Projektbericht - Beschreibung der Architektur des Schlangenspiels

Bei der Abgabe handelt es sich um ein IntelliJ Projekt. Es beinhaltet die Server- und die Client Komponente. Über die Pakete und Namensgebung ist dies auch ersichtlich. Beim Start des Clients wird als erstes ein Name abgefragt. Die Eingabe ist zwingend und kann icht umgangen werden. Anschließend wird die IP-Adresse abgefragt. Ein Wegdrücken des Dialoges oder die Bestätigung führt zur Verwendung der Defaulteinstellung (127.0.0.1 an Port 8888) und somit den lokalen Rechner.

Das Schlangenspiel wurde als Client Server Anwendung mittels Sockets umgesetzt. Basis bildet Java in der Version 8. Der Client wurde als JavaFX Anwendung realisiert.

Grundlagend wurden fast alle Integer sind als AtomicInteger angelegt und alle Listen als SynchonizedList. Fast jede Methode wurde ebenfalls als synchronized umgesetzt. Im weiteren wurde zur Synchronisation der einzelnen Threads mit sleep, final und der Synchronisierung einzelner Objekte gearbeitet.

Die zwei statischen Klassen **Screen** und **Keys** beinhalten Konstanten, die die visuelle Ausgabe der Modellwelt festlegen (Screen) und einen einfachen und einheitlichen Zugriff auf die benötigten Textschlüssel ermöglichen (Keys).

Die Kommunikation erfolgt textuell. Informationen werden mit einem führenden Schlüsselwort übermittelt. Die Schlüsselwörter sind in **Keys** definiert. Für die Auflistungen der Futtereinheiten (**Feed**) und Schlangen (**Snakes**) existiert eine definierte Ausgabe unter Ausnutzung der Anzahl und eines festen Übertragungsformates. Beide Klassen stellen Methoden hierfür zur Verfügung (CreationString der Snake und des Feedprovider).

Serverseite

Die Serverkomponente beinhaltet die vier Hauptklassen MainClass, GameServer, Game und Player. MainClass bildet hierbei lediglich den Einstieg in die Anwendung und enthält die Main-Methode.

Diese startet den **GameServer** in einem eigenen Thread. Seine einzige Aufgabe ist der Start von **Game** und die Aufnahme neuer **Player** (Spieler). Hierzu wird nach jeder Verbindung mit

einem neuen Client Rechner eine Instanz vom Typ Player erzeugt, gestartet und dem Spiel (Game) hinzugefügt. Hierzu enthält GameServer eine Instanz der Klasse Game.

Die Klasse **Game** läuft in einem eigenen Thread. Sie geht in regelmäßigen Abständen alle **Player** (Spieler) durch, positioniert deren **Snakes** (Schlangen) neu und berechnet die Futteraufnahme. Anschließend aktualisiert er die Eigenschaften der Schlangen und sendet zu jedem Teilnehmer den aktuellen Status. Hierfür wird u.a. die Methode goAndFeet der Schlange aufgerufen. Scheidet eine Schlange aus, so wird sie auf ihren Urzustand zurückgesetzt (Größe und Punkte) und dem Spiel am Anfangspunkt jeder Schlange neu hinzugefügt. Über die Konstante SPEED (Game) kann die Geschwindigkeit reguliert werden, über TURN_SPEED (Snake) die Drehgeschwindigkeit der Schlange. Bei sind aktuell eher etwas langsamer eingestellt.

Player kapselt einen Spieler. Jeder Spieler ist ein Client. Hierzu wird der Name, dessen Socket sowie **Snake** (Schlange) gehalten. Jeder Spieler läuft in einem eigenen Thread. Hauptaufgabe des Threads ist die Auswertung, ob ein Rotationsbefehl übermittelt wurde und wirkt insofern quasi als eine Art Puffer. Ist dies der Fall, so wird es vermerkt. Zudem stellt die Klasse Methoden zur Verfügung, um mit dem Client selbst zu kommunizieren.

Die Klasse **Snake** (Schlange) enthält hauptsächlich Eigenschaften der Schlange. Die wichtigsten sind: Position, Richtung, Anzahl der Segmente und deren Auflistung sowie den Punktestand. Die Positions und Richtungsangaben erfolgen als Vektoren, die Segmente selbst stellen ebenfalls Vektoren dar. Innerhalb der Schlange erfolgen in der Methode goAndFeed die Wichtigsten Methoden zur Neuberechnung der Schlange (Punkte, Segmente, Position).

Die Klassen **Feed** (Futter) stellt Futtereinheiten dar, die vom **Feedprovider** verwaltet werden. Der Feedprovider hält die einzelnen Futtereinheiten und ersetzt bei Bedarf “gegessenes” Futter, so dass immer die Mindestanzahl an Futtereinheiten vorhanden sind. Er läuft in einem eigenen Thread. Bei Bedarf wird neues Futter an einem zufälligen Ort mit zufälligen Wert generiert. Feed enthält nur die Informationen über sich selbst (Position und Wert). Daneben enthält er in der Methode reachable Logik zur Feststellung, ob sich ein gegebener Punkt in Reichweite befindet. Die Auslagerung dieser Logik ermöglicht eine einfache spätere Anpassung. Im Moment ist sie so eingestellt, dass Futter u.U. auch bei

großer Annäherung aufgenommen wird. Ein Algorithmus, der vektorbasiert über den Abstand rechnet, bietet eine Alternative, wurde aber durch den aktuellen ersetzt.

Der **Feedprovider** enthält eine List mit Futter sowie die Logik zur Entfernung und Erzeugung von Futter. Auch er wird innerhalb vom Game als final instanziiert. Defaultmäßig soll er beim Erstellen die Mindestanzahl an Futter generieren und einer Unterschreitung durch Erzeugung neuen Futters entgegenwirken.

Clientseite

Die wichtigsten Klassen der Clientseite sind **MySnake** und **ServerConnection**.

ServerConnection stellt eine Verbindung zu einem Spieleserver dar und kapselt die dazu notwendige Logik. Er empfängt Daten vom Server und reicht diese mit einem Callback zur Anwendung weiter. Hierfür nutzt er die funktionale Schnittstelle **ICallback**. Gleichzeitig bietet er Methoden an, um Nachrichten zum Server zu senden.

Bei **ICallback** handelt es sich um ein Interface mit nur einer Methode callback, die einen String weiterreicht.

Bei **MySnake** handelt es sich um eine JavaFX Anwendung. Sie zeigt eine Oberfläche an und erzeugt die visuelle Entsprechung der gelieferten Daten. Sie erzeugt eine Instanz von ServerConnection, die in einem eigenen Thread läuft. Die Aktualisierung wird durch den Aufruf der Callback Methode vorangetrieben. Um ohne Konflikte Elemente des GUI-Threads zu manipuliere, greift der Hintergrundthread mittels Plattform.runLater() hierauf zu.

Probleme und Kritik

Aktuell läuft das Spiel wie erwartet. Auch bei einem Test mit 8 Schlangen innerhalb von IntelliJ lief es problemlos. Es fällt auf, dass Futterstücke manchmal nicht aufgenommen werden, manchmal erst später gelöscht werden. Ebenso wird die Punktzahl manchmal verzögert aktualisiert. Die genaue Ursache konnte ich noch immer nicht feststellen und dagegen vorgehen.

Dieser Fehler trat zu Anfang doch wesentlich massiver auf. Eine Ersetzung der Processing Vektoren durch die Vektoren- und Matrizenklasse der Hausaufgabe in CG in diesem

Semester, sowie eine komplette Überarbeitung der Berechnung (Verlegung in die Snake-Klasse) brachte hier viel. Trotzdem ist er noch nicht behoben.

Was die Synchronisation der Methoden und die Verwendung von AtomicInteger angeht, so würde ich bei einer Überarbeitung nochmals genauer schauen, inwieweit dies nötig ist. Hier ist noch Potential für Verbesserungen. Das gleiche gilt für die Architektur der Threads. Es stellt sich die Frage, ob es wirklich sinnvoll ist, jeden Spieler einen eigenen und dazu noch unabhängigen Thread zu geben. Stichwort könnte hier ThreadPool sein.