Änderungen:

Renderthread

Jobsystem

Koordinierung

* BlocklibExecutor
  + Implementiert BlocklibExecutorService
  + Delegiert die Ausführung von Tasks and zwei ScheduledThreadpoolExecutors abhängig von der TaskPriority
  + Delegiert die Verpackung von Tasks, sodass diese ScheduledCompletableFutures sind
  + Zählt die Anzahl der aufgegebenen Tasks anhand ihrer Klasse für Statistik/Debugging-Zwecke
  + NamedThreadFactory
    - Erzeugt Threads mit bestimmten Namesn Präfixen, um das Debugging zu vereinfachen
  + TaskPriority
    - Enum, das die Priorität eines Tasks definiert, aktuell sind nur die Prioritäten NORMAL und BACKGROUND vorhanden. Sobald der Executorservice so umgebaut ist, dass er auch Prioritäten unterstützt, können mehr Prioritäten genutzt werden.

Verkettung von Tasks

* CompletableFutureWrapper
  + Erzeugt CompletingCallables aus Callables und Runnables
  + Nutzt DelegatingCompletabelFutures, da die darunterliegenden ExecutorServices eigene Futures erzeugen, die für die Interaktion genutzt werden müssen.
* CompletingCallable
  + Erhält ein CompletableFutureObjekt und ein Callable
  + Führt das Callable aus und ruft entsprechend complete oder completeExe im CompletableFuture auf.
* RunnableCallable
  + Verpackt ein Runnable als Callable<Void> oder mit einem Rückgabewert
* DelegatingCompletableFuture
  + CompletableFuture, das ein inneres Future nutzt und Methodenaufrufe an dieses weitergibt.
  + Besitzt eine setMethode, da die Erzeugung des inneren Future-Objekts später geschehen muss.

Konsolidierung:

* Alle Stellen, die bisher schon Threads genutzt haben, werden so angepasst, dass sie nun einen BlocklibExecutorService nutzen
* Übergabe geschieht mittels Konstruktor
* Context wird der Service mittels Konstruktor übergeben und gibt ihn an Manager weiter
* EventManager
  + benachrichtigt nun mittels BlocklibExecutorService sequentiell. Muss aktuell sequentiell sein, da die Eventlistener nicht darauf ausgelegt sind, dass Events parallel ausgegeben werden.
* Fluid
  + Nutzt BlocklibExecutor um in Regelmäßigen Abständen den neuen Zustand zu berechnen -> Keine Notwendigkeit für Thread.sleep mehr. Stattdessen scheduleAtFixedRate(…)
* ChunkStorage und abgeleitete Klassen
  + loadChunk. saveChunk, doLoadChunk und meshChunk nutzen Executor mit TaskPriority.BACKGROUND, damit die Berechnung des nächsten Bilds nicht aufgehalten wird.
* Nicht Konsolidiert:
  + Aus Zeitgründen:
    - NioClientNetworkLayer
    - NioServerNetworkLayer
  + Weil Dependency Injection kompliziert ist
    - ConnectionInfo