfeldhälfte gibt es ein kleines Feuerwerk. Im Gegensatz dazu lässt der Pinguin des Verlierers seine Flügel hängen und bewegt sich nicht weiter. Nach einem kurzen Moment wird der Split-Screen aufgehoben und die Kamera beginnt sich um das Spielfeld mit Blickrichtung zum Mittelpunkt zu drehen.

Technische Umsetzung

Entwicklungsumgebung und Voraussetzungen

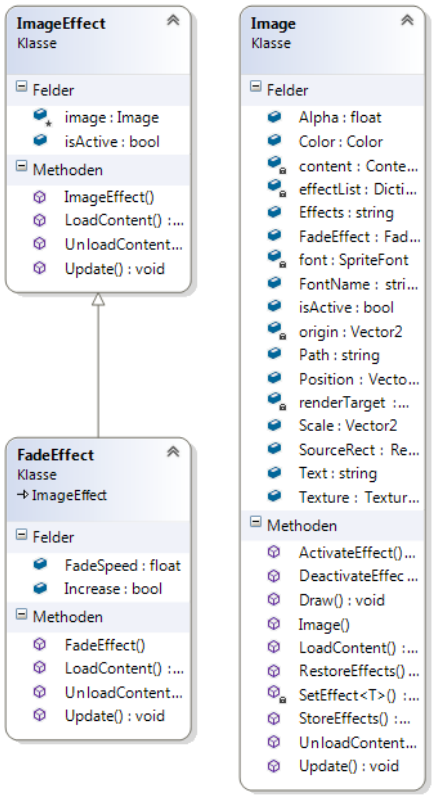
Das Spiel, welches eine Windows Applikation ist, wurde in der Entwicklungsumgebung Visual Studio mit dem Framework Monogame unter der Nutzung von Windows 7 und Windows 10 implementiert.

Um das Spiel spielen zu können, wird Windows 7 mit 64-Bit oder eine modernere Windows Version vorausgesetzt. Der benötigte Speicherplatz beträgt 500MB und es ist ein Arbeitsspeicher von mindestens 256 MB RAM erforderlich. Außerdem muss DirectX 11 oder höher, beziehungsweise OpenGl 3.0, und eine Grafikkarte vorhanden sein.

Bei dem Spiel handelt es sich um ein Zwei-Spieler-Spiel. Die Bedienung kann durch einen Kontroller und einer Tastatur erfolgen. Jedoch werden für eine angenehme Steuerung mit der Tastatur zu viele Tasten benötigt, weshalb empfohlen wird, das Spiel mit zwei Kontrollern zu bedienen.

Image und Image-Effect

Die Klasse „Image“ verbindet 2D-Texturen und 2D-Schrift miteinander. Dabei wird die Schrift immer mittig in die Textur „geschrieben“. Ist die Textur kleiner als die Schriftgröße des Schriftzugs, wird diese skaliert, sodass sie nicht mehr zu klein ist. Images können Image-Effects zugeordnet werden. Dafür ist die Klasse „ImageEffect“ die Basisklasse von der alle Image-Effects erben. Im Moment wird nur „FadeEffect“ verwendet, welches den Alpha-Wert des Images, also die Sichtbarkeit, skaliert und somit manipuliert.



Game States

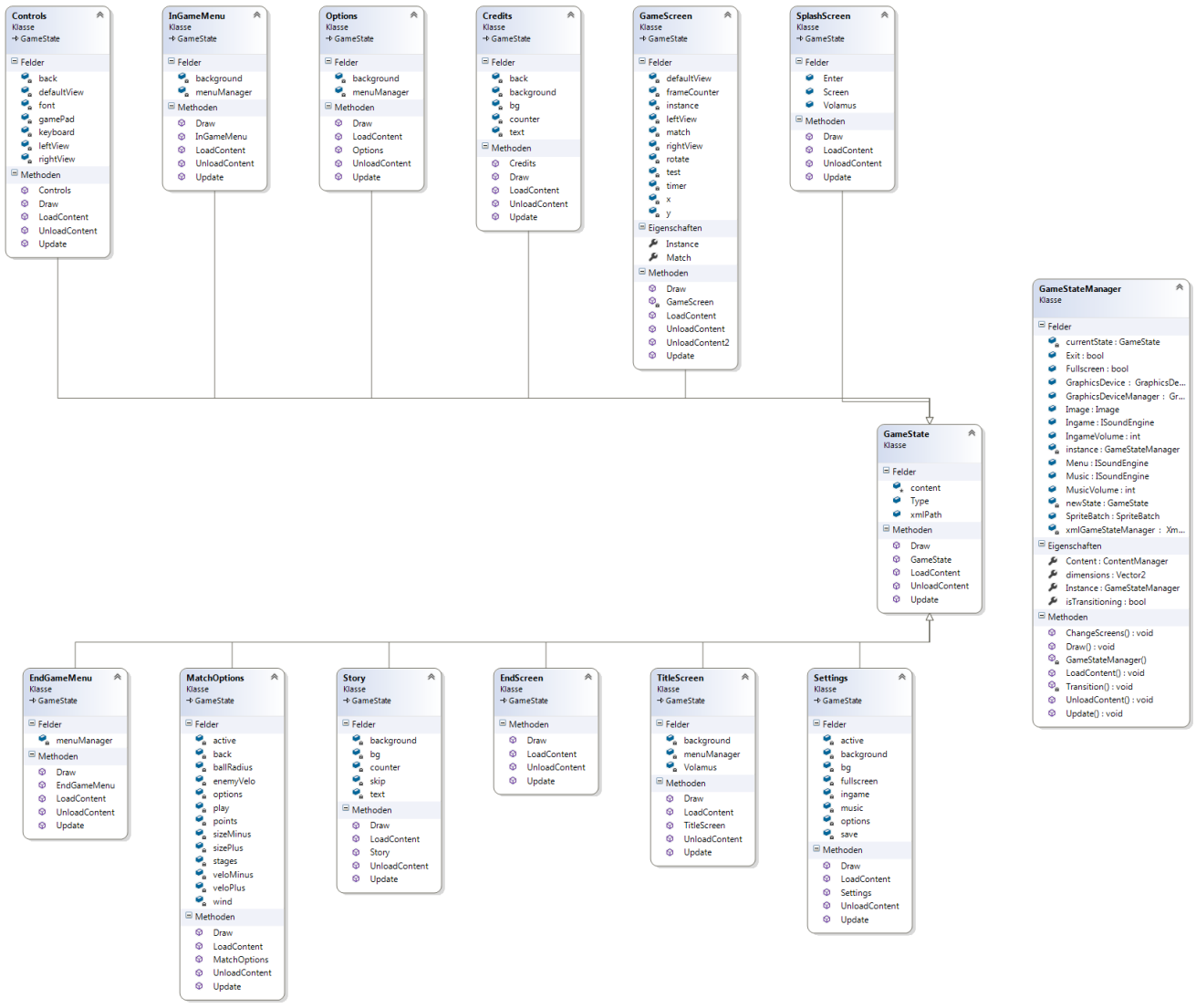
Die Basisklasse „GameState“ bildet die Grundlage aller Game States. Alle Game States erben von ihr.

Jeder Game State hat alle Methoden von der Klasse „GameState“ zu implementieren: Das Laden und Löschen vom jeweiligen Content, das Updaten und das Zeichnen.

Um den aktuellen Game State zu speichern und zu wechseln, wurde die Klasse „GameStateManager“ programmiert. Sie steuert außerdem das Aufrufen der Methoden des aktuellen Game States, sowie den Übergang von einem Game State zum nächsten. Es sollte kein „harter“ Übergang werden, deshalb wird der aktuelle Game State beim Wechsel erst nach und nach ausgeblendet. Dies

geschieht, indem ein schwarzes Bild langsam immer stärker eingeblendet wird. Danach wird dieses schwarze Bild ausgeblendet und der neue Game State wird angezeigt. So entsteht ein optisch angenehmer Wechsel.

Es gibt zwei Arten von Game States: einige Game States sind einfache Screens, andere Game States haben dazu noch ein Menü und haben daher einen Menü – Manager implementiert, der das Menü mithilfe von XML-Dateien lädt.



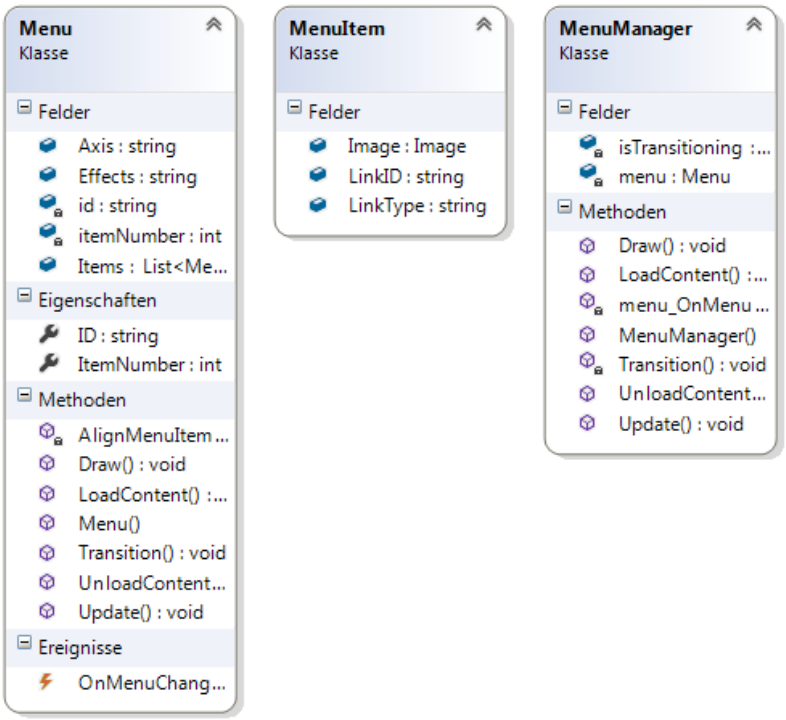
Menü

Ein Menü, welches unter der Klasse „Menu“ implementiert wurde, besteht aus einer Liste von „MenuItem“. Diese besitzen jeweils ein Image, ein LinkType und eine LinkID. Der LinkType gibt an ob der nachfolgende Game State auf den man mithilfe des MenuItems wechselt, ein Screen oder ein Menü ist. Die LinkID gibt an auf welchen Game State man genau wechselt und das Image bestimmt sowohl die Textur, als auch die Schrift innerhalb der Textur. Dies zusammen bildet den Button, den man im Spiel sieht.

Die Klasse „Menu“ beinhaltet nicht nur MenuItems, sondern steuert auch den Wechsel zwischen diesen, indem es durch die Liste je nach Eingabe iteriert.

Besonders zu erwähnen wäre noch das Verhalten des momentan aktiven MenuItems: Das Image des aktiven MenuItems hat dann einen Fade-Effect aktiviert. Das heißt, es wird langsam aus- und wieder eingeblendet. Der Fade-Effect wird beim Wechsel des alten MenuItems ausgeschaltet und beim neuen MenuItem eingeschaltet.

Um bessere Übersicht und leichtere Änderungen zu garantieren, haben die Game States, die ein Menü beinhalten, eine Instanz der Klasse „MenuManager“ implementiert. Diese lädt aus XML-Dateien mithilfe von XML-Serialization das Menü mit den MenuItems. Das Menü ordnet dann die MenuItems richtig an, sodass das gesamte Menü zentriert ist.



Steuerung

Da es sowohl möglich sein muss mit Kontroller oder Tastatur seinen Spieler zu steuern, wurden die Klassen „KeyboardControl“ und „ControllerControl“ programmiert. Beide beinhalten alle Knöpfe die man zum steuern des Spieles benötigt, allerdings jeweils von anderen Datentypen, nämlich Keys und Buttons. Benötigte Knöpfe sind: Up, Down, Left, Right, DirectionLeft, Direction Right, Jump, StrongServe und WeakServe.

Da es leider keine gemeinsame Basisklasse von diesen beiden gibt, konnte die Steuerung nicht vereinfacht werden. Um später jedoch die Erweiterungsmöglichkeit zu besitzen, dass der Spieler seine Steuerung ändern kann, wurde das Einlesen und Schreiben der Steuerung mithilfe von XML - Dateien eingeführt. Dabei war zu beachten, dass dies nicht mithilfe von einfacher XML-Serialization funktioniert, da sowohl Keys als auch Buttons kein Unterstützer Datentyp sind. Deshalb wird hier ein IntermediateSerializer verwendet.

Einstellungen

In dem GameState „Settings“ kann man einige Einstellungen vornehmen und diese mit dem Button „Save & Back“ in eine XML-Datei mithilfe von XML-Serialization speichern. Dabei werden die aktuell angezeigten Werte von den drei Optionen für das momentan laufende Spiel erst übernommen und dann gespeichert.

Jede der drei Optionen: „Ingame Volume“, „Music Volume“ und „Fullscreen“ besitzen ein Array, das mit den möglichen Auswahlmöglichkeiten gefüllt ist. Mithilfe von Eingaben können diese durchschalten werden.

Beim Starten des Spieles werden die Einstellungen für das Spiel aus der XML-Datei geladen und übernommen.

Ball

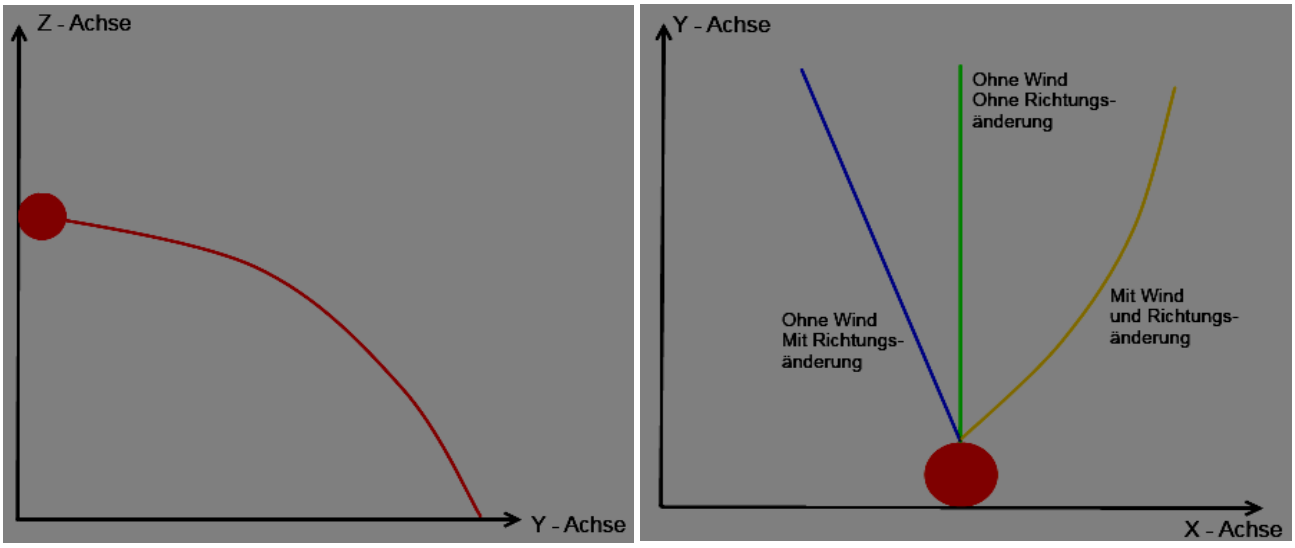
Das wichtigste Element des Spieles ist der Ball und seine korrekte Flugbahn. Diese wird mithilfe von Wurfparabeln und der ebenso genannten Klasse „Parabel“ gewährleistet. Der Ball besitzt eine aktive Parabel, die bei Änderung der Flugbahn neu gesetzt wird, sodass immer nur diese eine aktive Parabel betrachtet und aktuell gehalten werden muss. Bei jeder aktiven Parabel des Balles handelt es sich um die Wurfparabel eines schrägen Wurfes ohne die Beachtung und Mitberechnung des Luftwiderstandes.

Zu beachten ist dabei, dass der Ball nicht nur gerade fliegt. Indem der Spieler sich dreht und durch den Wind, der aktiv sein kann, ändert sich also nicht nur der Z – Wert (Höhe) und Y-Wert (Tiefe) des Balles sondern auch der X-Wert(Breite).

Zur Veranschaulichung:

Hierbei handelt es sich lediglich um eine Abstraktion zur Erklärung, weder Skalierung noch Berechnung ist korrekt – außerdem handelt es sich hier um Beispiele und keinen generellen Fall.

(Beispielsweise: Der Wind könnte den Ball auch nach rechts statt links abdrehen lassen, etc.)



Wind

Der Wind besteht generell aus seiner Windrichtung und seiner Windstärke. Beides wird berücksichtigt.

Die Möglichkeit des Änderns der Windstärke für den Spieler ist im Moment nicht vorhanden – er kann den Wind lediglich ein- und ausschalten. Jedoch ist es möglich, dieses Feature schnell hinzuzufügen, da es schon einmal vorhanden war, es aber als zu schwer zu Skalieren eingestuft wurde. Die Windstärke ist also ein fixer Wert, sobald der Wind eingeschaltet ist.

Wird der Wind aktiviert, wird zufällig am Anfang des Matches die Windrichtung als Winkel zwischen 0° und 360° bestimmt. Da sich auch die Windrichtung mit der Zeit ändert, wurde dies auch implementiert. Statt es jedoch mit der Zeit zu verknüpfen, wird der Winkel der Windrichtung aktualisiert, wenn der Spieler den Ball schlägt, der Ball den Boden oder den Spieler berührt. So wird das unnatürliche Wackeln des Balles in der Luft vermieden, das durch Änderung der Windrichtung im Flug des Balles entstehen würde.

Die Windrichtung wird zufällig geändert: Es besteht die 50% Chance, dass der Winkel der Windrichtung um 2° erhöht wird und die 50% Chance, dass der Winkel der Windrichtung um 2° verkleinert wird.

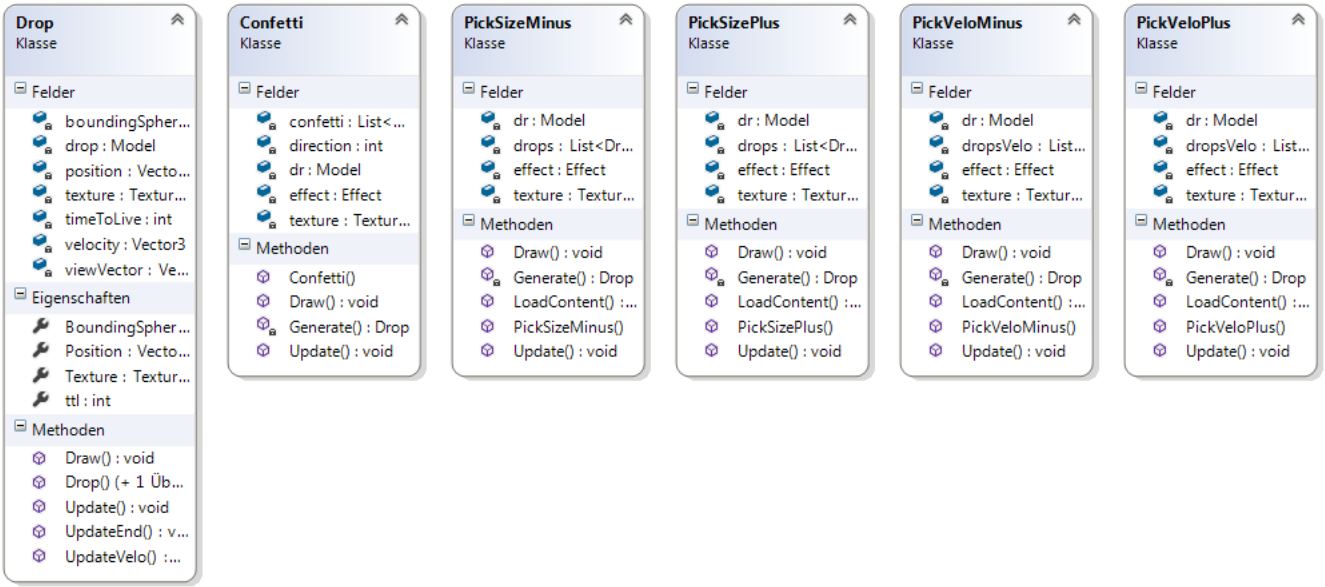
VelocityDrops und SizeDrops

Die Grundlage des Features, dass während des Spiels Gegenstände eingesammelt werden können, ist die Klasse „Drops“. Drops sind Objekte, für die einmal eine zufällige Position auf dem Spielfeld und eine zufällige Lebenszeit bestimmt werden. Bis diese Lebenszeit abgelaufen ist oder bis ein Spieler den Drop eingesammelt hat, befindet sich dieser Drop an seiner Position.

VelocityDrops sowie SizeDrops besitzen einen Dropgenerator, der eine zufällige Anzahl an Drops generiert. Damit das Feld nicht überfüllt ist, ist die maximale Anzahl an Drops in einer Liste drei.

Es gibt zwei unterschiedliche Arten von VelocityDrops, die auch verschiedene Auswirkungen beim Einsammeln haben. Solche, die die Bewegungsgeschwindigkeit des Gegners senken, was bis auf ein Minimum begrenzt ist. Und solche, die die eigene Bewegungsgeschwindigkeit erhöhen, bis auf ein Maximum.

Ebenso gibt es zwei unterschiedliche Arten von SizeDrops, die unterschiedliche Effekte beim Einsammeln hervorrufen. Die eine Art, die den Ballradius vergrößert und die andere Art, die den Ballradius verringert. Auch hier sind beide durch ein Minimum beziehungsweise Maximum begrenzt.



Kollision

Es mussten mehrere Kollisionen zwischen mehreren Objekten beachtet und behandelt werden.

Am einfachsten gestaltete sich die Kollision zwischen Spielermodell und Feldgrenzen, die der Spieler nicht überschreiten kann. Bei der Bewegung des Spielers wurde abgefragt ob dieser sich bereits am Rande des Spielfelds befindet. Falls dies der Fall ist, darf er sich nicht mehr in Richtung der Feldgrenze bewegen. Dies funktioniert mittels einer einfachen if-Abfrage. Dabei handelt es sich um die linke, rechte, hintere und vordere Feldgrenze des jeweiligen Spielers.

Um die jeweiligen Kollisionen zwischen Spieler, Netz, Ball und Drops sowie Boden umzusetzen, wurden sowohl Bounding-Boxen, Bounding-Spheres als auch Planes verwendet.

Der Ball besitzt eine Bounding-Sphere, die Monogame von alleine berechnen kann welche abgerufen werden kann. Der Boden wurde als Z = 0 Plane behandelt, da unser Boden bei Z = 0 festgelegt worden ist. Monogame stellt dabei eine Methode zur Verfügung, die überprüft ob eine Bounding-Sphere und eine Plane sich berühren und/oder schneiden. Diese wird zur Kollisionsberechnung verwendet.

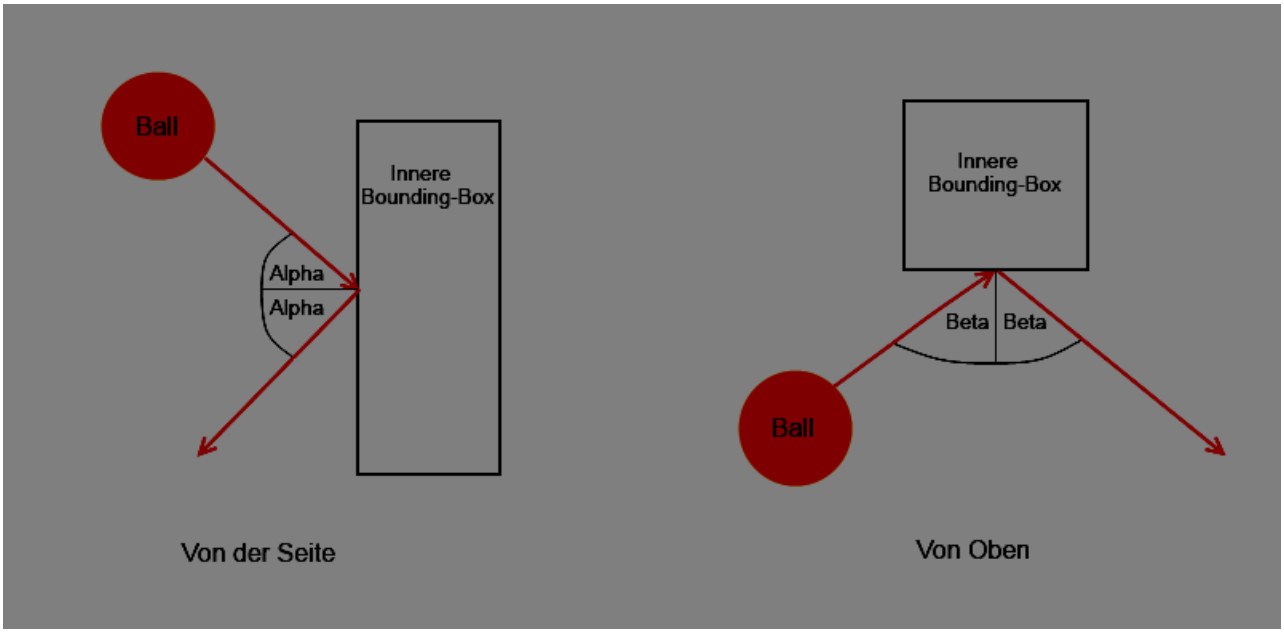
Das Netz besitzt eine Bounding-Box, der Spieler besitzt zwei Bounding-Boxen. Dabei war zu beachten, dass Monogame Bounding-Boxen nicht von sich aus zu einem Modell erstellen kann. Eine Bounding-Box kann nur aus einem minimalem Punkt und einem maximalem Punkt, jeweils bestehend aus den drei Koordinaten, berechnet werden. Dazu wird nach der Rotation, Translation und Skalierung in die Welt Koordinaten eines jeweiligen Modells jedes einzelne Model-Mesh-Parts durchgegangen und das Minimum und Maximum gesucht. Hieraus wird dann die Bounding-Box erstellt.

Die Kollision zwischen Ball und Netz war damit auch recht intuitiv, da Monogame auch hier eine Methode bereitstellt, ob eine Bounding-Sphere eine Bounding-Box schneidet. War dies zwischen Ball und Netz der Fall, muss der Ball in den richtigen Winkeln und mit verringerter Geschwindigkeit abprallen. Die Regel „Einfallswinkel ist gleich Ausfallswinkel“ wird dabei berücksichtigt.

Das wichtigste Element des Spiels ist dennoch die Kollision zwischen Spieler und Ball. Dazu hat der Spieler zwei Bounding-Boxen bekommen, eine äußere Bounding-Box und eine innere Bounding-Box. Die Innere wird mit dem oben bereits beschriebenen Konzept berechnet, die Äußere ist die Innere Bounding-Box addiert beziehungsweise subtrahiert mit einem bestimmten Offset. Wie bereits erwähnt stellt Monogame zur Überprüfung, ob eine Bounding-Sphere eine Bounding-Box schneidet eine Methode zur Verfügung. Diese wird auch zwischen Spieler und Ball verwendet. Berührt oder schneidet nun die Bounding-Sphere des Balles die äußere Bounding-Box des Spielers darf der Spieler den Ball schlagen. Visuell wird dies unterstützt, indem der Pfeil des Spielers grün eingefärbt wird.

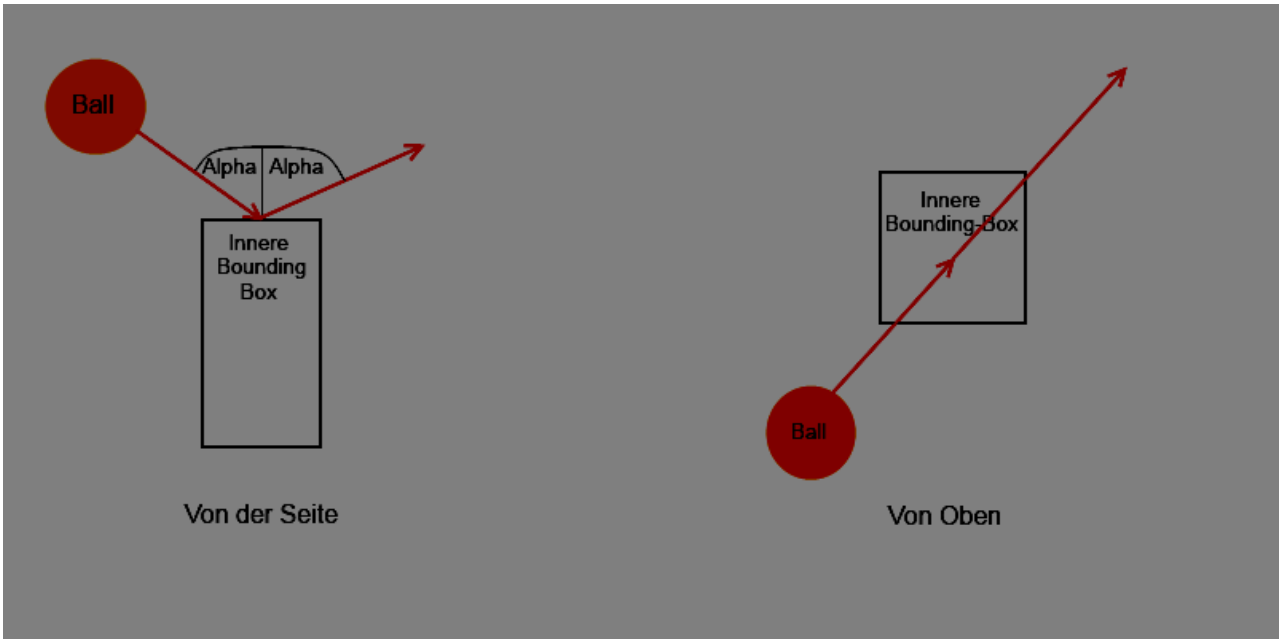
Berührt die Bounding-Sphere des Balles die innere Bounding-Box des Spielers wird der Ball je nach Einfallswinkel und Fläche der Bounding-Box, die er berührt, abprallen.

Zur Veranschaulichung (Skizzen vereinfacht und nicht maßstabsgerecht):



Der Ball prallt hier von der vorderen Seite der inneren Bounding-Box des Spielers ab.

Genau so funktioniert es bei der Kollision mit den beiden Seiten und Hinten. Bei der Kollision mit der oberen Seite prallt der Ball wie folgt ab:

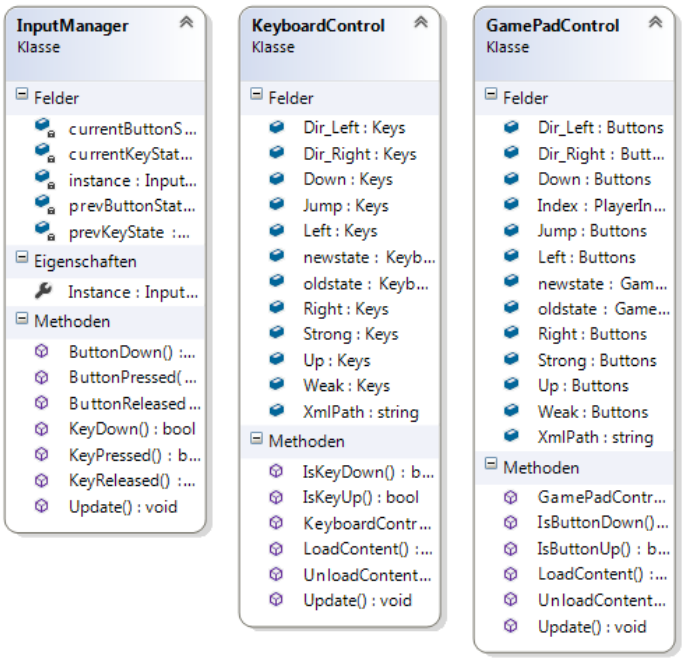


Shader

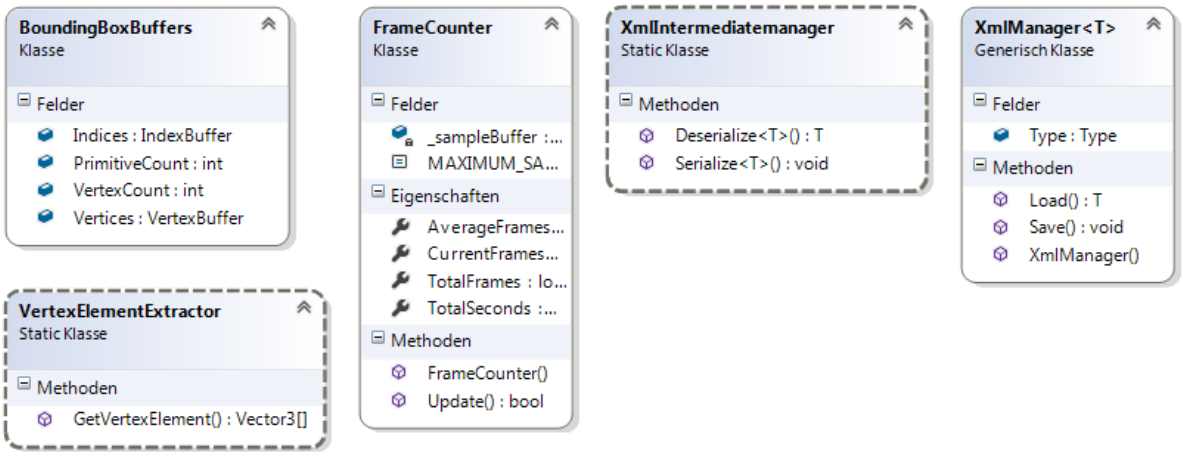
Als Grundlage für unseren Shader benutzen wir das Phong'sche Beleuchtungsmodell. Hierbei ist zu beachten, dass wir nicht nur eine Lichtquelle haben, sondern mehrere.

Restliches Klassendiagramm

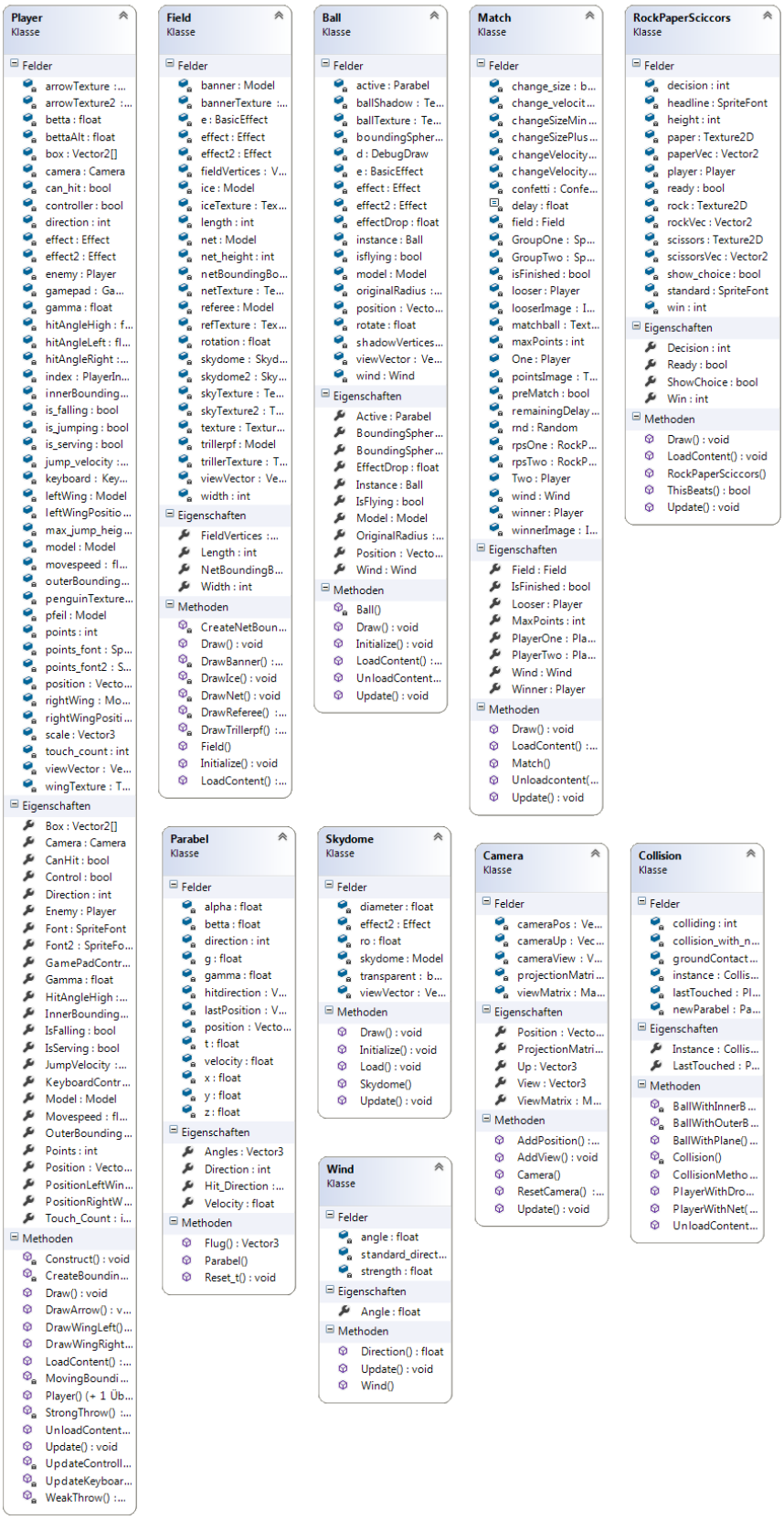
„Controls“:



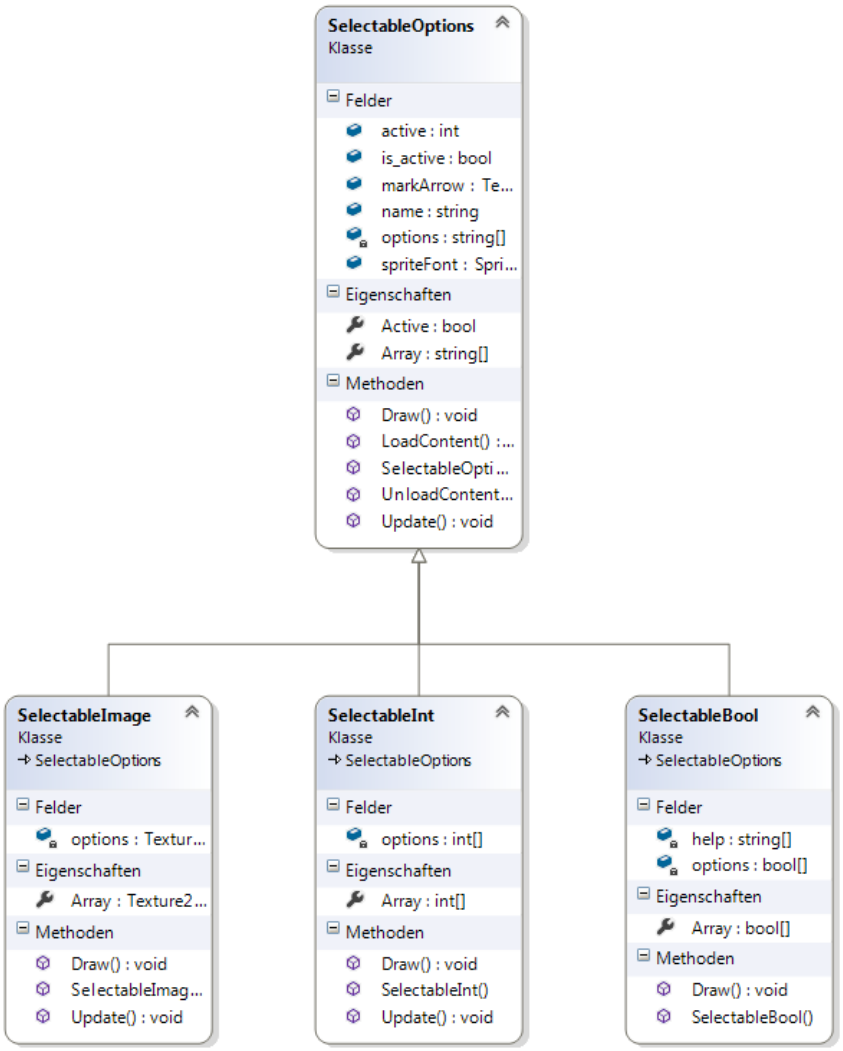
„Helper“:



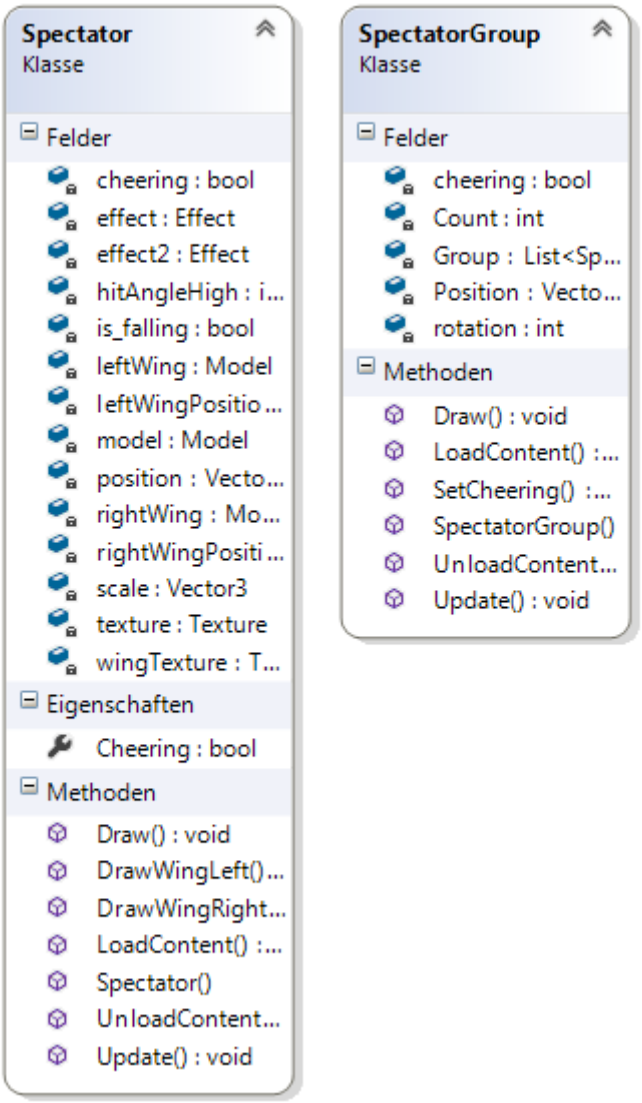
„Main“:


Player
Klasse
Felder
arrowTexture :...
arrowTexture2 :...
betta : float
bettaAlt : float
box : Vector2[]
camera : Camera
can_hit : bool
controller : bool
direction : int
effect : Effect
effect2 : Effect
enemy : Player
gamepad : Ga...
gamma : float
hitAngleHigh : f...
hitAngleLeft : fl...
hitAngleRight :...
index : PlayerIn...
innerBounding...
is_falling : bool
is_jumping : bool
is_serving : bool
jump_velocity :...
keyboard : Key...
leftWing : Model
leftWingPositio...
max_jump_heig...
model : Model
movespeed : fl...
outerBounding...
penguinTexture...
pfeil : Model
points : int
points_font : Sp...
points_font2 : S...
position : Vecto...
rightWing : Mo...
rightWingPositi...
scale : Vector3
touch_count : int
viewVector : Ve...
wingTexture : T...
Eigenschaften
Box : Vector2[]
Camera : Camera
CanHit : bool
Control : bool
Direction : int
Enemy : Player
Font : SpriteFont
Font2 : SpriteFo...
GamePadContr...
Gamma : float
HitAngleHigh :...
InnerBounding...
IsFalling : bool
IsServing : bool
JumpVelocity :...
KeyboardContr...
Model : Model
Movespeed : fl...
OuterBounding...
Points : int
Position : Vecto...
PositionLeftWin...
PositionRightW...
Touch_Count : i...
Methoden
Construct() : void
CreateBoundin...
Draw() : void
DrawArrow() : v...
DrawWingLeft()...
DrawWingRight...
LoadContent() :...
MovingBoundi...
Player() (+ 1 Üb...
StrongThrow() :...
UnloadContent...
Update() : void
UpdateControll...
UpdateKeyboar...
WeakThrow() :...
Field
Klasse
Felder
banner : Model
bannerTexture :...
e : BasicEffect
effect : Effect
effect2 : Effect
fieldVertices : V...
ice : Model
iceTexture : Tex...
length : int
net : Model
net_height : int
netBoundingBo...
netTexture : Te...
referee : Model
refTexture : Tex...
rotation : float
skydome : Skyd...
skydome2 : Sky...
skyTexture : Te...
skyTexture2 : T...
texture : Textur...
trillerpf : Model
trillerTexture : T...
viewVector : Ve...
width : int
Eigenschaften
FieldVertices :...
Length : int
NetBoundingB...
Width : int
Methoden
CreateNetBoun...
Draw() : void
DrawBanner() :...
DrawIce() : void
DrawNet() : void
DrawReferee() :...
DrawTrillerpf() :...
Field()
Initialize() : void
LoadContent() :...
Ball
Klasse
Felder
active : Parabel
ballShadow : Te...
ballTexture : Te...
boundingSpher...
d : DebugDraw
e : BasicEffect
effect : Effect
effect2 : Effect
effectDrop : float
instance : Ball
isflying : bool
model : Model
originalRadius :...
position : Vecto...
rotate : float
shadowVertices...
viewVector : Ve...
wind : Wind
Eigenschaften
Active : Parabel
BoundingSpher...
BoundingSpher...
EffectDrop : float
Instance : Ball
IsFlying : bool
Model : Model
OriginalRadius :...
Position : Vecto...
Wind : Wind
Methoden
Ball()
Draw() : void
Initialize() : void
LoadContent() :...
UnloadContent...
Update() : void
Match
Klasse
Felder
change_size : b...
change_velocit...
changeSizeMin ...
changeSizePlus ...
changeVelocity...
changeVelocity...
confetti : Confe...
delay : float
field : Field
GroupOne : Sp...
GroupTwo : Sp...
isFinished : bool
looser : Player
looserImage : I...
matchball : Text...
maxPoints : int
One : Player
pointsImage : T...
preMatch : bool
remainingDelay ...
rnd : Random
rpsOne : RockP...
rpsTwo : RockP...
Two : Player
wind : Wind
winner : Player
winnerImage : I...
Eigenschaften
Field : Field
IsFinished : bool
Looser : Player
MaxPoints : int
PlayerOne : Pla...
PlayerTwo : Pla...
Wind : Wind
Winner : Player
Methoden
Draw() : void
LoadContent() :...
Match()
Unloadcontent(...
Update() : void
RockPaperSciccors
Klasse
Felder
decision : int
headline : SpriteFont
height : int
paper : Texture2D
paperVec : Vector2
player : Player
ready : bool
rock : Texture2D
rockVec : Vector2
scissors : Texture2D
scissorsVec : Vector2
show_choice : bool
standard : SpriteFont
win : int
Eigenschaften
Decision : int
Ready : bool
ShowChoice : bool
Win : int
Methoden
Draw() : void
LoadContent() : void
RockPaperSciccors()
ThisBeats() : bool
Update() : void
Parabel
Klasse
Felder
alpha : float
betta : float
direction : int
g : float
gamma : float
hitdirection : V...
lastPosition : V...
position : Vecto...
t : float
velocity : float
x : float
y : float
z : float
Eigenschaften
Angles : Vector3
Direction : int
Hit_Direction :...
Velocity : float
Methoden
Flug() : Vector3
Parabel()
Reset_t() : void
Skydome
Klasse
Felder
diameter : float
effect2 : Effect
ro : float
skydome : Model
transparent : b...
viewVector : Ve...
Methoden
Draw() : void
Initialize() : void
Load() : void
Skydome()
Update() : void
Camera
Klasse
Felder
cameraPos : Ve...
cameraUp : Vec...
cameraView : V...
projectionMatri...
viewMatrix : Ma...
Eigenschaften
Position : Vecto...
ProjectionMatri...
Up : Vector3
View : Vector3
ViewMatrix : M...
Methoden
AddPosition() :...
AddView() : void
Camera()
ResetCamera() :...
Update() : void
Collision
Klasse
Felder
colliding : int
collision_with_n...
groundContact...
instance : Collis...
lastTouched : Pl...
newParabel : Pa...
Eigenschaften
Instance : Collis...
LastTouched : P...
Methoden
BallWithInnerB...
BallWithOuterB...
BallWithPlane() ...
Collision()
CollisionMetho...
PlayerWithDro...
PlayerWithNet(...
UnloadContent...
Wind
Klasse
Felder
angle : float
standard_direct...
strength : float
Eigenschaften
Angle : float
Methoden
Direction() : float
Update() : void
Wind()

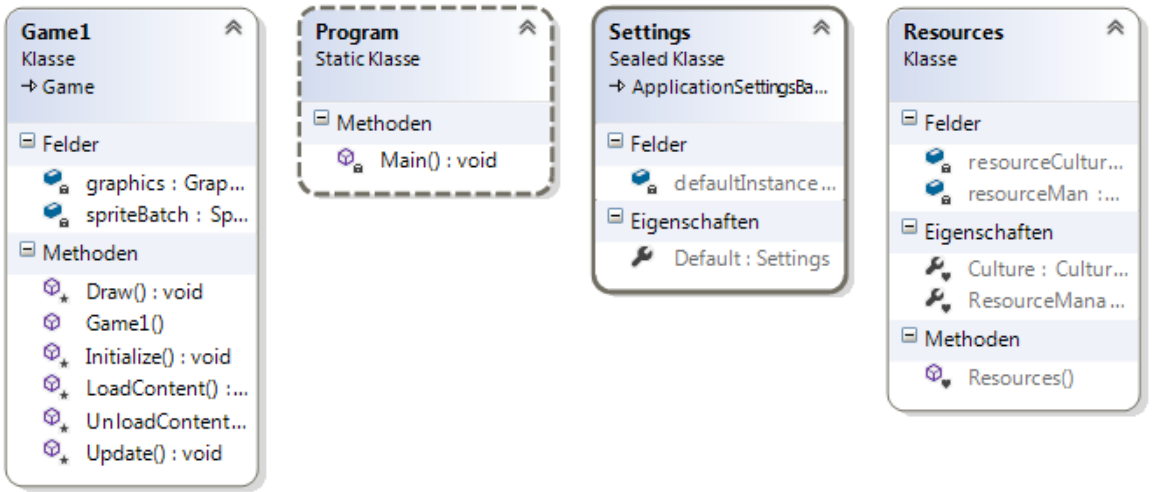
„SelectableOptions“:



„Spectator“:



„System“:



Verwendete Software und Bibliotheken

Modelle und Graphiken

Für das Spiel wurde ein Modell für den Pinguin, seine Flügel, den Ball, das Netz, die Eisscholle, die Trillerpfeife des Schiedsrichters, die einzusammelnden Gegenstände, der Skydome und das Feuerwerk benötigt. Diese wurden alle mit Blender 2.77a erstellt.

Das Modell für das Acagamic-Männchen, welches den Schiedsrichter darstellt, wurde vom Kursleiter Gerd Schmidt zur Verfügung gestellt.

Das Logo, „Matchball“,„Winner“ und „Loser“ Banner wurden mit der Graphik-Software Corel Draw Essentials X5 erstellt.

Sound

Das Spiel beinhaltet unterschiedliche Sounds.

Die Hintergrundmusik während eines Matches wurde von einem Teammitglied mithilfe der Software MuLab FREE 7.0.47 komponiert.

Die Hintergrundmusik im Menü und die Soundeffekte stammen aus der Bibliothek „freefsx“.

Um die Sounds in das Spiel zu integrieren wurde die Audiobibliothek „Irrklang“ verwendet.

Technische Probleme

Da es für uns alle eine ganz neue Herausforderung war, ein Spiel zu programmieren, traten während der Entwicklung immer wieder Unsicherheiten und auch unerwartete Probleme auf.

Eine Schwierigkeit war die richtige Darstellung physikalischer Abläufe. Diese müssen korrekt simuliert werden, um sicherzustellen, dass der Spielspaß nicht unter unrealistischen, unerwarteten Ereignissen leidet. Dazu zählt insbesondere die Kollision des Balls mit den Spielcharakteren, dem Boden und dem Netz. Er muss im richtigen Winkel abprallen, damit eine wahrheitsgetreue Flugbahn entsteht.

Vor Beginn stellte sich das Team die richtige Skalierung der Bewegungen schwer vor, da hier darauf geachtet werden muss, das Spiel nicht zu einfach, aber auch nicht zu schwer zu machen. Jedoch entpuppte sich dies während der Implementierung als relativ einfach und problemlos.

Auch zeitlich genaue Zuweisungen von Soundeffekten zu der jeweiligen Aktion waren im Nachhinein einfach zu implementieren als gedacht.

Des Weiteren stellte sich heraus, dass es einige Probleme mit den Animationen in Monogame gibt. Deshalb wurde uns davon abgeraten, Animationen zu verwenden. Schlag-, Lauf- und Jubelanimationen wurden nun durch Rotation, Translation und Skalierung einzelner Objekte nachgeahmt.

Da auch mit Blender zum ersten Mal gearbeitet wurde, waren nicht alle Funktionen der Software klar oder bekannt. Teilweise wurde erst im Nachhinein festgestellt, welche Optionen benötigt werden oder was vermieden werden sollte. Beispielsweise mussten zum Schluss alle Modelle überarbeitet werden, damit die zugehörigen Texturen im Spiel auf diese geladen werden konnten.

Vor allem hatten wir Probleme mit dem Modell des Pinguins. Dieses hatte immer ein Loch im Bauch. Erst gegen Ende des Projekts wurde herausgefunden, dass es an der UV-Map des Modells lag. Einige Teile des Netzes vom Bauch befanden sich auf für die Rückseite vorhergesehenen Teil der UV-Map.

Das Modell des Pinguins bereitete uns auch bei der Berechnung der seiner Bounding Box Probleme. Bei allen anderen Modellen, bei denen die gleiche Berechnung vorgenommen wurde, wurde die dazugehörige Bounding Box korrekt erstellt. Aufgrund einer starken Skalierung des Pinguins, bevor er in das Spiel geladen wurde, wurde seine Bounding Box oftmals falsch berechnet, beziehungsweise skaliert. Da das Modell des Pinguins oft überarbeitet wurde, mussten wir uns mit diesem Problem häufiger auseinandersetzen, wodurch viel Zeit verloren ging.

Projektmanagement

Zu Beginn des Projektes wurde der Entschluss gefasst, kein Projektmanagementtool für die Organisation zu verwenden. Für diese Entscheidung gab es zwei ausschlaggebende Gründe.

Bei der Entwicklung des Prototypens stellte sich heraus, dass wir alle Interesse haben, gerne mitarbeiten und eine gute Kommunikation zwischen uns möglich ist.

Der zweite und maßgebliche Grund liegt jedoch in der mangelnden Erfahrung aller Mitglieder begründet. Da kein Teammitglied über Vorwissen in der Spieleprogrammierung verfügte, fiel es uns schwer, im Vorhinein einzuschätzen, was alles benötigt wurde. Vor allem die Komplexität und den Aufwand der einzelnen Aufgaben abzuschätzen, beziehungsweise welche Aufgaben es überhaupt

geben würde, war schwierig. Deshalb wurden Aufgaben relativ spontan verteilt, beziehungsweise zu Beginn auch oftmals gemeinsam erledigt.

Je mehr das Grundgerüst des Spiels stand, desto eher konnten wir einzelne Aufgaben verteilen. Jedoch nahmen sich alle Teammitglieder immer Zeit, einem anderen bei einem Problem oder einer Frage zu helfen.

Regelmäßige Treffen, die später aufgrund zu großer Distanzen durch regelmäßigen Austausch über den Messenger WhatsApp ersetzt wurden, sorgten dafür, dass Veränderungen immer sofort an jeden von uns mitgeteilt wurden. So wurden auch Ideen entwickelt, Vorgehensweisen diskutiert und Probleme gelöst.

Damit jedes Mitglied auf seinem eigenen Rechner arbeiten konnte, beziehungsweise damit jeder die aktuelle Version hatte, wurde zur Versionierung git benutzt. Über gitHub wurde ein privates Repository angelegt, auf welchem auch Issues zu einigen Aufgaben erstellt wurden. Mithilfe von TortoiseGit wurden die Versionen auf den einzelnen Rechnern aktualisiert.

Zielsetzung, Aufgabenverteilung, Meilensteine

Meilenstein I

Der erste Meilenstein am 13. Mai diente zur Teamplanung und –organisation. Gleichzeitig mussten die ersten Dokumente für die endgültigen Spiele vorgestellt werden. Zu diesem Zeitpunkt haben wir noch nicht programmiert, sondern hauptsächlich Ideen für „Volamus“ gesammelt, niedergeschrieben, vorgetragen und überarbeitet. Dies ermöglichte uns die Entwicklung einer genaueren Vorstellung, was jeder einzelne von uns von dem Projekt erhofft und was umsetzbar ist.

Meilenstein II

Zum zweiten Meilenstein, der am 3. Juni vorzustellen war, sollte das Spiel über die groben Bewegungsabläufe und Handlungen verfügen. Das Ziel war, dass es zwei Charaktere gibt, bei denen es sich aber noch nicht um fertige Modelle handeln musste. Diese sollten sich innerhalb ihres Feldes bewegen können. Dabei stand weder die Optik des Feldes noch die der Umgebung im Vordergrund.

Gemeinsam wurde ein Spieler in das Spiel programmiert, der sich innerhalb seines Feldes bewegen konnte. Das Spielfeld wurde errichtet und die Kamera so positioniert, dass beide Charaktere von hinten gut sichtbar waren. Des Weiteren konnte der Spieler den Ball mit zwei verschiedenen Schlägen schlagen und der Ball prallte bereits vom Boden ab (Mareen). Bevor Schläge implementiert werden konnten, mussten sich die Mitglieder mit der Physik einer Flugbahn auseinandersetzen (Mareen, Lars). Für ein kleines Menü informierte sich ein Mitglied über das Einlesen von xml-Dateien (Lars). Zusätzlich waren bereits eine vorläufige Version des Netzes und die erste Version des Pinguins vorhanden (Lena).

Meilenstein III

Auch in dieser Entwicklungsphase haben wir zu Beginn zusammen gearbeitet. Dabei stellten wir die Kollisionen fertig und optimierten diese, so dass der Abprallwinkel des Balls möglichst realistisch war. Die Größenanpassung des Spielfeldes erfolgte ebenso gemeinsam.

Nun erfolgte die Einbindung des fertig modellierten Spielfelds und des überarbeiteten Pinguinmodells in das Spiel (Lena).

Um das Spielen zu zweit zu erleichtern, wurde der Split Screen implementiert (Lars). Die Einbindung eines zweiten Spielercharakters folgte und auch die damit zusätzlich erforderliche Steuerung mit dem Kontroller (Mareen). Da die Spielregeln am echten Volleyballspiel angelehnt sind, musste eingeführt werden, dass der Ball nur drei Mal nacheinander von demselben Spieler berührt werden darf (Lars).

Bis zu diesem Meilenstein am 1. Juli war das Spiel soweit entwickelt, dass zwei Personen erfolgreich miteinander spielen konnten.

Zu diesem Zeitpunkt kamen die ersten Ideen auf, das Spiel durch spezielle Features aufzuwerten.
So entstand das Konzept den Spielspaß durch einen einwirkenden Wind zu erhöhen. Der Wind wurde im weiteren Verlauf umgesetzt. Außerdem wurde mit dem Gedanken gespielt, einen vier-Spieler-Modus einzubauen. Gegen diesen entschieden wir uns jedoch. Grund dafür war, dass für vier Spieler ein alltäglicher Bildschirm oder gar Laptop zu klein ist und das Spiel somit zu unübersichtlich wäre.

Vorlesungsfreie Zeit

Ebenso wie die vorherige Entwicklungsphase wurde auch diese mit einem Treffen und gemeinsamen Programmieren begonnen. Hier beschäftigten wir uns damit, einen eigenen Shader zu schreiben.

Um für eine spielerische und fröhliche Atmosphäre zu sorgen, wurde das Spiel mit einer eigens komponierten Musik hinterlegt (Mareen).

Die nun folgende räumliche Distanz sorgte dafür, dass weitere Aufgabenpakete verteilt wurden. Trotzdem blieb jeder auf dem aktuellen Stand, was bereits erledigt wurde und wo Probleme auftraten.

Da die grundlegenden Spielhandlungen bereits fertig waren, erfolgte die Erweiterung des Spiels mittels Animationen, welche als Feedback dienten. Zusätzlich wurde ein Partikelgenerator implementiert, welcher das am Feldrand des Gewinners erscheinende Feuerwerk erzeugt.
Des Weiteren wurde dieser verwendet, um das Spiel durch zufällig erscheinende VelocityDrops und SizeDrops unterhaltsamer zu gestalten. Darüber hinaus wurden die „Winner“-, „Loser“- und „Matchball“- Banner erstellt und in das Spiel integriert. Um ein schönes Spielende zu haben, fand die Implementierung der Kameradrehung am Matchende statt. (Mareen)

Es erfolgte die Implementierung des Windes, der Auswirkungen auf die Ballflugbahn hat und die der dazugehörigen Anzeigefahnen.
Das bereits vorhandene, aber sehr einfache Menü wurde durch weitere Einstellmöglichkeiten und deren Speicherung für das folgende Match erweitert. Dies wurde mit Hilfe von XML-Dateien gelöst. Darüber hinaus wurde bedacht, dass sich das Spielfenster an die jeweilige Bildschirmauflösung anpasst. (Lars)

In diesem Zeitraum wurden die Texturen zu den Modellen erstellt. Dabei traten Probleme mit diesen auf. In Folge dessen mussten alle Modelle überarbeitet und teilweise sogar neu erstellt werden, bevor diese mit ihren Texturen in das Spiel geladen werden konnten.
Mit Hilfe eines Skydomes wurde ein 3D gerechter Hintergrund für das Spiel erstellt.
Das bereits vorhandene Menü wurde aufgewertet, indem die Geschichte und die Quellen mit Danksagung als Lauftext erscheinen. (Lena)

Erweiterungsmöglichkeiten

Zu Beginn des Projektes war aufgrund der Geschichte angedacht, ein mehrteiliges Spiel zu gestalten. Dies soll in drei verschieden Karten mit zugehörigen Charakteren äußern. Während der Entwicklung beschlossen wir jedoch, zunächst nur eine der Karten, diese aber detailliert auszuarbeiten und die anderen gegebenenfalls später zu entwickeln.

Für die zweite Karte war eine Wasserlandschaft angedacht, bei der Delfine gegeneinander antreten. Die dritte Karte soll eine Wiese darstellen, auf der Hummeln gegeneinander spielen.

Die drei Karten sollen sich nicht nur in der Optik unterscheiden, sondern auch in den Fähigkeiten der Charaktere. Delfine sind am schnellsten, Hummeln können am höchsten springen und Pinguine haben den größten Schlagradius.

Sind alle drei Karten implementiert, besteht die Möglichkeit, dass Spieler die Karten nacheinander freischalten müssen. Hierfür würde es einen Story Modus geben, bei dem sich Spieler beginnend bei der Eislandschaft durcharbeiten müssen. Erst nach erfolgreichem Beenden einer Karte ist diese freigeschaltet und kann im Match Modus jederzeit ausgewählt werden.

Fazit

Bevor die Entwicklung des Spiels begonnen hat, wurde innerhalb des Teams besprochen, was für uns ein erfolgreicher Abschluss des Projekts bedeutet. Dieser Erfolg sollte an drei verschiedenen Faktoren festgemacht werden.

Zwei davon wurden mit Hilfe außenstehenden Personen verifiziert und validiert.

Ein Faktor hiervon ist die Ästhetik des Spiels. Es war uns wichtig, ein optisch ansprechendes Spiel zu haben, so dass der Spaßfaktor nicht durch schlechte Optik beeinträchtigt wird.

Der zweite wichtige Punkt war, ein abgeschlossenes und in sich stimmiges Spiel. Der Spieler sollte von Anfang an intuitiv durch das Spiel gelangen. Verzögerungen und unnötige Wartezeiten wurden vermieden.

Für diese zwei Kriterien hat jeder von uns das Spiel regelmäßig in seinem Umfeld bewerten lassen und wir arbeiteten Verbesserungsvorschläge ein.

Das dritte Erfolgskriterium war, dass wir als Entwickler auf das Endergebnis stolz sind. Es ist uns gelungen, ein komplettes, spielbares Spiel mit verschiedenen Features und Feedbacks zu entwickeln. In Anbetracht der Tatsache, dass solch ein Projekt für uns alle eine komplett neue Herausforderung dargestellt hat, sind wir zufrieden, das vorliegende Ergebnis innerhalb der vorgegebenen Zeitspanne gemeinsam entwickelt zu haben.

Neben einigen neuen Programmen, wie git, Monogame und Blender haben wir auch gelernt, was es bedeutet, mit mehreren Leuten zusammen zu programmieren. Dabei stellten wir fest, dass es wichtig ist, uns gegenseitig auf dem aktuellsten Stand zu halten und sich gegenseitig zu unterstützen, da nicht jeder alle Details des entwickelten Programms kennen kann.