Inhaltsverzeichnis

Abbildungsverzeichnis

1 Beurteilung des Standes der Implementierung anhand der Kriterien

Im Folgenden werden die Kriterien aus der Pflichtenheft-Phase hinsichtlich des Fortschritts des Projekts beurteilt. Kriterien, die erreicht wurden, werden dabei mit einem Haken gekennzeichnet. Kriterien, deren Erreichen bis zur Endabnahme noch zu erwarten sind, werden mit einem leeren Kästchen gekennzeichnet. Kriterien, die nicht mehr erfüllt werden, werden mit einem Kreuz markiert.

- ☑ Beispiel für ein erreichtes Kriterium
- ☒ Beispiel für Kriterium, das nicht mehr erreicht werden wird
- ☐ Beispiel für Kriterium, das noch geplant, aber noch nicht abgeschlossen ist

1.1 Musskriterien

- ☑ $\langle RM1 \rangle$ Die Mod muss im Kreativmodus von Minecraft verwendet werden können.
- ☑ $\langle RM2 \rangle$ Die Mod muss den Bau eines Lerncomputers ermöglichen, anhanddessen der Nutzer Rechnerarchitektur lernen kann.
- ☑ $\langle RM3 \rangle$ Die Mod muss eine MIMA-Architektur nach Tamim Asfour simulieren können.
- ☑ $\langle RM4 \rangle$ Die Mod muss vom Nutzer eingegebenen MIMA-Assembler-Code ausführen können.
- ☑ $\langle RM5 \rangle$ Es muss ein Block-Typ für Busse, die ALU, den Taktgeber, den Hauptspeicher, das Steuerwerk, die Register und den Programmeditor existieren. Die Blöcke müssen die Standard-Minecraft-Blockfunktionen wie Platzieren, Zerstören, Explodieren und das Hinterlassen von Drops unterstützen.

☑ ⟨$RM6$⟩ Der Speicher-Block muss ein Programm-Item aufnehmen können und so Programme in den Speicher des Lerncomputers laden.

☑ ⟨$RM7$⟩ Der Steuerwerk-Block muss ein Befehlssatz-Item aufnehmen können und so den aktiven Befehlssatz festlegen.

☑ ⟨$RM8$⟩ Der Programmier-Block muss ein Programm auf ein Programm-Item schreiben können. Dazu müssen das Programm-Item und ein Befehlssatz-Item im Programmier-Block liegen.

☑ ⟨$RM9$⟩ Der Programmier-Block muss über eine GUI mit Text-Editor für die Programmeingabe durch den Nutzer verfügen.

☑ ⟨$RM10$⟩ Es muss ein Item für die Speicherung und den Transport von Programmcode, für jeden angebotenen Befehlssatz und für ein ins Spiel integriertes Handbuch existieren.

☑ ⟨$RM11$⟩ Der Taktgeber-Block muss einen schrittweisen Modus anbietet, bei dem durch Interaktion der jeweils nächste Simulationstakt ausgelöst werden kann. Eine solche Interaktion löst auch den Programmstart aus.

1.2 Sollkriterien

☑ ⟨$RS1$⟩ Der Taktgeber-Block soll die Festlegung verschiedener Taktfrequenzen für den Lerncomputer ermöglichen.

☑ ⟨$RS2$⟩ Der Register-Block soll eine GUI zur Festlegung seiner Funktion in der Architektur haben.

☑ ⟨$RS3$⟩ Der Taktgeber soll den Takt visualisieren.

☑ ⟨$RS4$⟩ Der Speicher-Block soll eine GUI zur Inspizierung des aktuellen Speicherinhaltes bieten.

☑ ⟨$RS5$⟩ Es soll ein Brillen-Item existieren, dessen Tragen es erlaubt, bei Betrachtung eines Lerncomputer-Blocks Statusanzeigen mit technischen Informationen zu erhalten.

☑ ⟨$RS6$⟩ Der Steuerwerk-Block soll eine GUI zur Erklärung der angebotenen Befehlssätze und dafür benötigten Bauteile, sowie ob diese bereits angebunden sind, anbieten.

☑ ⟨$RS7$⟩ Die Bus-Blöcke sollen von außen sichtbar visualisieren, ob und welche Art von Daten auf dem Bus präsent sind.

☑ ⟨$RS8$⟩ Die Mod soll eine RISC-V-Architektur simulieren.

☑ ⟨$RS9$⟩ Die Mod soll vom Nutzer eingegebenen RISC-V Assembler-Code ausführen können.

☑ ⟨$RS10$⟩ Es sollen alle relevanten Bauteile des RISC-V-Prozessors durch Minecraft-Blöcke dargestellt werden.

☑ ⟨$RS11$⟩ Der ALU-Block soll die Rechenaktivität äußerlich visualisieren.

☑ ⟨$RS12$⟩ Der Speicher-Block und die Register sollen Zugriffe äußerlich visualisieren.

1.3 Kannkriterien

☑ ⟨$RC1$⟩ Das Programmier-Block-GUI kann ein optionales Fenster mit Tipps haben. Um diese Funktion bereitzustellen muss ein Befehlssatz-Item im Programmier-Block liegen.

☑ ⟨$RC2$⟩ Die Bus-Blöcke können ihr Aussehen anpassen, genauer die Lage ihrer Anschlüsse. Diese können sie an umliegenden Blöcken anpassen, um die Verbindung zu visualisieren.

☑ ⟨$RC3$⟩ Die Items und Blöcke können Rezepte haben, sodass man sie auch im Überlebensmodus erhalten kann.

☑ ⟨*RC*4⟩ Es kann einen PAUSE-Befehl geben, den der Nutzer im Programmcode einbauen kann und welcher die Simulation in den schrittweisen Modus übergehen lässt.

☑ ⟨*RC*5⟩ Es kann ein Redstone-Interface-Block existieren, der als zusätzliches Register mit Ein- und Ausgabemöglichkeit im Minecraft-Redstone-System agiert.

☐ ⟨*RC*6⟩ Der Programmier-Block kann Beispielprogramme enthalten.

☐ ⟨*RC*7⟩ Es kann ein Terminal-Block existieren, der mithilfe eines ansprechbaren Registers Text anzeigen kann.

☑ ⟨*RC*8⟩ Das Steuerwerk kann über eine äußere Visualisierung seines Zustandes verfügen.

☑ ⟨*RC*9⟩ Der Taktgeber kann einen Echtzeit-Modus haben, der Taktfrequenzen oberhalb der Standard-Minecraft-Ticks erlaubt.

☑ ⟨*RC*10⟩ Es kann ein Wireless-Register-Block existieren, der seinen Registerwert mit Wireless-Register-Blöcken anderer Computer in der Minecraft-Welt synchronisieren kann.

☐ ⟨*RC*11⟩ Der Terminal-Block kann ein auslesbares Register haben, in der Benutzer Text eingeben kann.

☑ ⟨*RC*12⟩ Es kann ein konfigurierbares Befehlssatz-Item existieren, mit dem neue MIMA-Befehle mit Mikrocode definiert werden können. Mit diesem kann unter anderem der Aufbau der MIMA modifiziert werden.

☑ ⟨*RC*13⟩ Das Programmier-Block-GUI kann eine einfache MIMA-/RISC-V-Assembler-Syntax-Unterstützung anbieten. Um diese Funktion bereitzustellen muss ein Befehlssatz-Item im Programmier-Block liegen.

☑ ⟨*RC*14⟩ Es kann eine GUI für das konfigurierbare Befehlssatz-Item existieren, die eine einfache und fehlerfreie Bearbeitung erleichtert.

☑ ⟨$RC15$⟩ Es kann ein konfigurierbares Befehlssatz-Item existieren, das Assembler-Code, Anzahl der Register, Busbreite und die Größe des Speichers spezifiziert.

☒ ⟨$RC16$⟩ Der Programmier-Block kann den Dateiimport aus Textdateien unterstützen.

2 Nichtfunktionale Anforderungen

2.1 Funktionalität

Erwartungen zu Projektbeginn:

Produktqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Angemessenheit				x
Richtigkeit	x			
Interoperabilität			x	
Ordnungsmäßigkeit				x

Einschätzung: Die Mod simuliert MIMA in allen getesteten Anwendungsfällen korrekt und die Richtigkeit für alle erwarteten Situationen ist anzunehmen. Ein Beispiel für die Korrektheit der Ausführung eines MIMA-Programms findet sich in Kapitel 6. Die Zusammenarbeit der Mod mit Minecraft ist gegeben, da die Mod-Elemente in Minecraft von Minecraft-Elementen erben oder deren Funktionalität nutzen. Mod-Anteile, welche nicht die Darstellung in Minecraft betreffen, sind von Minecraft klar getrennt und werden über die Mod-Blöcke und -Items angesteuert.

2.2 Sicherheit

Erwartungen zu Projektbeginn:

Produktqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Zuverlässigkeit		x		
Reife	x			
Fehlertoleranz			x	
Wiederherstellbarkeit			x	

Einschätzung: Die Modifikation ist in ihrer aktuellen Version noch nicht ausreichend zuverlässig, allerdings werden in den folgenden Projektwochen Bugs und Fehler priorisiert behandelt und es ist zu erwarten, dass Einschränkungen der Zuverlässigkeit auf ein Minimum reduziert werden können. Abgesehen von Problemen experimenteller Features führen Fehlerzustände nicht zum Versagen des Produktes und im Falle des Versagens in benannten Fällen werden Daten und Zustände im Rahmen der Möglichkeiten von Minecraft gesichert.

2.3 Benutzbarkeit

Erwartungen zu Projektbeginn:

Produktqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Verständlichkeit	x			
Erlernbarkeit	x			
Bedienbarkeit		x		
Effizienz				x
Zeitverhalten		x		
Verbrauchsverhalten				x

Einschätzung: Durch die intuitive Gestaltung der Oberflächen und die Interaktion mit Blöcken nach dem Standard-Minecraft-Schema sowie ein umfassendes Handbuch im Spiel sind Verständlichkeit, Erlernbarkeit und Bedienbarkeit der Mod insbesondere für Nutzer mit Minecraft-Grundkenntnissen gegeben. Das Zeitverhalten der Simulation wurde durch einen einfachen Praxistest als im Rahmen der Einschränkungen von Minecraft ausreichend genau bewertet.

2.4 Änderbarkeit

Erwartungen zu Projektbeginn:

Produktqualität	**sehr gut**	**gut**	**normal**	**nicht relevant**
Analysierbarkeit	x			
Modifizierbarkeit	x			
Stabilität			x	
Prüfbarkeit			x	
Übertragbarkeit				x
Anpassbarkeit				x
Installierbarkeit	x			
Konformität				x
Austauschbarkeit				x

Einschätzung: Der Code der Mod ist durch umfangreiches Javadoc und zumeist intuitive Benamung aller Elemente verständlich und im Rahmen des Entwurfs gut dokumentiert. Während der Implementierung der Kann-Kriterien und experimentellen Features zeigte sich eine hohe Erweiterbarkeit und Modifizierbarkeit. Ebenso bietet insbesondere das JSON-Format für den Befehlssatz mit seiner im Spiel zur Anpassung existierenden Oberfläche eine hohe Anpassbarkeit, ohne dabei im Code oder in Resourcen-Dateien arbeiten zu müssen. Wie erwartet können nur

Teile der Software automatisiert getestet werden und oftmals sind Nutzertests für die Prüfung des Produkts nötig. Die Installierbarkeit durch den Fabric-Mod-Loader ist einfach möglich.

2.5 Nichtfunktionale Produktanforderungen

☑ $\langle Q1 \rangle$ Die Mod stellt die fehlerfreie Ausführung inhaltlich korrekter Programme basierend auf einem korrekten Befehlssatz sicher.

☑ $\langle Q2 \rangle$ Die Mod arbeitet ohne für den Nutzer erkennbare Umwege und Schwierigkeiten mit Minecraft zusammen.

☑ $\langle Q3 \rangle$ Die Modifikation erhöht die Versagenshäufigkeit von Minecraft nicht.

☑ $\langle Q4 \rangle$ Die Mod ist intuitiv und leicht verständlich aufgebaut und insbesondere für Nutzer mit Minecraft-Vorkenntnissen sehr leicht zu erlernen und zu bedienen.

☑ $\langle Q6 \rangle$ Die Mod ist modular aufgebaut, leicht erweiterbar und verständlich dokumentiert.

☑ $\langle Q7 \rangle$ Die Ausführung jedes möglichen Simulationstaktes des Lerncomputers benötigt eine Maximaldauer von einem Minecraft-Tick.

☑ $\langle Q8 \rangle$ Die GUIs und Datenausgaben werden auf Deutsch und Englisch angeboten.

☐ $\langle Q9 \rangle$ Der Projektcode wird unter einer MIT-Lizenz auf GitHub veröffentlicht.

☑ $\langle Q10 \rangle$ Die Mod ist als Fabric-Mod exportierbar.

☐ $\langle Q11 \rangle$ Die Mod wird auf Curse-Forge veröffentlicht.

3 Entwurfsänderungen

Im Folgenden werden Entwurfsänderungen im Rahmen der Implementierungsphase im Vergleich zur Entwurfsphase aufgelistet, in einen gemeinsamen Kontext gesetzt und begründet.

3.1 Änderungen zur Optimierung und Vervollständigung der Simulation

Die hier aufgelisteten Änderungen beinhalten Funktionalität für die Simulation komplexerer MIMA-Befehle und RISC-V-Befehle sowie die Basis für die Erweiterung von RISC-V um Floating-Point-Funktionalität.

Klasse	Änderung	Grund der Änderung
AluController		
	public Value executeAluOperation(String operation)	Der Rückgabetyp wurde von void zu Value geändert, um das Berechnungsergebnis direkt in das Zielregister der Operation zu schreiben.
	Private Operationen für alle Alu-Operationen	Es wurden Methoden für alle für MIMA, RISC-V-Standard und RISC-V-Float nötigen Alu-Operationen hinzugefügt, an die die Ausführung durch executeAluOperation im entsprechenden Fall delegiert wird.
MemoryController		
	public Value getInitialProgramCounter()	Die Methode wurde hinzugefügt um den vom Assembler erkannten Initialwert des Proram-Counter-Registers zu setzten.
SystemClockController		
	public void onUserTickTriggered()	Lässt und Ticks durch Redstone Input auslösen. Wird von der Entity aufgerufen.

Klasse	Änderung	Grund der Änderung
	public void registerModel-Observer(I-Simulation-TimingObserver)	Die Methode wurde hinzugefügt, um den Observer über den Controller umzuleiten und somit Model-Zugriffe aus der Simulation entsprechend des MVC zu kapseln.
NonExecutable-Micro-Instruction-Exception	ganze Exception	Die Exception wurde eingefügt, um Fehler bei der Ausführung zu werfen.
AluModel		
	public String get-Operation()	Die Methode wurde hinzugefügt, um direkten Datenzugriff aus dem Controller zu ermöglichen.
	public Value get-Operand1()	Die Methode wurde hinzugefügt, um direkten Datenzugriff aus dem Controller zu ermöglichen.
	public Value get-Operand2()	Die Methode wurde hinzugefügt, um direkten Datenzugriff aus dem Controller zu ermöglichen.
MemoryModel		
	public Value getInitialPro-gramCounter()	Die Methode wurde hinzugefügt um den vom Assembler erkannten Initialwert des Proram-Counter-Registers zu setzten.
AluInstruction		
	public Micro-Instruction clone(String[] from, String to)	Die Methode wurde in IExecutableMicroInstruction hinzugefügt. Siehe Eintrag dafür für die Begründung.
ComplexMicro-Instruction		

Klasse	Änderung	Grund der Änderung
	public MicroInstruction getFilled(Map<String, String> argumentsInstructionMap, HashMap<Integer, String> intRegisters, HashMap<Integer, String> floatRegisters)	Die Methode wurde hinzugefügt, um Mikroinstruktionen mit parametrisierten Ursprungsregistern und Speicheradressen dynamisch füllen zu können.
ConditionedInstruction		
	public MicroInstruction getFilled(Map<String, String> argumentsInstructionMap, HashMap<Integer, String> intRegisters, HashMap<Integer, String> floatRegisters)	Diese Methode überschreibt die gleichnamige Methode aus ComplexMicroInstruction und ergänzt dabei die hier im besonderen nötige Funktionalität, auch parametrisierte Register und Speicheradressen in der Bedingung zu füllen.
	public MicroInstruction clone(String[] from, String to)	Die Methode wurde in IExecutableMicroInstruction hinzugefügt. Siehe Eintrag dafür für die Begründung.
DataMovementInstruction		
	public MicroInstruction clone(String[] from, String to)	Die Methode wurde in IExecutableMicroInstruction hinzugefügt. Siehe Eintrag dafür für die Begründung.

Klasse	Änderung	Grund der Änderung
IExecutable-Micro-Instruction		
	MicroInstruction clone(String[] from, String to)	Die Methode wurde hinzugefügt, um die Kopie einer Mikroinstruktion zu erzeugen, in der parametrisierte Register und Speicheradressen ausgefüllt wurden.
Instruction		
	public Instruction(Instruction instruction, String binary, HashMap<Integer, String> intRegisters, HashMap<Integer, String> floatRegisters)	Der Konstruktor erfüllt die Funktion eines Copy-Konstruktors beim Clonen der Instruktionen nach dem Ausfüllen von parametrisierten Mikroinstruktionen.
	public boolean matchesBinary(String binaryValue)	Die Methode bietet Funktionalität, um einen binären String mit dem Format der Binär-Übersetzung der Instruktion abzugleichen und so das Übersetzen von Instruktionen im Speicher zurück in ausführbare Instruktionen zu ermöglichen.
Instruction-BuildException	ganze Exception	Die Exception wurde eingeführt, um Probleme bei Extrahieren des Instruction Sets aus der .json-Datei abzufangen.
InstructionSet-Builder		
	public static InstructionSetModel buildInstructionSetModel (InputStream is) throws UnsupportedEncodingException, InstructionBuildException	Die InstructionBuildException wurde der Methode hinzugefügt, um Fehler bei der .json-Extraktion abzufangen.

Klasse	Änderung	Grund der Änderung
InstructionSet-Model		
	public Integer getFloatRegister(String key)	Diese Methode implementiert die neue gleichnamige Methode im IQueryableInstructionSet.
	public IQueryableInstruction getInstructionFromBinary(String binaryValue)	Diese Methode implementiert die neue gleichnamige Methode im IQueryableInstructionSet.
	public String getRegisterInitialValue(String key)	Diese Methode implementiert die neue gleichnamige Methode im IQueryableInstructionSet.
	public boolean equals(Object o)	Die Methode erlaubt, zu prüfen, ob zwei Befehlssätze gleich sind.
InstructionSet-Registers		
	public void generateRegisterAddressMaps()	Die Methode wurde eingeführt, um die Registeradressierung von RISC-V umzusetzen.
	public List<String> getRegisterNames()	Die Methode bietet Zugriff auf eine Auflistung der Namen aller Register, die für den Befehlssatz benötigt werden, um so die Vollständigkeit des Computers verallgemeinert prüfen zu können.
	public String getInitialValue(String key)	Die Methode wurde eingeführt, um auf den Initalwert der Register zugreifen zu können und sie zu initialisieren.
	public HashMap<Integer, String> getIntRegisterAddressMap()	Die Methode wird genutzt, um die Integer-Register an die Instruktion zu übergeben, wenn sie kopiert wird und dabei die Register-Adressen in Identifier umgewandelt werden.

Klasse	Änderung	Grund der Änderung
	public HashMap<Integer, String> getFloatRegisterAddressMap()	Die Methode wird genutzt, um die Float-Register an die Instruktion zu übergeben, wenn sie kopiert wird und dabei die Register-Adressen in Identifier umgewandelt werden.
IQueryableInstruction		
	IExecutableMicroInstruction[] getExecution()	Diese Methode existierte in einer Klasse und wurde zur Kapselung in das Interface gezogen.
IQueryableInstructionSetModel		
	Integer getFloatRegister(String key)	Die Methode erlaubt den Zugriff auf ein durch den key gegebenes Float-Register und wurde für die Ergänzung von RISC-V um Floating-Point-Funktionalität eingeführt.
	IQueryableInstruction getInstructionFromBinary(String binaryValue)	Die Methode bietet die Funktionalität, eine Instruktion aus ihrem Binär-String zu laden. In dieser Form liegt die Instruktion vor, damit sie so im Speicher dargestellt werden kann.
	List<String> getRegisterNames()	Die Methode bietet Zugriff auf eine Auflistung der Namen aller Register, die für den Befehlssatz benötigt werden, um so die Vollständigkeit des Computers verallgemeinert prüfen zu können.
	String getRegisterInitialValue(String key)	Die Methode bietet Zugriff auf Initialwerte der Register, um beispielsweise das Eins-Register zu initialisieren.
MemoryInstruction		
	public MicroInstruction clone(String[] from, String to)	Die Methode wurde in IExecutableMicroInstruction hinzugefügt. Siehe Eintrag dafür für die Begründung.

Klasse	Änderung	Grund der Änderung
	public String getFlag()	Die Memory-Flag wurde in die MemoryInstruction gezogen, da sie vorgibt, ob das Ursprungs- oder das Zielregister im Speicher liegt und so Fehler beim Zugriff vorgebeugt werden. Diese Methode bietet Zugriff auf die Flag.
MicroInstruction		
	public MicroInstruction getFilled(Map<String, String> argumentsInstructionMap,HashMap<Integer, String> intRegisters, HashMap<Integer, String> floatRegisters)	Die Methode wurde hinzugefügt, um Mikroinstruktionen mit parametrisierten Ursprungsregistern und Speicheradressen dynamisch füllen zu können.
Memory		
	public void setInitialProgramCounter(Value address)	Die Methode wurde hinzugefügt um den vom Assembler erkannten Initialwert des Proram-Counter-Registers zu speichern.
	public Value getInitialProgramCounter()	Die Methode wurde hinzugefügt um den vom Assembler erkannten Initialwert des Proram-Counter-Registers zu setzten.
Value		
	public static Value fromBinary(String s, int length, boolean autoSignExtend)	Der Methode wurde ein optionaler Parameter hinzugefügt um sign extension zu unterstützen

Klasse	Änderung	Grund der Änderung
	public boolean lowerThan(Value comparator)	Die Methode wurde hinzugefügt um das vergleichen von Values bei z.B. JMN oder bge zu erleichtern
	public boolean lowerThanUnsigned(Value comparator)	Die Methode wurde hinzugefügt um das vergleichen von Values bei z.B. JMN oder bge zu erleichtern
	public boolean lowerThanFloat(Value comparator)	Die Methode wurde hinzugefügt um das vergleichen von Values bei z.B. JMN oder bge zu erleichtern
	public boolean greaterThan(Value comparator)	Die Methode wurde hinzugefügt um das vergleichen von Values bei z.B. JMN oder bge zu erleichtern
	public boolean greaterThanUnsigned(Value comparator)	Die Methode wurde hinzugefügt um das vergleichen von Values bei z.B. JMN oder bge zu erleichtern
	public boolean greaterThanFloat(Value comparator)	Die Methode wurde hinzugefügt um das vergleichen von Values bei z.B. JMN oder bge zu erleichtern
	public Value negate()	Die Methode wurde eingefügt um die Eingabe von negativen Werten im Assembly code zu erleichtern
SystemClockController		
	public int getClockSpeed()	Die Methode wurde eingeführt, um Zugriffe auf die Takt-Geschwindigkeit im Taktgeber-Model, welche von der Simulation ausgehen, nach MVC zu kapseln.
	public ClockMode getClockMode()	Die Methode wurde eingeführt, um Zugriffe auf den Takt-Modus im Taktgeber-Model, welche von der Simulation ausgehen, nach MVC zu kapseln.

3.2 Änderungen zur Visualisierung der Simulation

Im Entwurf gab es einen boolean hasunqueriedStateChange() das zur Visualisierung genutzt werden sollte und zusätzlich eine Synchronisation zum View anstoßen sollte. Diese Doppelrolle erwies sich als nicht umsetzbar, da wir damit in Probleme mit Minecraft gekommen sind, wenn wir die Daten zur richtigen Zeit aktuell im Client haben wollten. Deshalb haben wir die Funktion aufgespalten und ein boolean activeVisualisation() für die Visualisierung hinzugefügt.

Klasse	Änderung	Grund der Änderung
BusController		
	public void setBusSystem-Model(Bus-SystemModel busSystemModel)	Die Methode übergibt dem BusModel über den Controller sein BusSystemModel, da so am einfachsten die Aktivitätszustände einzelner Busse von BusSystem angefragt werden können.
Computer-Block-Controller		
	public void activateVisualization()	Die Methode bietet der Simulation die Möglichkeit, die Visualisierung eines Blocks zu aktivieren.
	public void stop-Visualization()	Die Methode wurde hinzugefügt, um der Simulation zu erlauben, die Visualisierung eines Blocks zu deaktivieren.
IConnectable-Computer-BlockEntity		
	enum ComputerEffects	Das Enum wurde hinzugefügt, um der Simulation die Möglichkeit zu geben, den Blöcken in bestimmten Situationen Effekte zuzuordnen.
IQueryable-Computer-Controller		
	Void stopVisualization()	Die Methode ermöglicht, die Visualisierung des Blocks aus der Simulation und dem Clustering über seinen Controller zu stoppen.
IQueryableSimController		

Klasse	Änderung	Grund der Änderung
	public void activateVisualization()	Die Methode ermöglicht, die Visualisierung des Blocks aus der Simulation über seinen Controller zu aktivieren.
Executor		
	public Executor(List<IQueryableSimController> blockControllers, int wordLength, IBusSystem busSystem)	Der Konstruktor erhielt den Parameter wordLength für das gleichnamige Attribut, um das Extrahieren von Konstanten aus Binär-Strings im Falle parametrisierter Mikroinstruktionen zu ermöglichen. Der Konstruktor erhielt den Parameter busSystem für das gleichnamige Attribut, um dem Executor die Weitergabe seiner Aktionen an das Bus-System zu deren Visualisierung zu ermöglichen.
IRealtimeSimulationCallbackReceivable	ganzes Interface	Ermöglicht die Benachrichtigung des SimulationTimeHandlers über das Ende eines Echtzeit-Simulationstaktes durch den SimulationSequenceHandler, um so Simulation und Visualisierung im Echtzeit-Modus steuern zu können.
SimulationSequenceHandler		
	public SimulationSequenceHandler(List<IQueryableSimController> blockControllers, IBusSystem busSystem, IRealtimeSimulationCallbackReceivable callbackReceivable)	Der Konstruktor erhielt die Parameter busSystem und callbackReceivable für die gleichnamigen Attribute, um aus dem SimulationSequenceHandler die Visualisierung ansteuern und den Simulationsfluss-Status nach außen geben zu können.
	public void resetVisualisation()	Die Methode ermöglicht das Zurücksetzen der Visualisierung.

Klasse	Änderung	Grund der Änderung
	public void fullVisualisation()	Die Methode ermöglicht, die Visualisierung für den gesamten Computer zu aktivieren. Dies wird für den Echtzeit-Modus genutzt.
	public void setVisualizationMode(VisualisationMode mode)	Die Methode ermöglicht, den Modus der Visualisierung entsprechend des Echtzeit-Modus und der normalen tick- oder schrittweisen Simulation zu setzen.
public enum VisualisationMode	ganz neu hinzugefügt	Das Enum ermöglicht die Unterscheidung verschiedener Visualisierungsmodi.
SimulationTimeHandler		
	public class SimulationTimeHandler implements ISimulationTimingObserver, IRealtimeSimulationCallbackReceivable	Das Callback-Receivable-Interface wurde der Klasse hinzugefügt, um zu ermöglichen, dass der SimulationSequenceHandler den SimulationTimeHandler über den Kontrollfluss-Status der Simulation informieren kann.
	public SimulationTimeHandler(List<IQueryableSimController> blockControllers,IBusSystem busSystem)	Der Konstruktor erhielt den neuen Parameter busSystem, um die Visualisierung der Busse manipulieren zu können.
BlockModel		
	public boolean getVisualisationState()	Die Methode ermöglicht dem View, anzufragen, kann, ob der entsprechende Block leuchten soll.
	public void setVisualisationState(boolean visualisationState)	Die Methode ermöglicht, den Visualisierungszustand des Blocks, zu dem das Model gehört, zu setzen.

Klasse	Änderung	Grund der Änderung
BusModel		
	public boolean getVisualisationState()	Die Methode wurde hinzugefügt und überschreibt die Methode aus dem BlockModel, da ein Bus keinen eigenen Visualisierungszustand hat, sondern für die Visualisierung von Buspfaden beim Bus-System anfragen muss, ob er zum aktiven Pfad gehört.
IController-Queryable-BlockModel		
	void setVisualisationState(boolean visualisationState)	Die Methode ermöglicht den Controllern, den Visualisierungszustand im Model zu setzen, damit der View diesen anfragen kann.
IView-Queryable-BlockModel		
	boolean getVisualisationState()	Die Methode ermöglicht dem View, den Visualisierungsstatus beim Model anzufragen.
BusSystem-Model		
	public class BusSystemModel implements IQueryableBusSystem, IBusSystem	Die Klasse implementiert nun auch das IBusSystem-Interface, welches Visualisierungsfunktionalität zur Verfügung stellt.
	public void setBusDataPath (BlockPosition startPos, BlockPosition endPos, Value presentData)	Die Methode wird zum Erstellen von Visualisierungspfaden von Bussen zwischen zwei Blöcken benötigt.
	public void resetVisualisation()	Die Methode setzt die aktiv visualisierten Pfade zurück.
	public void activateVisualisation()	Die Methode setzt alle Blöcke, die zu einem BusSystemModel gehören, auf aktiv.

Klasse	Änderung	Grund der Änderung
IBusSystem	neues Interface	Das Interface wurde eingeführt, damit die Simulation die Busse visualisieren kann, ohne die Graph-Repräsentation zu kennen.
IQueryable-BusSystem		
	void resetVisualisation()	Das Bussystem berechnet die Wege der Daten. Diese Methode wird am Ende des Taktes aufgerufen, um die gesetzten Wege zurückzusetzen.
Computer-BlockEntity		
	public boolean isActive()	Die Methode wird gebraucht, damit Busblöcke anfragen können, ob der Block an einer Datenbewegung aktiv ist, um zu entscheiden, welche Zweige des Busblocks visualisiert werden müssen.
	public void spawnEffect-(ComputerEffect effect)	Die Methode ermöglicht den Rauch-Effekt im Echtzeit-Modus und die Explosion bei Teilen durch Null.

3.3 Änderungen zur Visualisierung von Blöcken

Klasse	Änderung	Grund der Änderung
BusSystem-Model		
	public List<Block-Position> get-BusSystemNeighbors(BlockPosition blockPosition)	siehe Interface IQueryableBusSystem
IQueryable-BusSystem		

Klasse	Änderung	Grund der Änderung
	List<BlockPosition> getBusSystemNeighbors(BlockPosition blockPosition)	Die Methode wird benötigt, damit die Busse ihre in Minecraft sichtbaren Verbindungen mir der Graph-Repräsentation abgleichen können.
BusBlock		
	@Override public void onPlaced	Um das Kannkriterium $\langle RC2 \rangle$ umzusetzen, das nicht im Entwurf enthalten war.
	@Override public BlockState getStateForNeighborUpdate	Um das Kannkriterium $\langle RC2 \rangle$ umzusetzen, das nicht im Entwurf enthalten war.
BusBlockEntity		
	public void updateBlockState()	Um das Kannkriterium $\langle RC2 \rangle$ umzusetzen, das nicht im Entwurf enthalten war.
SystemClockBlock		
	public static final int MAX_CURSORPOS	Der Zeiger in der Textur hat 8 verschiedene Stellen.
	public static final IntProperty CURSORPOS	Die Texturen werden über verschiedene BlockStates geändert.
SystemClockBlockEntity		
	public int getSystemClockSpeed()	Diese Methode wurde hinzugefügt, um die derzeitige Geschwindigkeit in der Benutzeroberfläche der System Clock anzuzeigen.
	public String getSystemClockMode()	Diese Methode wurde hinzugefügt um die Visualisierung des derzeitigen Modus in der Benutzeroberfläche anzuzeigen.
ConnectingComputerBlock	ganze Klasse neu	Damit die Busse ihr Aussehen an die Umgebung anpassen können. Die Funktionalität ist für eine bessere Erweiterbarkeit hier ausgelagert.

3.4 Änderungen am Entwurf zum Refactoring und zur Optimierung des Clusterings

Klasse	Änderung	Grund der Änderung
Computer-Block-Controller		
	public void startClustering-(BlockPosition pos)	Da nicht immer beim Erstellen eines Controllers das Clustering gestartet werden, bietet dieses Methode das starten des Clusterings zu einem späteren Zeitpunkt an.
ControlUnit-Controller		
	@Override public void set-ClusterHandler-(ClusterHandler clusterHandler)	Wenn die ControlUnit einem neuen ClusterHandler zugewiesen wird, muss dieser informiert werden, ob ein Befehlsatz-Item in der ControlUnit eingelegt ist.
Memory-Controller		
	public boolean is-MemorySet()	Nötig, damit der der ClusterArchitectureHandler auf isMemorySet() im Model zugreifen kann.
ClusterHandler		
	public boo-lean setIst-Model(IQueryable-InstructionSet-Model istModel)	Methode zum setzen des aktuellen InstructionSet-Model.
	public void removeIstModel()	Methode um das aktuelle InstructionSetModel zu entfernen.
BusModel		
	@Override public void setPosition-(BlockPosition position)	Diese Methode schreibt zusätzlich in die Block-Position, das es sich um einen Bus handelt. Dies wird im BusSystemModel benötigt um BusBlöcke und NichtBusBlöcke zu unterscheiden.
IController-Queryable-BlockModel		

Klasse	Änderung	Grund der Änderung
	BlockPosition getPosition()	Die BlockPosition der Blöcke ist für das Erstellen von der Graph-Repräsentation relevant. Um die BlockPosition zu bekommen ist diese Methode notwendig.
MemoryModel		
	public boolean isMemorySet()	Diese Methode wird gebraucht um zu prüfen ob eine Architektur valide ist, da eine Computer ohne beschriebenen Memory nicht valide ist.
BlockPosition		
	public static boolean isNeighbourPosition(BlockController block1, BlockController block2)	Diese Methode wird gebraucht, um zu prüfen, ob die ALU-Register neben der ALU stehen oder nicht.
InstructionSetModel		
	public List<String> getRegisterNames()	Die Methode bietet Zugriff auf eine Auflistung der Namen aller Register, die für den Befehlssatz benötigt werden, um so die Vollständigkeit des Computers verallgemeinert prüfen zu können.

3.5 Refactoring des Daten-Transfers

Der Entwurf enthielt für den Datentransfer zwischen Model und View und Client und Server ein Data-Paket, das sich leicht in NBT-Daten konvertieren ließ. Grundsätzlich funktioniert der gesamte Datentransfer immer noch nach diesem Schema, allerdings wurde viel auf dieses Grundkonstrukt aufgebaut, um einerseits mehr Funktionalität bereitzustellen und andererseits um mit Eigenheiten von Minecraft umzugehen, die zum Zeitpunkt des Entwurfes noch nicht bekannt waren. Zusätzlich kam durch die Aufteilung und Refactoring der hasUnqueriedStateChange Variable Methoden hinzu, um Daten nur dann zu synchronisieren, wenn es nötig ist. Man könnte das zwar auch jeden tick machen, bei großen Computern würde das aber wahrscheinlich Performance kosten. Daten immer zu den richtigen Zeiten zu synchronisieren, aber auch nicht zu oft, war nicht einfach und hat einige neue Methoden nötig gemacht. Da die Screens jetzt viel Funktionalität und Informationen bereitstellen, müssen auch mehr Daten als geplant synchronisiert werden.

Klasse	Änderung	Grund der Änderung
BlockModel		
	public boolean hasUnqueried-StateChange() aus allen erbenden Klassen weg	Durch das Aufteilen der hasUnqueriedStateChange Funktionalität konnte die verbleibende Funktionalität in die Superklasse abstrahiert werden.
	public void on-StateQuery()	Zeigt dem Block Model, dass seine Daten erfolgreich im View angekommen sind. Ist nötig, da während der Welt laden Daten manchmal verloren gehen können.
	public void on-StateChange()	Wird außerhalb der Models genutzt, um nochmal neue Daten anzufragen.
ControlUnit-Model		
	public void setClustering-Data(IData-Element clusteringData)	Im ControlUnitScreen sollen alle angeschlossenen und fehlenden Blöcke angezeigt werden. Dafür brauchen wir hier diese Daten.
IController-Queryable-BlockModel		
	void onState-Change()	Wird von Controllern genutzt, um nochmal neue Daten anzufragen.
IQueryable-BlockModel		
	IDataElement getData()	Die Methode wurde hinzugefügt, um Datenzugriffe aus dem View auf die Models zu verallgemeinern.
IView-Queryable-BlockModel		
	IDataElement getData()	Die Methode wurde hinzugefügt, um Datenzugriffe aus dem View auf die Models zu verallgemeinern.
	boolean ha-sUnqueried-StateChange()	Vom View genutzt, um anzufragen, ob neue Daten synchronisiert sind.

Klasse	Änderung	Grund der Änderung
	void onStateQuery();	Genutzt um mitzuteilen das die Daten erfolgreich angekommen sind.
RegisterModel		
	public static final String UNASSIGNED_REGISTER	Standardname damit der View fehlersicher checken kann ob ein Register schon ausgewählt wurde.
	public void setMissingAvailableRegisters(String[] missingAvailableRegisters)	Die Methode wurde hinzugefügt, damit im Register Screen angezeigt werden kann welche Registertypen noch frei sind.
SystemClockModel		
	public void setClockSpeed(int clockSpeed)	Diese Methode erlaubt das Setzen verschiedener Taktgeschwindigkeiten, welche im Tick-Modus berücksichtigt werden.
DataConstants	ganze Klasse neu	In NBT Containern und Data Container sind Daten in einem key, value Format gespeichert. In dieser Klasse befinden sich alle Schlüssel las Konstanten.
Memory		
	public boolean equals(Object o)	Im Screen muss unterschieden werden, ob das neu eingelegte Programm wirklich neu ist, um die Simulation nicht, anzuhalten, wenn sich eigentlich nichts geändert hat.
	public int hashCode()	siehe oben
ControlUnitScreen		
	public List<ArchitectureEntry> fetchEntries()	Zur Darstellung der Architektur müssen die Daten aus der Entity geholt werden.
ControlUnitBlockEntity		

Klasse	Änderung	Grund der Änderung
	public void inventory-Changed()	Anstatt wie geplant selbst immer das Inventar zu beobachten, gibt es eine Minecraft Methode im Screen Handler. Diese Methode wird dann vom Screen Handler aufgerufen
	public void updateUI()	Ist da um den Edge Case das in einem Cluster mehrere Control Units mit verschiedenen Befehlssatzes existieren durch Auswurf des Befehlssatzes zu lösen.
	public List[] get-Structure()	Wird benötigt, um im Screen die Architektur darzustellen.
MemoryBlock-Entity		
	public void inventory-Changed()	Anstatt wie geplant selbst immer das Inventar zu beobachten, gibt es eine Minecraft Methode im Screen Handler. Diese Methode wird dann vom Screen Handler aufgerufen
Memory-ScreenHandler		
	public String get-MemoryLine(int address)	Diese Methode kann aus den synchronisierten Daten eine Speicherzeile zum Anzeigen auslesen.
	public int get-MemorySize()	Diese Methode kann aus den synchronisierten Daten die Gesamtgröße auslesen. Das wird zur Anzeige der Scrollbar benötigt.
Register-ScreenHandler		
	public String get-RegisterValue()	Diese Methode ermöglicht es, den aktuellen Wert des Registers vom Server zum Client zu laden, um ihn dort in der Benutzeroberfläche anzuzeigen.
	public String get-CurrentRegister()	Diese Methode ermöglicht es, den gesetzten Typ des Registers vom Server zu Laden, um ihn anzuzeigen in der Benutzeroberfläche des Registers.
	public List<String> getRegisters-(String key)	Diese Methode wird, genutzt um die Liste der möglichen Register vom Server zu Laden, damit im Client, in der Benutzeroberfläche des Registers, die Auswahl des Registertyps zu ermöglichen.
SystemClock-BlockEntity		

Klasse	Änderung	Grund der Änderung
	public void updateUI()	Der Zeiger in der Block-Textur kann rotieren. Diese Methode ändert den BlockState, um die Textur zu ändern.
ComputerBlockEntity		
	@Overide public void writeNbt(NbtCompound nbt)	Minecraft Methoden, die für die Synchronisation eigener Daten genutzt werden.
	@Overide public void readNbt(NbtCompound nbt)	Minecraft Methoden, die für die Synchronisation eigener Daten genutzt werden.
	public void requestData()	Über das Client-Server Network kann ein Screen neue Daten anfragen. Dann wird diese Methode ausgeführt.
ComputerBlockEntityWithInventory		
	public void readNbt(NbtCompound nbt)	Minecraft Methoden, die für die Synchronisation eigener Daten genutzt werden.
	public void writeNbt(NbtCompound nbt)	Minecraft Methoden, die für die Synchronisation eigener Daten genutzt werden.
	public void inventoryChanged()	Implementierung hängt von Unterklassen ab. Muss jede Unterklasse haben.
ProgrammingBlockEntity		
	public void writeNbt(NbtCompound nbt)	Minecraft Methoden, die für die Synchronisation eigener Daten genutzt werden.
	public void readNbt(NbtCompound nbt)	Minecraft Methoden, die für die Synchronisation eigener Daten genutzt werden.

Klasse	Änderung	Grund der Änderung
ModBlock-EntityWith-Inventory		
	public void readNbt(Nbt-Compound nbt)	Minecraft Methoden, die für die Synchronisation eigener Daten genutzt werden
	public void writeNbt(Nbt-Compound nbt)	Minecraft Methoden, die für die Synchronisation eigener Daten genutzt werden
Networking-Constants	neu hinzugefügt	Stellt alle Adressen für die Client-Server und Server-Client einmal gesammelt als Konstanten bereit.

3.6 Entwurfsänderungen für neue, verbesserte und angepasste Screens

Klasse	Änderung	Grund der Änderung
ControlUnit-Controller		
	public void reject-IstModel()	Kann das Item aus dem Block auswerfen, wenn es schon ein Item gibt.
ControlUnit-Screen		
	aus Minecraft angebotene Methoden für Screens	Die Methoden sind nötig für die Screen-Funktionalität des Programmier-Screens und wurden aus der Minecraft-Superklasse HandledScreen überschrieben.
MemoryScreen		
	aus Minecraft angebotene Methoden für Screens	Die Methoden sind nötig für die Screen-Funktionalität des Programmier-Screens und wurden aus der Minecraft-Superklasse HandledScreen überschrieben.
Programming-Screen		
	aus Minecraft angebotene Methoden für Screens	Die Methoden sind nötig für die Screen-Funktionalität des Programmier-Screens und wurden aus der Minecraft-Superklasse HandledScreen überschrieben.
RegisterScreen		

Klasse	Änderung	Grund der Änderung
	aus Minecraft angebotene Methoden für Screens	Die Methoden sind nötig für die Screen-Funktionalität des Programmier-Screens und wurden aus der Minecraft-Superklasse HandledScreen überschrieben.
SystemClockScreen		
	aus Minecraft angebotene Methoden für Screens	Die Methoden sind nötig für die Screen-Funktionalität des Programmier-Screens und wurden aus der Minecraft-Superklasse HandledScreen überschrieben.
TerminalScreen		
	aus Minecraft angebotene Methoden für Screens	Die Methoden sind nötig für die Screen-Funktionalität des Programmier-Screens und wurden aus der Minecraft-Superklasse HandledScreen überschrieben.
InstructionSetScreen		
	aus Minecraft angebotene Methoden für Screens	Die Methoden sind nötig für die Screen-Funktionalität des Programmier-Screens und wurden aus der Minecraft-Superklasse Screen überschrieben.
ManualScreen		
	aus Minecraft angebotene Methoden für Screens	Die Methoden sind nötig für die Screen-Funktionalität des Programmier-Screens und wurden aus der Minecraft-Superklasse Screen überschrieben.
ControlUnitScreenHandler		
	public ControlUnitScreenHandler(int syncId, PlayerInventory playerInventory, ModBlockEntity blockEntity)	Es wurde ein neuer Konstruktor benötigt, der die BlockEntity als Parameter hat, um auf deren Daten zugreifen zu können.
RegisterBlock		

Klasse	Änderung	Grund der Änderung
	public ActionResult onUse(BlockState state, World world, BlockPos pos, PlayerEntity player, Hand hand, BlockHitResult hit)	Diese Methode wird benötigt zum öffnen der Register-GUI.
RegisterScreenHandler		
	public RegisterScreenHandler-(int syncId, PlayerInventory inventory, ModBlockEntity blockEntity)	Es wurde ein neuer Konstruktor benötigt, der die BlockEntity als Parameter hat, um auf deren Daten zugreifen zu können.
ProgrammingScreenHandler		
	public String getCode()	Die Methode ist nötig, weil der Code erhalten werden soll, wenn der Screen geschlossen und wieder geöffnet wird. Die Methode holt den Code aus der Entity.
ModScreenHandler		
	public ModBlockEntity getBlockEntity()	Um Daten vom Server zu erhalten, muss oftmals auf die Block Entity zugegriffen werden, deswegen ist es nötigt die Block Entity erfragen zu können.
InstructionSetItem		
	public TypedActionResult-<ItemStack> use(World world, PlayerEntity user, Hand hand)	Die Methode wurde eingefügt, um den Screen des Befehlssatz-Items öffnen zu können.

Klasse	Änderung	Grund der Änderung
InstructionSet-Model		
	public ArrayList<String[]> getPossible-Instructions()	Die Methode erlaubt den Zugriff auf alle möglichen Instruktionen im Befehlssatz, um diese als Hilfe im Programmierblock-Screen darzustellen.

3.7 Neue Widgets

Die von Minecraft bereitgestellten Klassen sind kaum erweiterbar, da sie meistens zugeschnitten sind auf den nutzen in Minecraft Vanilla. Deswegen haben wir uns dazu entschieden, die Widgets, die wir verwenden wollen in großen Teilen eigenständig zu implementieren. Insofern behandelt diese Sektion ausschließlich neue Klassen.

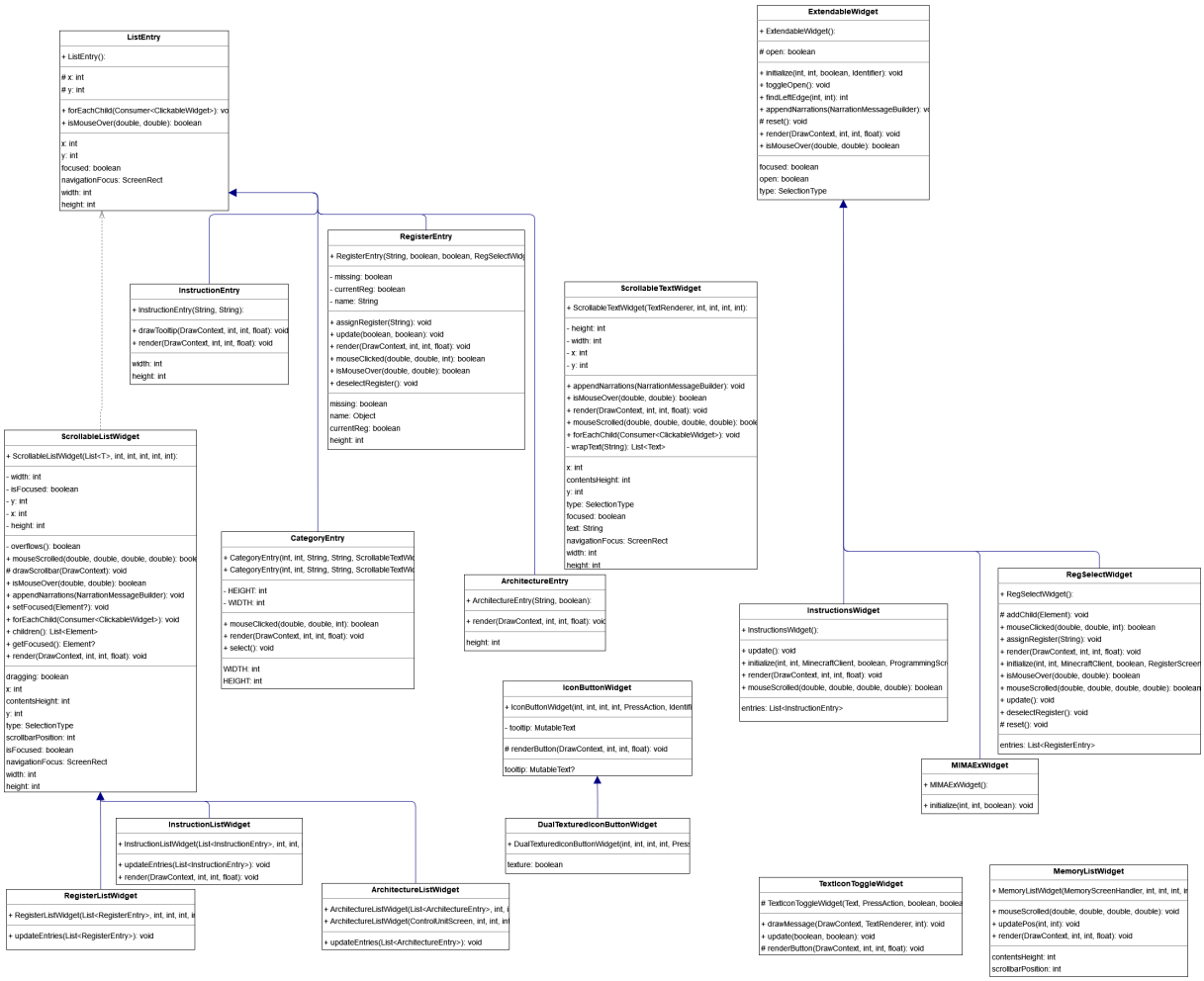

Abbildung 3.1: Neue Screen-Widgets

- **ExtendableWidget**: Diese Klasse wurde hinzugefügt, um die grundlegende Funktion für alle ausklappbaren Widgets bereit zu stellen.
  - InstructionsWidget
  - MIMAExWidget
  - RegSelectWidget
- **ScrollableListWidget**: Diese Klasse wurde hinzugefügt um die grundlegenden Funktionen für alle scrollbare Listen, mit Variablen Eintrags Typen, bereit zu stellen.
  - ArchitectureListWidget
  - InstructionListWidget
  - MemoryListWidget
  - RegisterListWidget
- **ListEntry**: Diese Klasse wird als Superklasse für alle Einträge genutzt, welche in einer ScrollableList angezeigt werden sollen.
  - ArchitectureEntry
  - CategoryEntry
  - InstructionEntry
  - RegisterEntry
- **Button Widgets**: Die folgenden Klassen sind Buttons, welche hinzugefügt wurden, da Minecraft zum Rendern ihrer Buttons den Atlas Manager nutzt. Der Atlas Manager ist (so weit ich es verstanden habe) eine Klasse welche alle Texturen von Minecraft kennt. Da dieser jedoch nicht unsere eigenen Texturen kennt, haben wir die Buttons selber geschrieben. Dies hat außerdem geholfen Besonderheiten in der Funktionsweise von Minecraft Klassen zu umgehen.
  - DualTexturedIconButtonWidget
  - IconButtonWidget
  - TextIconToggleWidget
- **ScrollableTextWidget**: Da Minecraft nicht mit scrollbaren Texten arbeitet, mussten wir ein neues Widget entwickeln.

- **Text-Editor**: Der Text-Editor wurde hinzugefügt, da ungekennzeichnete Zeilenumbrüche im, von Minecraft bereitgestellten EditBoxWidget, Code sehr unverständlich gemacht haben. Des Weiteren konnten wir so ein Syntax Highlighting zu implementieren.
  - AssemblerSyntaxTextEditWidget
  - Line
  - TextEditWidget

3.8 Änderungen und Ergänzungen zur Implementierung von im Entwurf nicht einbezogenen Kann-Kriterien

Klasse	Änderung	Grund der Änderung
InstructionSet-Builder		
	public static InstructionSetModel buildInstructionSetModel (String s) throws UnsupportedEncodingException, InstructionBuildException	Die Methode fügt die Funktionalität hinzu, Instruction Sets auch aus einem String zu extrahieren. Zur Veränderung der Instruction Sets müssen ihre Daten auf dem Item gespeichert werden und nicht in einer Datei. In dem Item-NBT liegen sie als String, deshalb ist hier eine Methode mit String-Parameter wichtig.
SystemClock-Controller		
	public void setSimulationMode(ClockMode mode)	Zur Umsetzung der PAUSE Instruction.
Wireless-Register-Controller	ganze Klasse neu	siehe WirelessRegisterBlock
Wireless-RegisterModel	ganze Klasse neu	siehe WirelessRegisterBlock
Wireless-RegisterBlock	ganze Klasse neu	Benötigt für die Implementierung des Wireless-Registers aus Kannkriterium $\langle RC10 \rangle$.

Klasse	Änderung	Grund der Änderung
Wireless-RegisterBlock-Entity	ganze Klasse neu	siehe WirelessRegisterBlock
Terminal	ganze neue Klassen	neue Klassen: TerminalEntity, TerminalBlock, TerminalInputController, TerminalModeController. Eine Entity besitzt drei RegisterController (Input, Output, Modus). Wobei mit dem Modus der Input gestuert werden kann. Deswegen exestieren die beiden extra Controller. Sonst benutzt das Terminal Register Klassen. Für den Screen neu: TerminalScreen, TerminalSelectionWidget da wir drei Register zuweisen können müssen.
Redstone I/O-Register		
	ganze Klassen neu	Für jeweils Input und Output einen Block und eine Entity, die vom Register erben. Für den Rest werden Register, Klassen verwendet.

4 Zeitplan und Vorgehen

Dieses Kapitel enthält ein Diagramm des Commit-Verlaufs für die Implementierung, sowie weitere Informationen über das Vorgehen im Projekt.

4.1 Integrationsstrategie

Im Team wurde für die Implementierung des Projekts die outside-in-Integrationsstrategie angewandt. Diese bot sich an, da es zwei code-intensive Projektbausteine gab, welche zunächst abgesehen von entworfenen Schnittstellen unabhängig voneinander Funktionalität erhalten mussten. Diese Bausteine waren der View und die Minecraft-Interaktion sowie Controller und Model. Im Verlauf der Implementierung wurde dann die Interaktion der beiden Bausteine zusammengeführt. Nachdem so alle Muss-Kriterien und ein Teil der Soll-Kriterien implementiert waren, wurde anschließend funktionsorientiert die Funktionalität für fehlende Soll-Kriterien und die ersten Kann-Kriterien erarbeitet.

4.2 Zeitlicher Verlauf

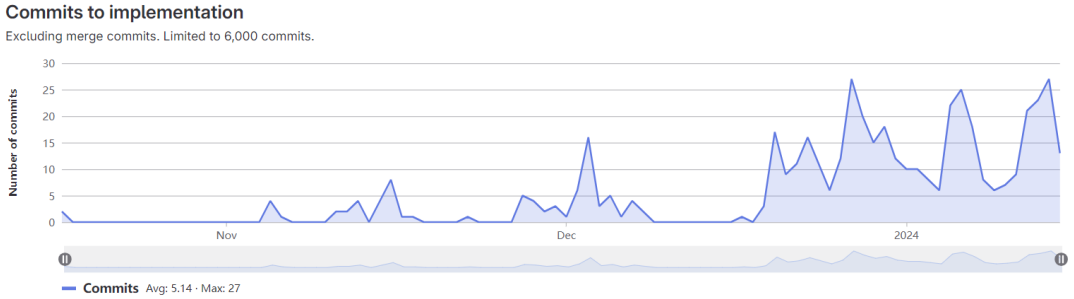

Abbildung 4.1: Commit-Verlauf auf dem Implementierungsbranch

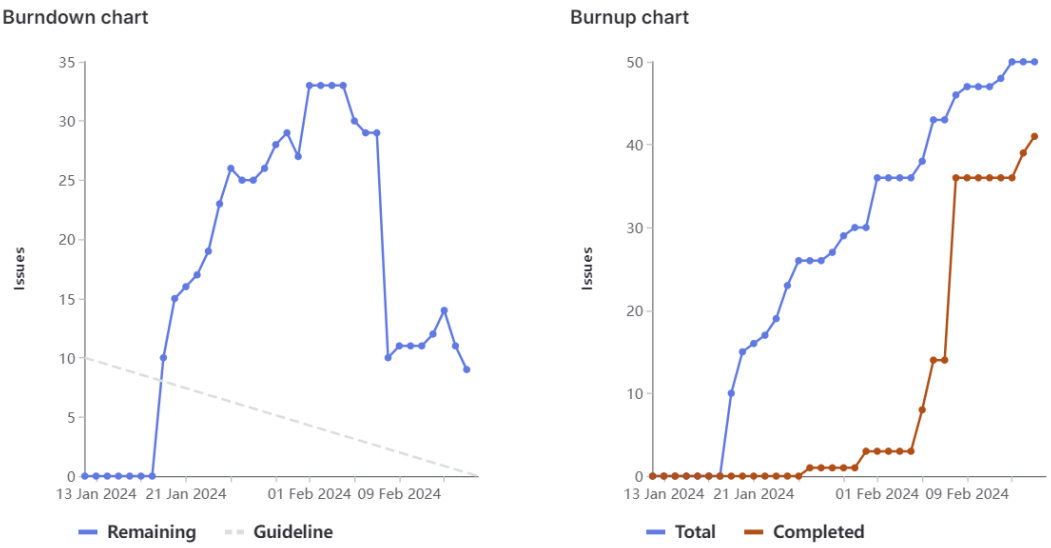

Abbildung 4.2: Burn-Up-Chart und Burn-Down-Chart für die Muss- und Soll-Kriterien

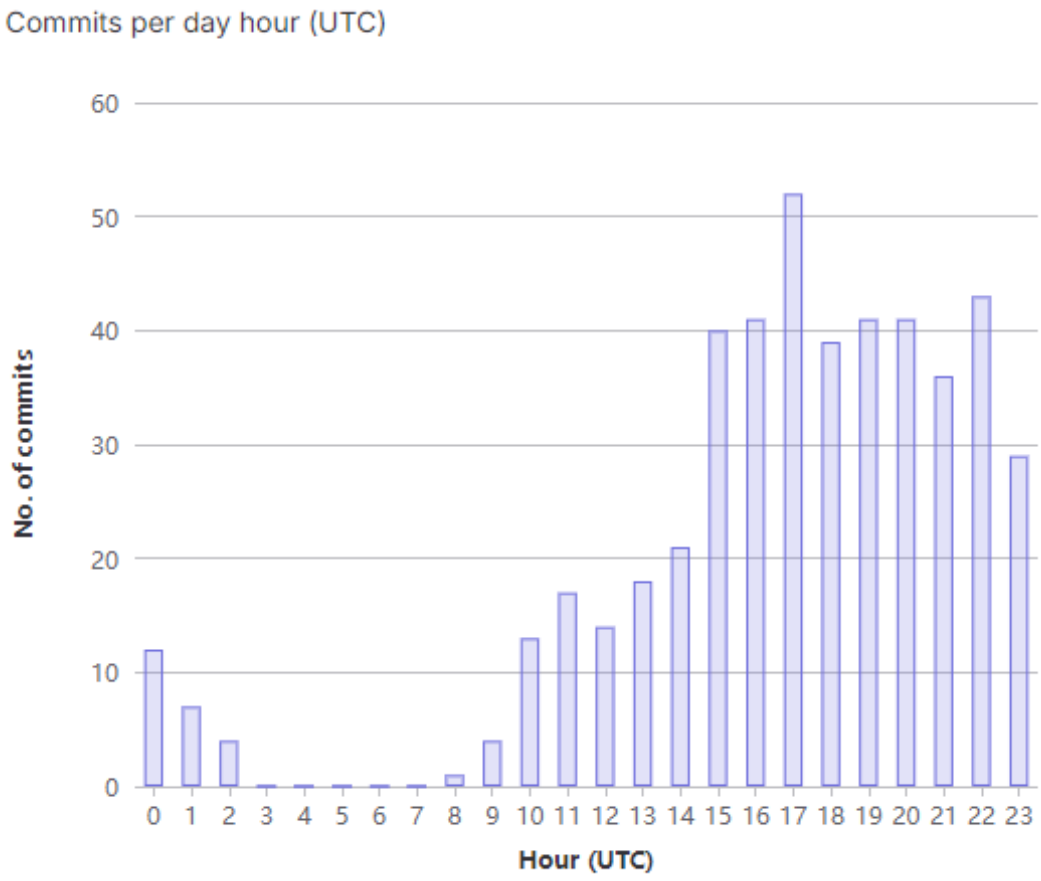

Abbildung 4.3: Commit-Verteilung nach Tageszeit

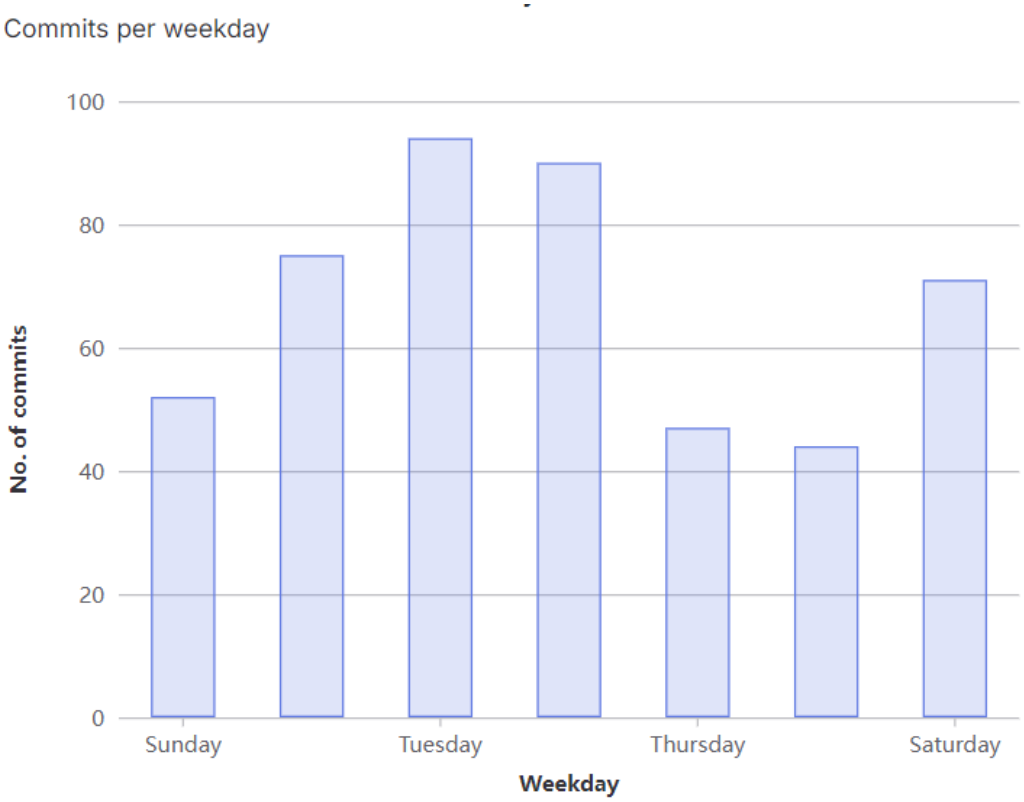

Abbildung 4.4: Commit-Verteilung nach Wochentag

5 Benutzte Tools

- **Discord** Hauptsächliches Kommunikationsmittel für schnelle Besprechungen.
- **GitLab** Git-Code-Speicherung und Issue-Management.
- **JetBrains Code With Me** Für remote Pair-Programming bei komplizierten Codestellen.
- **JaCoCo** Generierung von Test-Code-Coverage Berichten.
- **Mockito** Vor allem auf der Minecraft-Seite ist es einfacher, viel zu mocken.
- **GitLab CI-CD** Sehr nützlich, um sofort mitzubekommen, ob Tests durch Änderungen nicht mehr funktionieren.
- **SonarLint/SonarQube** Statische Code-Analyse, um kritische Stellen schnell zu sehen und beheben zu können.
- **GitHub Copilot** Hat repetitive Arbeit übernommen und damit die Effizienz gesteigert.
- **Gradle** Natives Build-System von Fabric. Hat auch für unsere Zwecke gut funktioniert.
- **Blockbench** Tool um Pixelgrafiken zu erstellen.
- **Gimp** Um Grafiken, Logos, etc. zu bearbeiten.
- **Paint** Um Grafiken, Logos, etc. zu bearbeiten.

6 Beispiel

Im Folgenden wird ein Beispielbefehl aus dem MIMA-Befehlssatz ausgeführt. Takt 0 zeigt den Startzustand des Computers, wie er in Abbildung 6.1 zu sehen ist. Zudem sind in derselben Abbildung das Programm in Register-Transfer-Schreibweise und der schematische Aufbau des Computers zu sehen. Takt 1-5 stellen die Hol-Phase dar. Takt 6 wird in der für das Beispiel genutzten Version des MIMA-Befehlssatzes übersprungen, da die Decodierung des Befehls intern passiert. Takt 7-12 zeigt dann die Ausführung des Befehls.

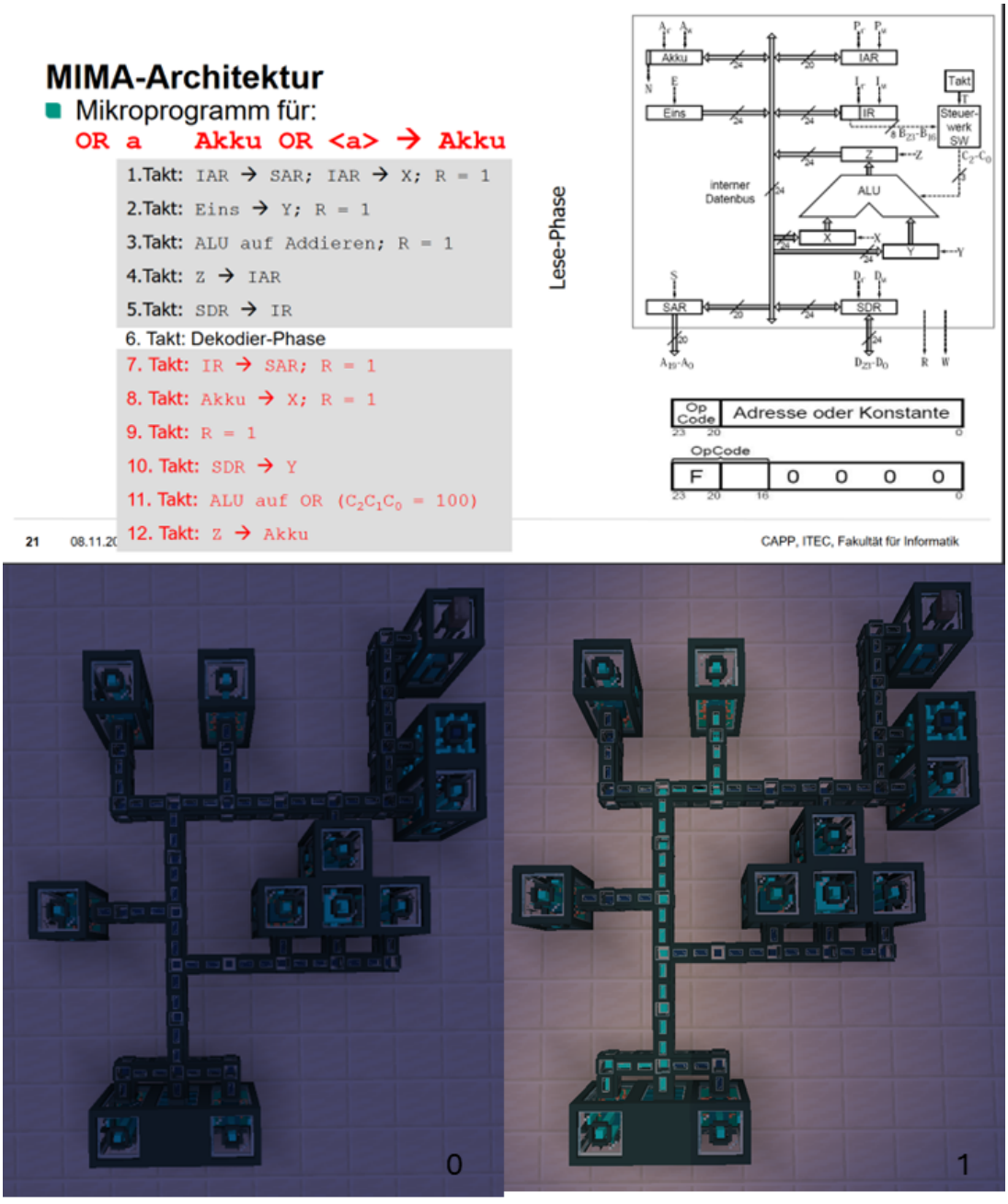

Abbildung 6.1: Takt 0 und 1

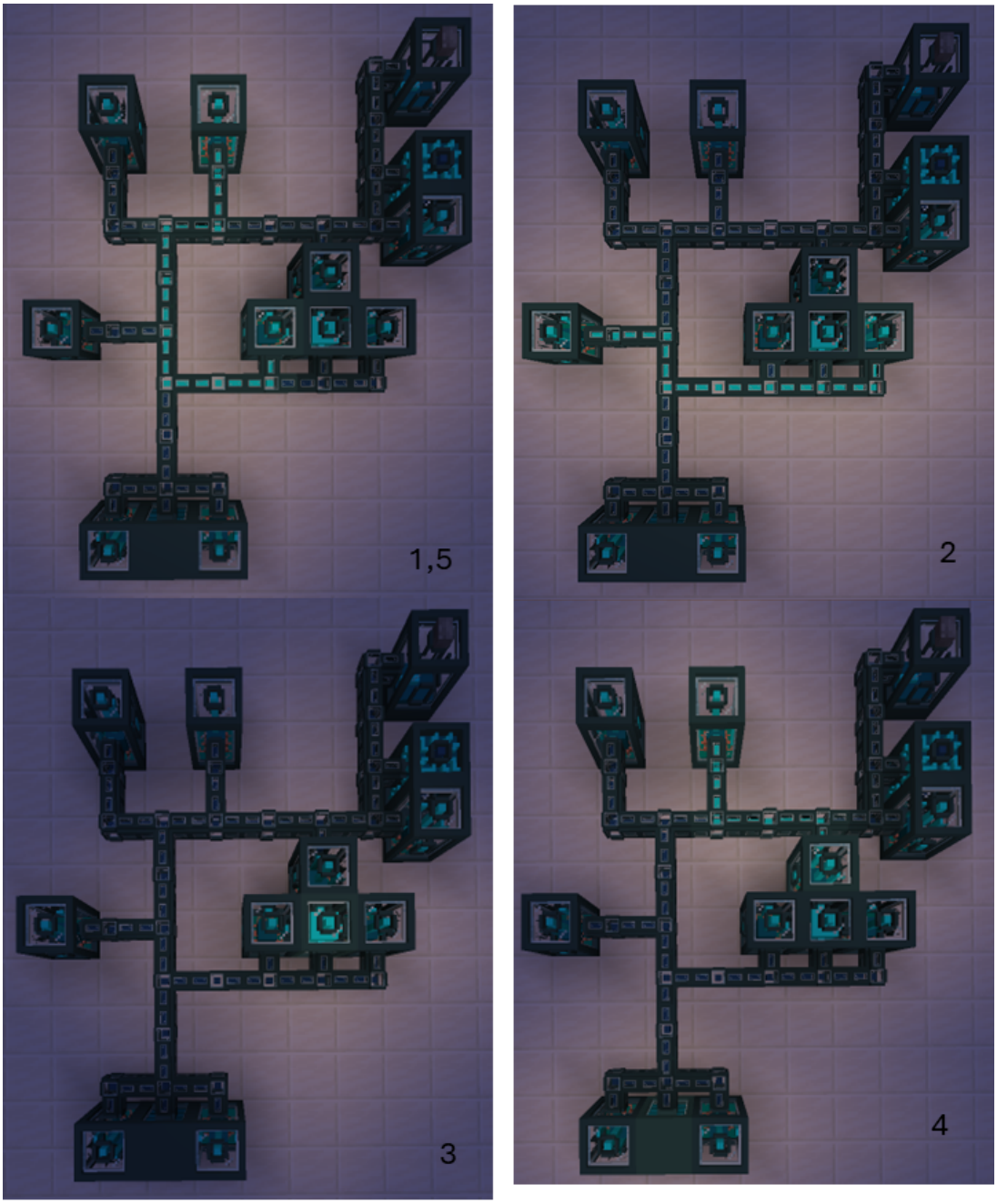

Abbildung 6.2: Takt 1-4

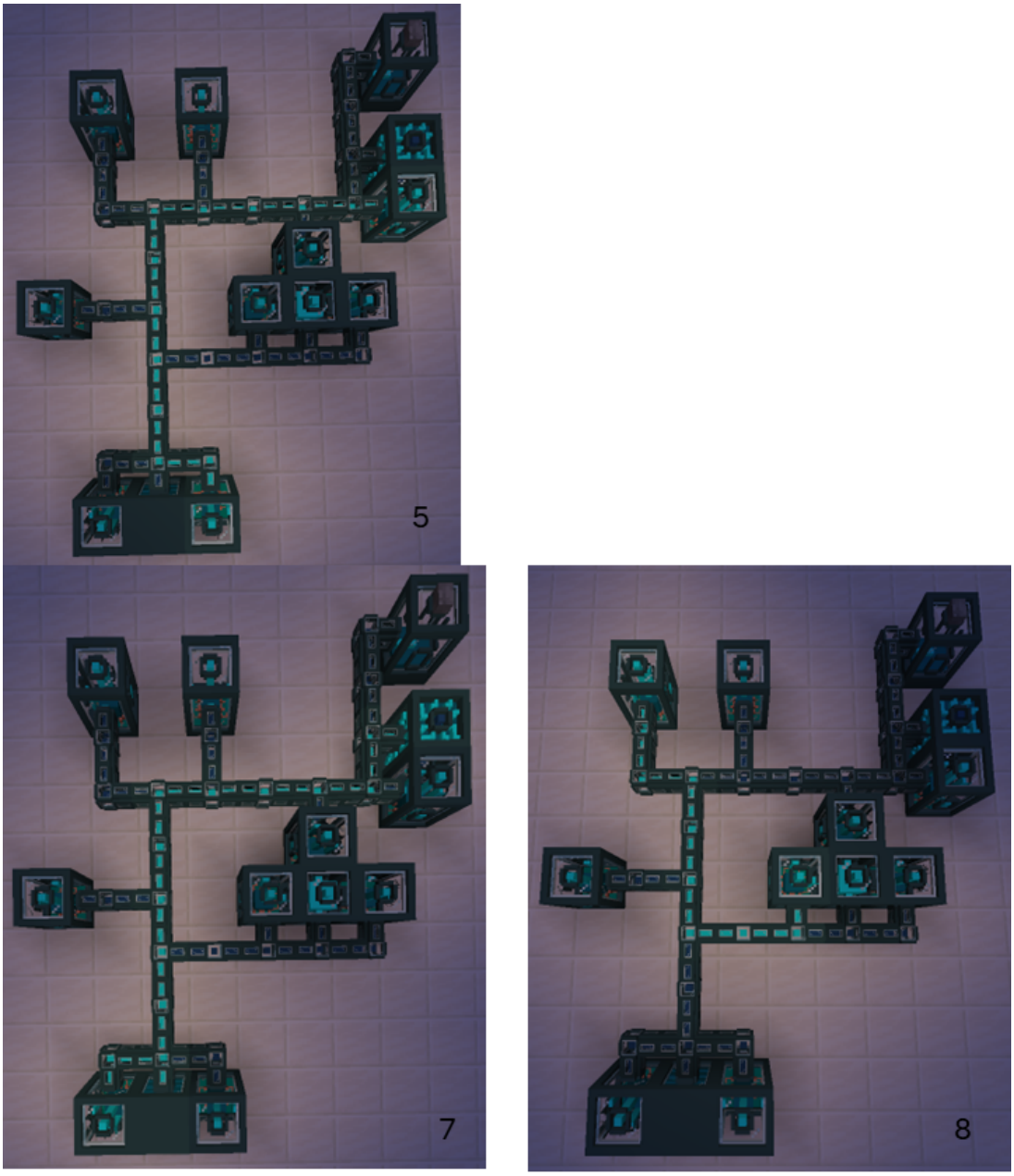

Abbildung 6.3: Takt 5-8

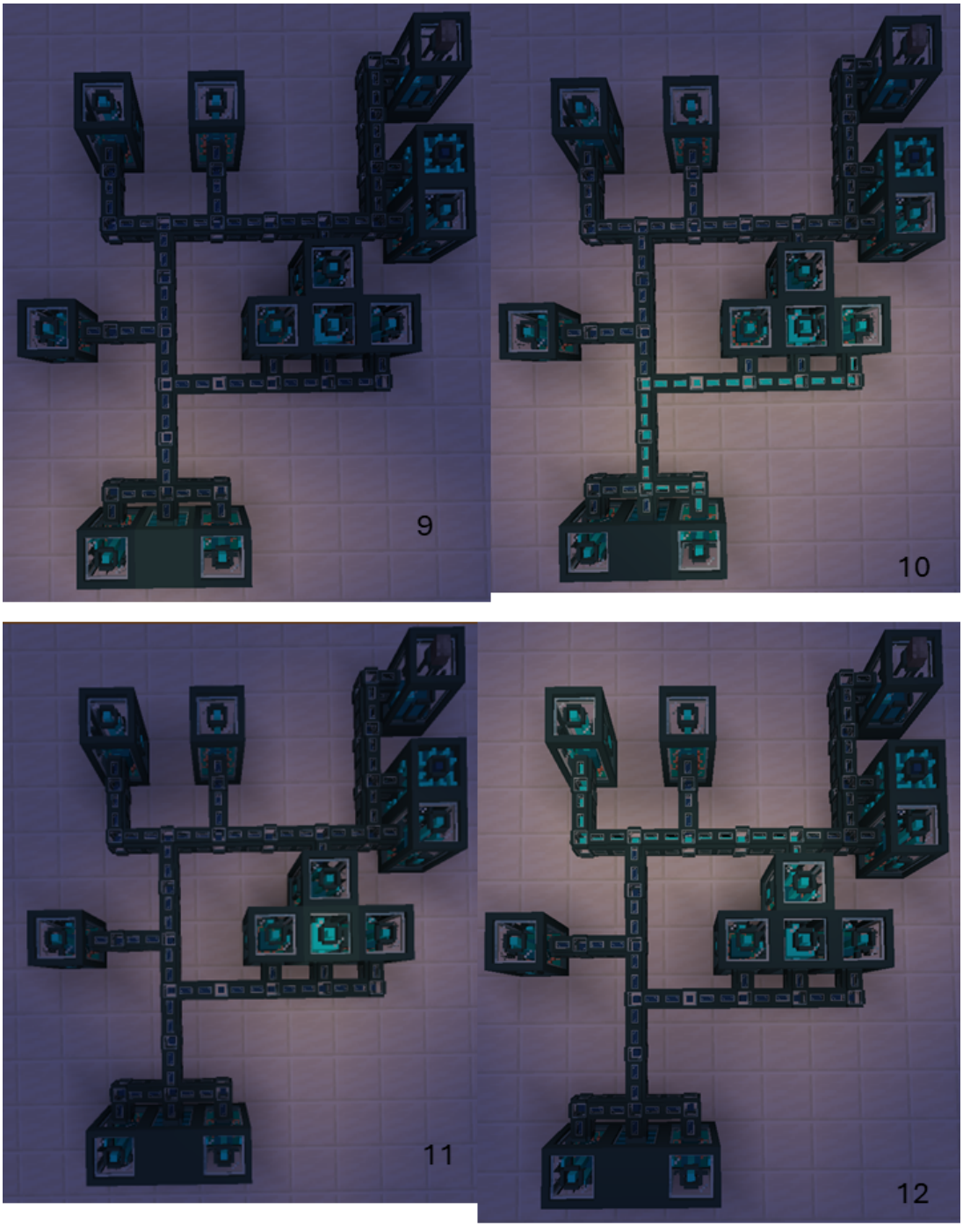

Abbildung 6.4: Takt 9-12

7 Glossar

Assembler: Ein Assembler ist ein Computerprogramm, das Assemblersprachcode in Maschinencode übersetzt. Assemblersprache ist eine Low-Level-Programmiersprache, die eng mit dem Maschinencode verwandt ist, aber für Menschen leichter zu lesen und zu schreiben ist. [1]

Befehlssatz:

Der Befehlssatz eines Prozessors ist in der Rechnerarchitektur die Menge der Maschinenbefehle, die ein bestimmter Prozessor ausführen kann. Je nach Prozessor variiert der Umfang des Befehlssatzes zwischen beispielsweise 33 und über 500 Befehlen. CISC-Prozessoren haben tendenziell größere Befehlssätze als RISC-Prozessoren. [2]

Block:

Ein Block ist ein besonderer Gegenstand, den man in der Minecraft-Welt platzieren kann. Nahezu die gesamte Spielwelt besteht aus Blöcken. [3]

Drop:

Der Begriff Drop bezeichnet fallen gelassene (vom engl. dropped) Objekte, die in der Spielwelt auf dem Boden schweben und aufgenommen werden können. Als Drop sind sie keine Gegenstände, sondern Objekte, denn sie bewegen sich. [4]

Inventar:

Im Überlebensmodus ist das Inventar der Speicher des Spielers für Blöcke und sonstige Gegenstände, sozusagen ein Rucksack, den man immer bei sich trägt. Das Inventar hat ein begrenztes Fassungsvermögen. Im Kreativmodus ist das Inventar kein Speicher, sondern eine Auswahl fast aller Blöcke und sonstiger Gegenstände in unbegrenzter Menge. [5]

[1] Quelle: https://techwatch.de/blog/understanding-the-basics-what-is-an-assembler-and-how-does-it-work/
[2] Quelle: https://de.wikipedia.org/wiki/Befehlssatz
[3] Vgl.: https://minecraft.fandom.com/de/wiki/Block
[4] Quelle: https://minecraft.fandom.com/de/wiki/Drop
[5] Quelle: https://minecraft.fandom.com/de/wiki/Inventar

Item:

Ein Gegenstand (engl. Item) ist alles, was man in sein Inventar aufnehmen und in der Hand halten kann. Ein Block ist ein besonderer Gegenstand, den man in der Welt platzieren kann (z.B. Erde oder eine Werkbank). Daneben gibt es noch einige Gegenstände, die beim Platzieren zu einem beweglichen Objekt werden, z.B. ein Boot. [6]

MIMA:

Die mikroprogrammierte Minimalmaschine (MIMA) ist ein Lehrmodell zur vereinfachten Darstellung von Mikroprozessoren, basierend auf der Von-Neumann-Architektur, welche von Tamim Asfour am Karlsruher Institut für Technologie entwickelt wurde. [7]

Minecraft-Objekte:

Objekte (engl. Entities) sind neben den Gegenständen, zu denen auch die Blöcke gehören, die andere große Gruppe der Spielelemente in Minecraft. Eine Minecraft-Welt besteht aus Blöcken und Objekten. Im Gegensatz zu den Blöcken sind die Objekte meist beweglich (z.B. eine Lore oder Projektile). Zu den Objekten zählen auch jegliche Drops. [8]

Mod:

Als Modifikation (Abkürzung Mod) wird alles bezeichnet, das den Spielinhalt von Minecraft verändert. [9]

Redstone:

Redstone ist ein flacher, transparenter Block, der Redstone-Signale übertragen kann. Ein Signal geht von einem Signalblock über den Redstone zu einem Empfängerblock und löst dort eine Aktion aus.

Rezepte:

Rezepte geben an, mit welchen Gegenständen ein Block oder Gegenstand in Minecraft hergestellt werden kann.

RISC-V:

RISC-V ist eine Befehlssatzarchitektur, die sich auf das Designprinzip des Reduced Instruction Set Computers (RISC) stützt. Das Designziel von RISC ist der Verzicht auf einen komplexen Befehlssatz hin zu einfach zu dekodierenden und schnell auszuführenden Befehlen. [10]

[6] Quelle: https://minecraft.fandom.com/de/wiki/Gegenstand
[7] Vgl.: https://de.wikipedia.org/wiki/Mikroprogrammierte_Minimalmaschine
[8] Vgl.: https://minecraft.fandom.com/de/wiki/Objekt
[9] Quelle: https://minecraft.fandom.com/de/wiki/Modifikation
[10] vgl.: https://de.wikipedia.org/wiki/RISC-V und https://de.wikipedia.org/wiki/Reduced_Instruction_Set_Computer

Slot:

Ein Slot in Minecraft ist ein Platz in der GUI, an den Items gelegt werden können. Das Inventar und Kisten bestehen aus Slots.