CUSANUS – GYMNASIUM ERKELENZ

SpaceNET

-

Exemplarische Entwicklung eines Browserspieles

Alexander Lüpges

Jugend Forscht 2016

Kurzfassung

Immer mehr Menschen widmen Computerspielen einen beachtlichen Teil ihrer Freizeit - Zum Teil nur als Zeitvertreib.
Warum soll man diese Zeit nicht dazu nutzen grundsätzliche Informationen zu vermitteln?

In dieser Arbeit werde ich exemplarisch das Konzept eines Spieles planen und dies dann in die Tat umsetzen. Hauptaspekte sind **Kompatibilität**, **Erweiterbarkeit**, **Offenheit** sowie das **Vermitteln von Lerninhalte**.
Dem Spieler sollen die Grundlagen der Programmierung von künstlichen Intelligenzen vermittelt werden. Dazu kann er eine Weltraumbasis kontrollieren, Drohnen-Raumschiffe programmieren und ausschicken können.

Inhalt

1. Die Idee

Heutzutage spielen viele Menschen Spiele, ohne jedoch nur annähernd zu verstehen, was im Hintergrund für ihr Vergnügen, bzw. ihre Frustration sorgt.
Da ich Künstliche Intelligenzen schon immer spannend fand, habe ich dieses Gebiet heraus gepickt. Und mit was kann man Spielern besser zeigen, wie ein Spiel funktioniert, als mit einem Spiel?

2. Die Konzipierung

Der Spieler soll durch Programmieren Spielobjekte steuern können.
Wer schon mal einen Roboter (z.B. den Lego oder Fischertechnik Roboter) programmiert hat, weiß, dass Störeinflüsse für sehr viel Frustration sorgen können.
Folglicherweise vereinfacht ein „leerer" Raum die Prozedur. Und vereinfacht betrachtet ist der Weltraum leer.

Nun zu den programmierbaren Spielobjekten: Sie sollen steuerbar und verwundbar sein, eine begrenzte Lebenszeit (z.B. durch Treibstoff) und eine Aufgabe haben.
Praktisch eine Beschreibung für Raketen und Satelliten. Im Spiel werden sie **Drohnen** genannt.

Viele Spiele haben eine Geschichte (engl. story), mit der der Spieler gefesselt werden soll. Da SpaceNET ein Echtzeit-Strategiespiel sein soll, ist diese nicht notwendig und auch nur sehr schwer machbar.

2.1 Spielwelt

Das Spiel findet, wie bereits festgelegt, im Weltraum statt. Jetzt gibt es aber noch verschiedene Möglichkeiten um Spielwelten zu implementieren:

2.1.1 Open World oder linear

Grundlegend gibt es **Open World** Spiele und **lineare** Spiele.

Bei Open World Spielen ist der Spieler in der Lage (fast) jeden Ort der Welt zu jedem beliebigen Zeitpunkt zu besuchen. Sie kommen auch ohne Geschichte aus.
(Das wohl bekannteste Spiel ist das Klötzchenbauspiel *Minecraft*)

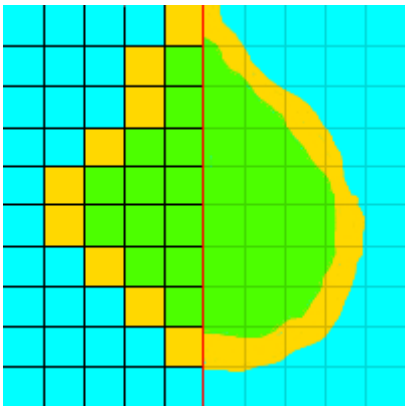
Lineare Spiele haben eine lineare Geschichte, d.h. der Spieler hat keine oder kaum Handlungsspielraum. Sie sind mit dem Aristotelischen Drama zu vergleichen.

Mittlerweile gibt es viele lineare Spiele (z.B. FINAL FANTASIE ® XIII, The Last of Us, The Walking Dead, Mirrors Edge), wobei im Jahr 2015 keine „Großen" erschienen sind (Quelle: Steam).
Pure Open World Spiele gibt es dagegen kaum, da praktisch alle Kombinationen aus den beiden Möglichkeiten sind. So sind zum Beispiel Regionen der Spielwelt erst nach einem gewissen Punkt in der Geschichte zu erreichen, usw.
(Neuerscheinungen an großen Open World Spielen 2015: The *Witcher 3*, *Fallout 4* und *Just Cause 3* (welches ebenso das Sandbox Prinzip beinhaltet) (Quelle: Steam))

Da SpaceNET praktisch keine Geschichte zu erzählen hat, ist eine offene Welt die bessere Lösung.

2.1.2 Genauigkeit der Spielwelt

Der nächste Aspekt ist die Genauigkeit eines Spieles. Reicht es aus die Spielwelt in Kacheln zu unterteilen oder braucht es genaue Positionsangaben?



Kacheln(diskret) oder kontinuierlich?

Das wohl bekannteste Strategie-Browserspiel mit Kachel-Welten dürfte *Travian* sein.

Travian (Weltkarte) | Rechte: Travian Games GmbH

Da die Drohnen in SpaceNET sich in Echtzeit bewegen sollen, muss die Welt kontinuierlich sein.
(Zumindest bis die Grenzen des Computers erreicht sind)

2.2 Basis

In den meisten Strategiespielen hat der Spieler eine Heimat. So auch in SpaceNET.
Die Basis befindet sich an einem festen Ort in der Spielwelt und erlaubt dem Spieler die Umgebung zu sehen.
Ebenso hat er die Möglichkeit hier neue Drohnen zu produzieren und zu starten.

2.3 Drohnen

Sie sollen vom Spieler komplett konfigurierbar sein, einen Antrieb haben, Schaden nehmen und austeilen können, Sachen aufsammeln und einfache Daten der Umgebung kennen.

2.4 Bauteile/Komponenten der Drohnen

Drohnen sollen sich aus folgenden Bauteilen zusammensetzten lassen:
Chassis: Stellt Platz für weitere Bauteile zur Verfügung.
Antrieb: Legt den Schub fest.
Waffen: Legt die Art der Bewaffnung und die Stärke fest.
Kameras: Sensoren, die die Umwelt des Schiffes beobachten (Farbe oder Energie) und z.B. zum anvisieren eines Zieles benutzt werden.
Kargo: Steigert die Anzahl an Ressourcen die mitgenommen werden können.
Batterien: Erlaubt es eine gewisse Anzahl an Energie zu speichern.
Generatoren: Generieren Energie aus Treibstoff
Tanks: Speichern Treibstoff
Wegpunkt-Assistenten: Speichern Wegpunkte und geben der KI Richtung & Entfernung an
TV-Stationen: Erlauben es dem Spieler die Umgebung zu Beobachten.
Computer: Legt die Art des Programmes (z.B. NodeAI) und dessen Eigenschaften fest
Schilde: Schützen vor Schäden

Jedes Bauteil kostet außerdem Ressourcen, Platz und hat eine Bauzeit.

Verschiedene Stufen von Bauteilen sollen als Anreiz dienen, um Drohnen zu bauen, die Ressourcen anschaffen können.
Die Komponenten für Drohnen sollen ebenso frei durch den Serveradministrator konfigurierbar sein.

2.5 Ressourcen

Je „stärker“ die Drohne, desto teurer ist das Bauen. Um an Ressourcen zu gelangen, kann man (bzw. die eigenen Drohnen) Ressourcen einsammeln (z.B. von Asteroiden oder zerstörten Drohnen) oder warten, da die Basis automatisch Ressourcen generiert (aber sehr langsam).
Und um an zerstörte Drohnen zu gelangen müssen diese ja erst einmal zerstört werden.

2.6 Weitere Spielobjekte

Um die Spielwelt zu bevölkern befinden sich diverse Objekte in dieser.
Da wären erstmals:

Sterne: Große Objekte die andere Objekte in der Nähe verbrennen und anziehen.
Asteroiden: Einsammelbare Ressourcenpakete

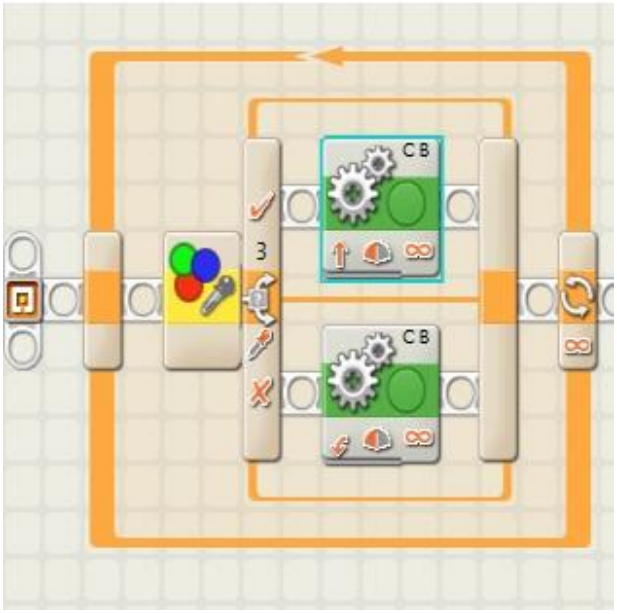
2.7 Erweiterbarkeit

Durch die Offenheit und Aufbau des Spiels soll es möglich sein, einfach neue Features hinzuzufügen.
(z.B. Planeten oder vom Computer gesteuerte Basen)

2.9 Programmierung durch den Spieler

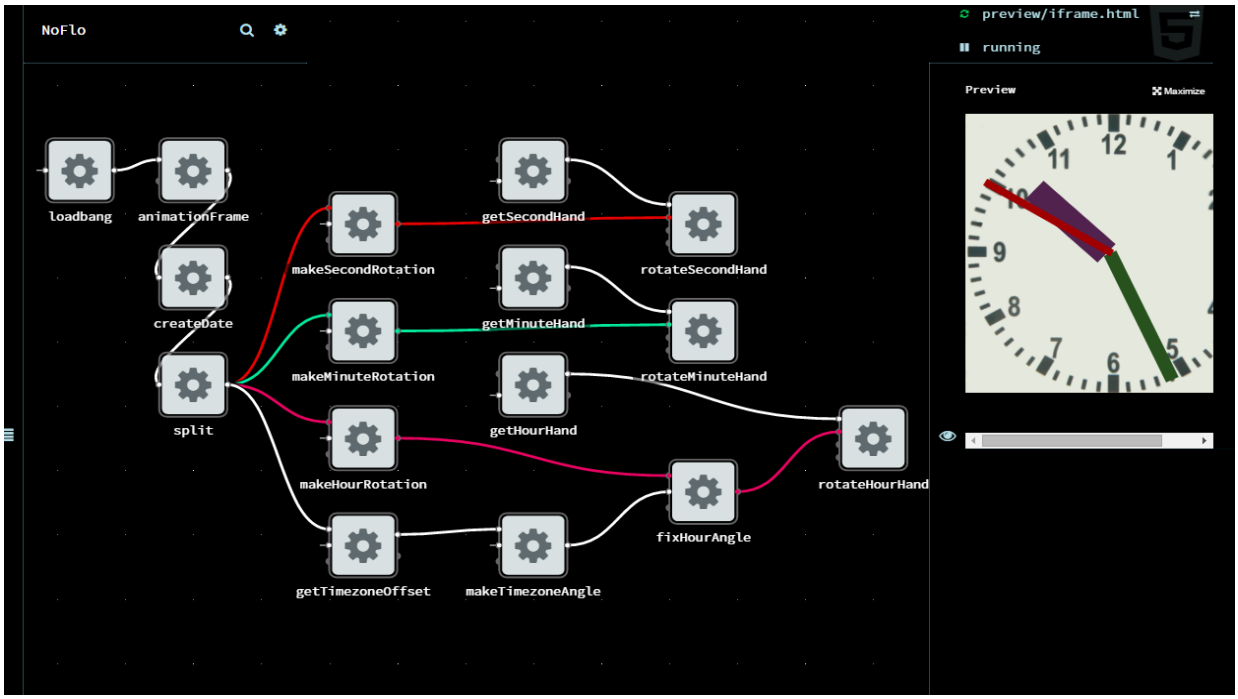
Da der Spieler in der Lage sein soll, seine Drohnen zu programmiere, muss ein Weg her, um dies zu tun. Häufig genutzte Wege wären hier Skripte (wie z.B. Lua) oder graphische Programmierungsoberflächen wie LabVIEW (wie es zum Beispiel Lego mit dem Lego® Mindstorms® tut).

LUA in Minecraft (OpenComputers)
Rechte: Florian "Sangar" Nücke



Graphisches Programmieren - Lego Mindstorms NXT
Rechte: LEGO A/S

Labview (worauf Lego baut) setzt auf das Programmierparadigma des **flow-based programming**.
Eine Webapplikation die FBP nutzt, wäre z.B. *NoFlo*.



NoFlo Demo einer Uhr
Rechte: noflojs.org

Da das graphische Programmieren für Neulinge einfacher sein dürfte, habe ich mich dafür entschieden. (Durch die Offenheit des Servers ist es allerdings kein Problem weitere Möglichkeiten hinzuzufügen).

3 Die Umsetzung des Konzeptes und Entwicklung des Spieles

Im nächste Schritt wäre zu überlegen, was auf das Spiel zukommt.

- Große Anzahl an Spielobjekten die möglichst in Echtzeit berechnet werden wollen
- Große Anzahl an Eigenschaften(z.B. Inventar oder Lebenspunkte)
- Viele Spielobjekte teilen sich Eigenschaften
- Spielobjekte (und deren Eigenschaften) werden zur Laufzeit instanziiert

Der erste (und wohl älteste) Ansatz ein Spiel zu entwickeln, wäre durch Vererbung.
Die Unreal Engine nutzt zum Beispiel diesen Ansatz. Das Problem ist nur: Alles muss implementiert und definiert sein (z.B. eine Drohne, die keine Waffe hat, besitzt trotzdem die Methoden, Funktionen und Variablen um z.B. eine Waffe zu feuern). Dies führt sehr schnell zu Speicher-Overhead.
Und wenn man hier „mal eben“ etwas ändern möchte, sucht man extrem viel.

3.0.1 Entity Component Systems (ECS)

In den letzten Jahren hat dagegen eine andere Architektur Bekanntheit erlangt. Sogenannte **Entity Component Systems**.
Hier verarbeiten Systeme (z.B. das Bewegungssystem) alle Entities mit von ihnen gewünschten Components (Komponente).
Die Entities(Spielobjekte) bestehen hierbei nur noch als eine Art Sammlung von Komponenten.

In den Komponenten werden nur Daten gespeichert.
Die Verarbeitung dieser Daten findet dann in den Systemen statt.

Nun lassen sich ganz einfach neue Entities/Spielobjekte erstellen.
Z.B.: Ein Spielobjekt mit Leben, Position, Bewegung, Beschleunigung und Masse
Jetzt bearbeitet das Physiksystem die Bewegung abhängig von der Beschleunigung und die Position abhängig von der Bewegung, usw.

3.0.2 Events

Um das Ganze noch modularer zu gestalten, ruft das Waffensystem nicht etwa eine Funktion im Lebenssystem auf oder verändert direkt die Lebenspunkte, sondern leitet ein *Schaden-genommen-Event* ein.
Das Lebenssystem wird darüber informiert, da es sich für *Schaden-genommen-Events* interessiert und verändert nun die Lebenspunkte.

Wenn man jetzt z.B. Animationen einfügen möchte, muss man nicht wie beim herkömmlichen Ansatz Programmcode einbauen, der erst die Lebenspunkte abzieht und dann eine Animation einleitet, sondern man muss nur ein System registrieren, welches sich für *Schaden-genommen-Events* interessiert.

3.1 Die Programmierung des Servers

> Da der Quellcode sämtliche Masse sprengen würde, werde ich jeweils nur Auszüge präsentieren.
> Der gesamte Quellcode ist unter http://spacenet.askarian.net/source/ einzusehen.

Nachdem jetzt feststeht, dass ein ECS (Entity Component System) mit Events genutzt wird, geht es an die Umsetzung:

Um das Rad nicht komplett neu zu erfinden, wird **Artemis-odb** (eine populäre Weiterentwicklung des Artemis ECS genutzt).
Ebenso der EventBus aus Googles **Guava** Projekt.

Da ich **Java** als gut zu lesende Sprache empfinde (vor allem für Programmieranfänger) und sie eine einigermaßen gute Performance bietet, wird der Server in **Java** programmiert sein.

Da der Server mehrere APIs anbietet, spielt es keine große Rolle, wie der Client aufgebaut ist.
Am einfachsten für den Spieler ist allerdings der **Browser**, deshalb wird der Client eine Webseite sein, die über **JavaScript** (**WebSockets**) mit dem Server kommuniziert.

3.1.1 Hauptprogrammschleife

Dieser Programmteil versucht den Rest des Spiels möglichst regelmäßig (hier 20 mal die Sekunde) aufzurufen:

```
final long optimaleTickZeit =  1000000000 / 20; //20 mal pro Sekunde
long letzteLoopZeit = System.nanoTime();

while (true){ //Hauptschleife des Spieles
    long jetzt = System.nanoTime();
    long updateZeit = jetzt-letzteLoopZeit;

    letzteLoopZeit = jetzt;

    float delta = updateZeit / (float)optimaleTickZeit;

    artemisWorld.setDelta(delta);
    artemisWorld.process();

    Thread.sleep( (letzteLoopZeit-System.nanoTime() + optimaleTickZeit)/1000000
);
}
```

Nun ruft das ECS Artemis die verschiedenen Systeme auf.

3.1.2 PhysicsIntegrationSystem (Die Bewegung)

Zum Beispiel das System, was sich um die Bewegung kümmert (PhysicsIntegrationSystem.java)

```
@Override
protected void process(int entity) {
    Acceleration acceleration = accelerationComponentMapper.get(entity);
    Velocity velocity = velocityComponentMapper.get(entity);
    Position position = positionComponentMapper.get(entity);
    Mass mass = massComponentMapper.getSafe(entity);
    ForceAccumulator forceAccumulator = forceAccumulatorComponentMapper.get
(entity);
    //Unendlich große Masse => return
    if (mass == null || mass.inverseMass <= 0.0d)
        return;
    //lineare Position updaten
    position.pos.addScaledSelf(velocity.vector, duration);

    //Über update informieren
    if (!velocity.vector.isZero()) {
        regionReference.getRegion().postEvent(new UpdatePosEvent(entity));
    }

    //Beschleunigung(Velocity) von Kraft(Force)
    Vector2d resAcc = acceleration.acc.clone();
    resAcc.addScaledSelf(forceAccumulator.res, mass.inverseMass);

    Vector2d v = velocity.vector.clone();
    //Beschleunigung(Velocity) updaten
    velocity.vector.addScaledSelf(resAcc, duration);

    //Dragging
    velocity.vector.multSelf(Math.pow(damping, duration));

    //Kraft zurücksetzten
    forceAccumulator.res.clear();
}
```

Hier sieht man zum einen, dass ein System die „Arbeit“ erledigt. Und zum anderen, dass ein *Event* (UpdatePosEvent) andere Systeme informiert.

3.1.3 ChunkSystem

Ein System, was sich für dieses Event interessiert wäre das ChunkSystem (ChunkSystem.java).
Um für bessere Performance zu sorgen, ist die Welt in Chunks eingeteilt. Jeder Chunk ist ein Quadrat von z.B. 100x100 und hält eine Referenz zu Objekten die ihm zugehören, d.h. die sich innerhalb seines Bereiches befinden.
Jeder Chunk ist durchgehend mit einer Ganzzahl für je seine X und Y Position beginnend bei 0 nummeriert.

```
final Table<Integer, Integer, List<Integer>> chunkTable;

@Subscribe
public void onUpdatePosition(UpdatePosEvent updatePosEvent) {
    int entity = updatePosEvent.getEntity();
    Position pos = positionComponentMapper.get(entity);
    ChunkReference chunk = chunkReferenceComponentMapper.get(entity);
    int x = (int) pos.pos.getX() / chunkSize, y = (int) pos.pos.getY() /
chunkSize;

    if (chunk.x != x || chunk.y != y) { //Chunk ändern
        List<Integer> entityList
                = chunkTable.get(chunk.x, chunk.y);
        if (entityList != null) { //Aus Liste entfernen
            entityList.remove(Integer.valueOf(entity));
            if (entityList.isEmpty()) //Da die Liste jetzt leer ist,
entfernen
                chunkTable.remove(chunk.x, chunk.y);
        }
        //Chunk updaten
        chunk.x = x;
        chunk.y = y;

        //In die Liste eintragen
        entityList = chunkTable.get(x, y);
        if (entityList == null) {
            entityList = Lists.newLinkedList();
            chunkTable.put(x, y, entityList);

        }
        entityList.add(Integer.valueOf(entity));
    }
}
```

Durch die Referenz lassen sich z.B. viel schneller Objekte in der Nähe suchen:

```
public List<Integer> getNearbyEntities(int posX, int posY, int radius) {
    List<Integer> entityList = Lists.newLinkedList(), chunkList;
    int radiusSQ = radius * radius;
    Position pos;
    int radiusChunks = (int) Math.ceil(radius / chunkSize), startChunkX =
(int) Math.floor(posX / chunkSize), startChunkY = (int) Math.floor(posY /
chunkSize);
    for (int cX = startChunkX - radiusChunks; cX <= startChunkX +
radiusChunks; ++cX) {
        for (int cY = startChunkY - radiusChunks; cY <= startChunkY +
radiusChunks; ++cY) {
            chunkList = chunkTable.get(cX, cY);

            if (chunkList == null || chunkList.isEmpty()) //Keine entities
in dem chunk
                continue;
            for (Integer entityID : chunkList) {
                pos = positionComponentMapper.getSafe(entityID);
                if (pos == null) continue;
                //Ist Entity im Radius?
                if (pos.pos.distanceSQR(posX, posY) < radiusSQ) {
                    //Add to return liste
                    entityList.add(entityID);
                }
            }
        }
    }
    return entityList;
}
```

Ohne eine Gruppierung (nach Chunks) müssten alle Spielobjekte der Welt verglichen werden. Und dies kann sehr lange dauern.
Um dies zu optimieren, sind die Spielobjekte nach Chunks gruppiert. Somit müssen nur noch die Spielobjekte in Chunks die in Frage kommen betrachtet werden.

Ähnlich verfährt es sich mit den anderen Systemen und Komponenten, die aber den Rahmen sprengen würden. (Bei Interesse bitte http://spacenet.askarian.net/source/ besuchen).

3.2 Server-Client Kommunikation

Der Server sendet Updates (über Events) an den Client.
Im Falle des Browser-HTML-Clients werden diese über eine WebSocket Verbindung als JSON-Text gesendet.
Um das Rad nicht neu zu erfinden, wird hier auf **Java WebSockets** zurückgegriffen. Eine Implementierung eines WebSockets Servers.

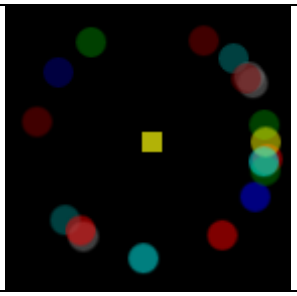
3.3 Browser-Client | Die Karte

Browser sind seit einiger Zeit in der Lage mithilfe von JavaScript auf **<canvas>** HTML Elementen zu zeichnen. Neuere Browser (seit ca. März 2011) können aber auch direkt die OpenGL Schnittstelle über WebGL ansprechen, wodurch die Performance deutlich gesteigert wurde.

Um auch hier das Rad nicht neu zu erfinden, wird **Pixi.js** genutzt. Eine Open-Source 2D Graphik Bibliothek für den Webbrowser.

Durch eine schwarzes Hintergrundbild (das später durch Sterne im Hintergrund verschönert werden kann) wird die Umgebung dargestellt.

Mittig sieht der Spieler das Spielobjekt, von welchem er die Welt beobachtet. (Meist die eigene Basis)
Basen werden als Quadrat angezeigt, Drohnen als farbige Kreise, etc.



Ausschnitt der Karte. Hier einer Basis mit mehreren Drohnen

Durch das Klicken auf ein Spielobjekt werden Informationen über das Objekt angezeigt.
So erscheint zum Beispiel beim Klick auf eine Drohne ein Fenster, welches die wichtigsten Information anzeigt.

Drohnen Informationsfenster

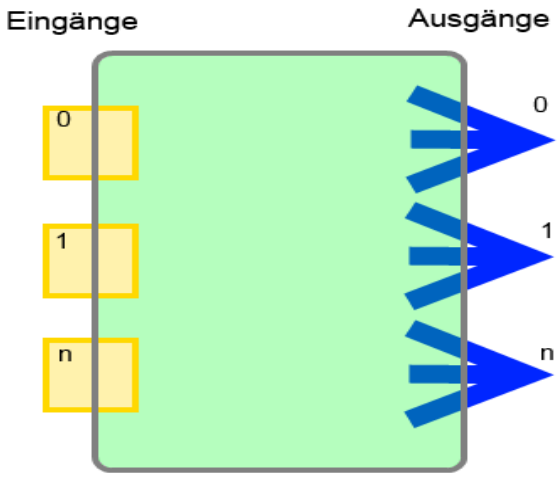
Ein Klick auf die eigene Basis dagegen öffnet ein Fenster in dem zum einen Informationen über den Spieler angezeigt werden und zum anderen die Einstellungen für das Produzieren neuer Drohnen.

Basis Steuerpult

3.4 Server | Programmierung durch den Spieler

Die erste zu implementierende Art die Drohnen zu Programmieren soll auf dem **flow-based-programming** Programmierparadigma aufbauen.

Es werden hierbei Komponenten (oder auch Prozesse) an ihren jeweiligen Ports über Verbindungen miteinander verbunden.
Der Name stammt daher, dass Informationen somit von Komponente zu Komponente fließen (engl. *to flow*).
Ein Synonym für flow-based-programming ist node-based-programming (engl. node=Knoten).
Daher stammt auch der Name dieser „Komponente“ des Spieles: **NodeAI** (Eine künstliche Intelligenz die auf Knoten basiert)



Aufbau einer Komponente

Informationen, die auf einem Port eingehen, werden intern zwischengespeichert.

Implementiert ist im Folgenden eine Komponente mit 3 Eingängen und einem Ausgang, die die Informationen die in den Eingängen 0 (erster Summand) und 1 (zweiter Summand) ankommen addiert und die Summe über den Ausgang 0 (Summe) weiterschickt.
Der 3. Eingang (Nummer 2) löscht die intern gespeicherten Werte für die Komponente.

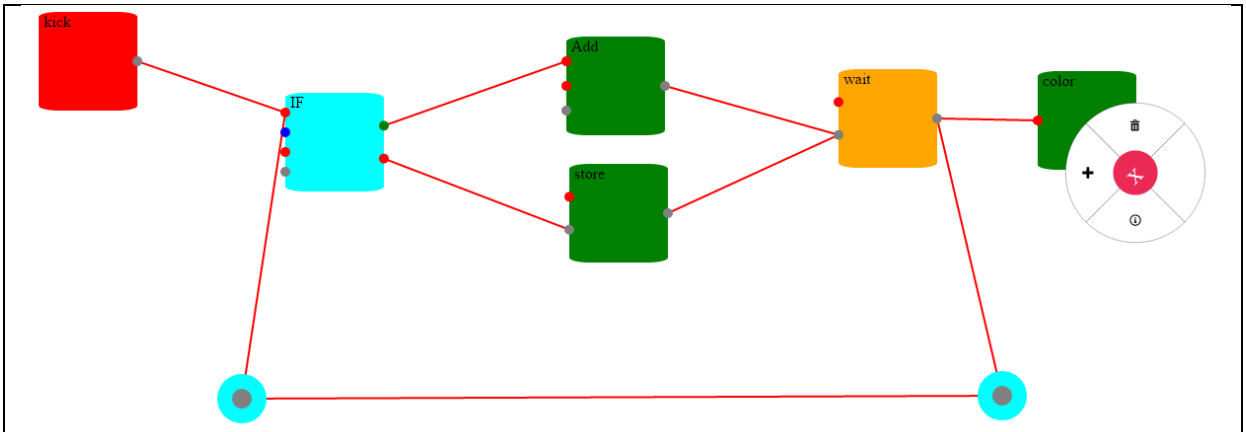
```
@Override
public void onInput(ProgramNode me, int port) {
    if (port == 2)
        me.clearPorts();
    else if (IP.isNumber(me.getPortIP(0)) && IP.isNumber(me.getPortIP(1)))
        me.finish(new Object[]{((Number) me.getPortData(0)).doubleValue() +
((Number) me.getPortData(1)).doubleValue()});
}
```

Die finish Methode erwartet ein Array von Objekten (Jedes Objekt entspricht der Information die auf einem Ausgang rausgeht) und sendet die Informationen an verbundene Komponenten.

3.5 Browser-Client | Programmierung durch den Spieler

Nun zu dem Visuellen: Ein flussbasiertes Programm lässt sich durch ein gerichtetes Petrinetz beschreiben. Und HTML unterstützt **SVG** (*Scalable Vector Graphics*), ein Grafikformat zur Beschreibung von zweidimensionaler Vektorgrafiken.

Mit etwas JavaScript lassen sich so SVG Grafiken erstellen, die ein NodeAI Programm repräsentieren.



NodeAI Programm welches die Farbe der Drohne regelmäßig ändert.
So lange der Wert (zu Anfang **1**) **unter 12** bleibt (IF), wird **1** hinzugefügt (add).
Ansonsten wird der Wert auf **0** gesetzt (store).
Dann werden **100** Ticks abgewartet (wait), bis die Farbe gesetzt wird(color).
Jetzt wird die Information über Strukturpunkte zurück zum IF geleitet.
Fettgedruckte Werte repräsentieren durch den Nutzer festgelegte Konstanten.

Rechts zu sehen ist ein Kontextmenü, in welchem der Nutzer die Node löschen, eine Neue hinzufügen, oder Informationen zur Node erhalten kann.

Ebenso lassen sich Nodes verschieben.
Verbindungen werden geknüpft, indem man zuerst auf den Eingangsknoten, dann den Ausgangsknoten klickt.

Eine Besonderheit stellen die runden Nodes dar. Sie leiten eingehende Informationen direkt weiter und dienen nur der Übersichtlichkeit. (Sie werden Strukturpunkte genannt.)

Das oben zu sehende Programm sieht in JSON Form wie folgt aus:


```
[{"oP":{"0":[[5,0]]},"iP":{},"x":10,"y":10,"id":0,"type":"kick"},{"oP":{"0":
[[6,0],[2,0]]},"iP":{"0":[100],"1":[[3,0],[4,0],[3,0],[4,0],[3,0],
[4,0]]},"x":814,"y":69,"id":1,"type":"wait"},{"oP":{},"iP":{"0":[[1,0],
[1,0],[1,0]]},"x":1023,"y":70,"id":2,"type":"color"},{"oP":{"0":
[[1,1]]},"iP":{"0":[[-1,"0"]],"1":[[5,1],[5,1],
[5,1]]},"x":548,"y":164,"id":3,"type":"store"},{"oP":{"0":[[1,1]]},"iP":
{"0":[[5,0],[5,0],[5,0]],"1":[[-1,"1"]],"2":
[]},"x":545,"y":35,"id":4,"type":"add"},{"oP":{"0":[[4,0]],"1":
[[3,1]]},"iP":{"0":[[0,0],[7,0],[0,0],[7,0],[0,0],[7,0]],"1":["<"],"2":
[12],"3":[]},"x":260,"y":92,"id":5,"type":"condition"},{"oP":{"0":
[[7,0]]},"iP":{"0":[[1,0],[1,0],
[1,0]]},"x":987,"y":399,"id":6,"type":"struct"},{"oP":{"0":[[5,0]]},"iP":
{"0":[[6,0],[6,0],[6,0]]},"x":216,"y":402,"id":7,"type":"struct"}]
```


3.6 Debugger | Programmieren

Um das Programmieren einfacher zu machen, existiert eine Debugging Funktion.
Während der Debugger läuft, sieht der Nutzer im Client den Wert den eine Verbindung transportiert bzw. den ein Port zwischengespeichert hat.
Deshalb hat der Client die Möglichkeit dem Server mitzuteilen, dass er ein Programm debuggen möchte.
Nun informiert der Server dem Client (über eine WebSocket Verbindung) über Veränderungen der Daten. Im Falle des Farbwechselprogramms wären dies die Zahlen 0 bis 12.

4 Probleme

Während des Programmieren und des Testens sind mehrere Probleme aufgetreten:

Performance der NodeAI: Da die NodeAI für jede Drohne relativ viel berechnen muss, steigt die benötigte Zeit in etwa linear mit der Anzahl der Drohnen.
Lösung: Die NodeAI in mehreren Threads laufen lassen (Multithreading)

Performance des Rendern im Browser: Ab einer gewissen Anzahl an Objekten hängt der Browser.
Lösung: Rendering optimieren sowie weitere Clients programmieren und anbieten

Datenaustausch via WebSockets: Je mehr Daten gesendet werden, desto größer ist die Verzögerung von Server zu Client und andersrum.
Lösung: Daten komprimieren (was mehr Arbeit für Browser und Server bedeutet)

Schlichter Client: Zumindest der Browser-Client sieht im Moment sehr schlicht aus. (Manche behaupten sogar, er sähe hässlich aus.)
Lösung: Die Webseiten (Übersicht, Spiel und Login) mit CSS und Bildern verschönern

Fehlendes Tutorial: Fast alle Tester bemängelten das Fehlen eines Tutorials.
Lösung: Ein Tutorial einbauen.

5 Zukunft

Was die meisten Tester bemängelten, war die fehlende Interaktion mit anderen Spielern.
Sei es Gruppenbildung, Handel, etc.
Ebenso ist ein Tutorial dringend notwendig (es sei denn, man erklärt immer Allen alles über Skype oder e-mail).

6 Fazit

Rückblickend lässt sich sagen, dass das Programmieren eines Spieles viel Arbeit, aber machbar ist.

Das Spiel ist mit allen neuen Browsern kompatibel. Mögliche Server müssen nur Java 8 installiert haben.
Es ist sehr gut und einfach erweiterbar. Um zum Beispiel einen neuen Client hinzuzufügen, ist nur ein System notwendig, was sich für bestimmte Events interessiert.
Weitere Spielobjekte(z.B. schwarze Löcher) lassen sich ebenso leicht hinzufügen.
Und neue Komponenten für Drohnen lassen sich sogar über Dateien direkt modifizieren und hinzufügen.
Die Offenheit ist durch die Offenlegung des Quellcodes und dem akzeptieren von Pull-Request gewährleistet.
Nur das Vermitteln von Lerninhalten müsste noch verbessert werden. Manche Tester haben sich sehr schnell hineingearbeitet und haben innerhalb kürzester Zeit Drohnen programmiert, die meisten benötigten aber eine längere Erklärung.

7 Quellen

Generelle Informationen

- AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors (Alex J. Champandard)
- Game Physics Engine Development: How to Build a Robust Commercial-Grade Physics Engine for your Game (Ian Millington)
- Nebenläufige und verteilte Systeme - Theorie und Praxis (Thomas Peschel-Findeisen)
- https://en.wikipedia.org/wiki/List_of_massively_multiplayer_online_real-time_strategy_games
- https://developer.mozilla.org/de/docs/Web/Guide/HTML/Canvas_Tutorial
- https://developer.mozilla.org/de/docs/Web/API/WebGL_API
- https://wiki.selfhtml.org/wiki/SVG
- https://github.com/pixijs/pixi.js/
- https://github.com/junkdog/artemis-odb
- https://github.com/google/guava
- https://github.com/noflo
- http://noflojs.org/
- http://www.heise.de/developer/artikel/Component-Based-Entity-Systems-in-Spielen-2262126.html
- http://www.gamedev.net/page/resources/_/technical/game-programming/understanding-component-entity-systems-r3013
- http://gameprogrammingpatterns.com/component.html
- http://www.jpaulmorrison.com/fbp/
- Flow-Based Programming, 2nd Edition: A New Approach to Application Development (http://dl.acm.org/citation.cfm?id=1859470)
- Stack Overflow (http://stackoverflow.com/) für die Hilfe bei spezifischen Fragen
- Screeps.com für die Inspiration (https://screeps.com/)

Spiele-Neuerscheinungen

- http://store.steampowered.com/tag/en/Linear/#p=0&tab=NewReleases
- http://store.steampowered.com/tag/en/OpenWorld/#p=0&tab=NewReleases

Abbildungen

- http://cil.li/mc/oc/01_computer_thumb.png (OpenComputers Bild)
- Alle Weiteren: Eigen. Rechte an den Programmen (falls unterschiedlich) angegeben

8 Danksagungen

Ein Dank gilt allen freiwilligen Testern von SpaceNET für Verbesserungsvorschläge und gefundene (und ausgenutzte) Fehler, Herrn Pretzel für das Hosten eines SpaceNET Servers der dauerhaft erreichbar ist (oder sein sollte) und lima-city für das Hosten meiner Webseite(n).

Ein besonderer Dank gilt Herrn Winkens für die Betreuung der Arbeit sowie meinen Eltern für die „Betreuung“ und das Korrekturlesen.