HAW HAMBURG

Masterarbeit

Wie effektiv ist der für das Projekt „Learning By Playing“ entwickelte Prototyp für den Erwerb grundlegender Programmierkonzepte?

vorgelegt am 30. September 2025

Jonas Mack

Erstprüfer: Prof. Ralf Hebecker
Zweitprüferin: Charlotte Knorr

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Department Medientechnik
Finkenau 35
20081 Hamburg

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Entwicklung und Evaluation eines spielbasierten Lernprototyps zur Vermittlung grundlegender Programmierkenntnisse („Learning by Playing“). Ziel der Studie ist die Prüfung der Wirksamkeit des Prototyps im Hinblick auf Lernfortschritte und die Evaluation der Qualität des eingesetzten Wissenstests.

Die Untersuchung wurde als Pilotstudie mit Schüler:innen und Studierenden durchgeführt. Zur Datenerhebung wurden Pre- und Post-Tests eingesetzt, die Wissen über grundlegende Programmierkonzepte abfragen. Es wurde eine Hypothese zur Leistungssteigerung formuliert und eine Prüfung der Reliabilität der Tests sowie eine Analyse der Itemschwierigkeit durchgeführt. Die Ergebnisse zeigen keine generelle Leistungssteigerung, allerdings weisen die Post-Test-Ergebnisse eine höhere interne Konsistenz und verbesserte Itemkennwerte auf.

Die Ergebnisse liefern wichtige Hinweise für die Optimierung des Lernprototyps und der Testinstrumente. Trotz methodischer Limitationen zeigt die Studie das Potenzial spielbasierter Ansätze zur Förderung grundlegender Programmierkenntnisse.

Abstract

This thesis focuses on the development and evaluation of a game-based learning prototype for teaching fundamental programming skills (“Learning by Playing”). The study aimed to assess the effectiveness of the prototype and the quality of the knowledge test.

The pilot study was conducted with secondary school and university students using pre- and post-tests measuring basic programming knowledge. A hypothesis was formulated regarding performance improvement and test reliability as well as item difficulty were analysed. Results showed no general improvement in performance, but post-test items demonstrated higher internal consistency and improved item properties.

The findings provide valuable guidance for optimizing the learning prototype and assessment instruments. Despite methodological limitations, the study highlights the potential of game-based approaches for fostering foundational programming skills.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In einer zunehmend digitalisierten Welt gewinnen Programmierkenntnisse immer stärker an Bedeutung – nicht nur im IT-Sektor, sondern auch in zahlreichen anderen Berufs- und Lebensbereichen. Bereits 2020 forderte die EU-Kommission eine breitere Integration digitaler Kompetenzen, insbesondere Computational Thinking, in den schulischen Bildungssektor (European Commission, 2020). Gleichzeitig zeigt sich jedoch, dass der Erwerb von Computational-Thinking-Fähigkeiten für viele Lernende nach wie vor eine erhebliche Hürde darstellt. Trotz gewisser Verbesserungen in den letzten Jahren liegen die durchschnittlichen Abbruchquoten in einführenden Informatikkursen (z. B. CS1) weiterhin häufig um die 30 % (Bennedsen & Caspersen, 2019), während Massive Open Online Courses (MOOCs) einen Median von lediglich 6,5 % an abgeschlossenen Kursen aufwiesen (Jordan, 2014). Neben den hohen Abbruchraten werden in der Forschung insbesondere kognitive Überforderung, fehlende Vorerfahrungen und mangelnde Motivation als zentrale Herausforderungen hervorgehoben (The University of Western Australia & Cardell-Oliver, 2014). Diese Problematik verdeutlicht den Bedarf an neuen, motivierenden Lernformaten, die insbesondere Einsteiger:innen ohne Vorkenntnisse einen niedrigschwelligen und unterstützenden Zugang zu Programmierkonzepten ermöglichen.

Ein vielversprechender Ansatz zur Förderung von Lernmotivation und nachhaltigem Verständnis ist der Einsatz von spielerischen Lernumgebungen im Rahmen des Game-based Learning. Verschiedene Studien konnten bereits nachweisen, dass Lernspiele in der Lage sind, sowohl intrinsische Motivation als auch langfristiges Verständnis von Inhalten zu steigern (Hamari et al., 2014; Sailer et al., 2013). Gerade bei komplexen Themen wie dem Programmieren kann die Verbindung aus Interaktion, Feedback und Exploration das Lernen effektiver gestalten. Vor diesem Hintergrund entstand im Rahmen des Projekts *Learning By Playing* die Idee, ein Lernspiel zu entwickeln, das grundlegende Programmierkonzepte spielerisch vermittelt.

1.1 Forschungsfrage

Im Zentrum dieser Arbeit steht die Untersuchung eines Prototyps, der im Rahmen des Projekts *Learning by Playing* entwickelt wurde. Ziel des Projekts ist die Konzeption eines Lernspiels aus dem Genre der sogenannten Automatisierungsspiele. In dem Prototyp steuern die Spielenden virtuelle Einheiten mithilfe blockbasierter Programmieranweisungen (z. B. *if*-Bedingungen oder *while*-Schleifen), um Aufgaben innerhalb der Spielwelt zu lösen (z. B. die Bewegung einer programmierbaren Einheit zu einer bestimmten Position). Auf diese Weise sollen grundlegende Prinzipien des Programmierens, wie zum Beispiel Bedingungslogik, Wiederholungsstrukturen und allgemeines algorithmisches Denken, intuitiv und kontextbezogen erlernt werden.

Der entwickelte Prototyp dient dabei als exemplarische Umsetzung des Konzepts mit dem Ziel, das pädagogische und spieltechnische Potenzial aufzuzeigen, um zukünftig weitere Fördermöglichkeiten für eine langfristige Weiterentwicklung des Spielkonzepts einzubringen.

Diese Arbeit untersucht, inwiefern der Prototyp dafür geeignet ist, grundlegende Programmierkonzepte verständlich und nachhaltig zu vermitteln. Dabei besteht die primäre Zielgruppe aus Schüler:innen der Oberstufe mit wenig bis keinen Vorkenntnissen im Programmieren. Daraus ergibt sich die folgende Forschungsfrage: Wie effektiv ist der für das Projekt *Learning By Playing* entwickelte Prototyp für den Erwerb grundlegender Programmierkonzepte?

1.2 Gliederung der Arbeit

Zur Beantwortung der Forschungsfrage wird zunächst ein grundlegendes Verständnis theoretischer Konzepte wie Gamification und spielbasiertem Lernen erarbeitet. Darauf aufbauend wird der aktuelle Forschungsstand zu Programmierspielen und gamifizierten Lernumgebungen beleuchtet. Anschließend wird der entwickelte Prototyp im Detail vorgestellt und das methodische Vorgehen zur Evaluation seiner Wirksamkeit beschrieben. Die gewonnenen Daten werden schließlich analysiert und im Hinblick auf die Forschungsfrage diskutiert.

2 Theoretischer Hintergrund

Im Rahmen der Entwicklung eines Lernspiels zur Förderung von Programmierkompetenzen ist es zunächst erforderlich, verschiedene zentrale Begriffe zu erläutern. Ziel dieses Kapitels ist es, eine fundierte theoretische Grundlage zu schaffen, um die Gestaltung des Spiels besser nachvollziehen zu können und eine Basis für die spätere Evaluation und kritische Analyse des Spiels zu legen.

Dabei werden zentrale Konzepte aus den Bereichen der Informatikdidaktik, der Lernpsychologie sowie der spielbasierten Pädagogik herangezogen, insbesondere die Konzepte des Game-based Learning, des Playful Learning Frameworks und der Gamification, die eng mit den Zielen des Prototyps verknüpft sind.

2.1 Lernen, Spielen und Programmierlernen

Um die theoretische Grundlage der Lernspiele einordnen zu können, muss zunächst die in dieser Arbeit verwendete Definition zentraler Begriffe geklärt werden. Lernen wird im Rahmen dieser Arbeit aus konstruktivistischer Perspektive verstanden. Wissen entsteht durch aktive Auseinandersetzung mit Problemen, durch Reflexion und durch soziale Interaktion (Piaget, 1970; Vygotsky, 1978). Dabei spielt Feedback eine zentrale Rolle, insbesondere wenn Lernende ihre Hypothesen direkt am Lerngegenstand überprüfen können (Mayer et al., 2004). Der konstruktivistische Ansatz wird in einem der folgenden Abschnitte im Detail erläutert. Spielen wird nicht nur als Freizeitaktivität, sondern als bedeutungsvoller kultureller und kognitiver Prozess verstanden, der exploratives Verhalten, intrinsische Motivation und kreatives Problemlösen begünstigt (Gee, 2003; Huizinga, 1987) . In pädagogischen Kontexten wird Spielen zunehmend als ernstzunehmender Lernkontext betrachtet, da es dynamisches Lernen durch Handeln ermöglicht. Programmieren lernen bezeichnet den Erwerb von Fähigkeiten und Konzepten, die erforderlich sind, um algorithmische Probleme zu lösen, Programme zu verstehen und zu erstellen. Neben Syntax und Semantik sind auch Denkprozesse wie Abstraktion, Zerlegung und Mustererkennung von Bedeutung (Grover & Pea, 2013; Wing, 2006). Dieses Verständnis steht in engem Zusammenhang mit dem Konzept des Computational Thinking.

2.2 Computational Thinking

Der Begriff Computational Thinking (CT) wurde von Wing (2006) als eine zentrale Fähigkeit beschrieben, die über die reine Programmierpraxis hinausgeht und grundlegende Denkweisen zur Problemlösung umfasst. Darunter fallen insbesondere die Zerlegung komplexer Aufgaben in Teilprobleme, die Abstraktion relevanter Strukturen, die Erkennung von Mustern sowie die Entwicklung und Evalua-

tion von Algorithmen. Diese Fähigkeiten gelten nicht nur als Grundlage für das Programmieren, sondern auch als allgemeine Problemlösestrategien, die nicht nur in der Informatik, sondern auch in Bereichen wie in der Mathematik und im Ingenieurwesen Anwendung finden (Grover & Pea, 2013).

In Hinsicht auf die Informatikdidaktik bedeutet dies, dass das Erlernen von Programmieren nicht nur technisches Verständnis beinhaltet. Vielmehr soll der Fokus darauf liegen, durch geeignete Lernumgebungen die Denkweisen zu fördern, die Lernenden einen nachhaltigen Zugang zu Problemlösungsfähigkeiten eröffnen. Spielbasierte Ansätze werden in diesem Zusammenhang immer häufiger diskutiert, da sie exploratives Problemlösen, Feedbackschleifen und intrinsische Motivation vereinen.

2.3 Grundlegende Programmierkonzepte

Die Vermittlung von Programmierkenntnissen setzt das Verständnis grundlegender Konzepte voraus, die als Bausteine algorithmischen Denkens gelten. Dazu zählen insbesondere: Sequenz, Verzweigung (Konditionalstrukturen wie *if*-Anweisungen), Wiederholung (z. B. *while*- oder *for*-Schleifen), Rückgabewerte sowie die Nutzung von Variablen zur Speicherung und Manipulation von Werten.

Diese Konzepte bilden das Fundament nahezu jeder Programmiersprache und sind in der Informatikdidaktik fest verankert (Mühling et al., 2015). Sie ermöglichen es Lernenden, Probleme systematisch zu analysieren und algorithmisch zu lösen. Nach Sentance und Waite (2017) haben gerade Anfänger:innen in der Informatik häufig Schwierigkeiten beim Verständnis solcher abstrakten Konzepte, weshalb ein schrittweiser und kontextualisierter Einstieg empfohlen wird.

Im Folgenden werden die wichtigsten grundlegenden Programmierkonzepte erläutert, die auch in der Informatikdidaktik eine zentrale Rolle spielen:

Sequenz

Die Sequenz bezeichnet die lineare Abfolge von Befehlen oder Anweisungen in einem Programm. Jede Anweisung wird nacheinander ausgeführt, was den Fluss der Programmausführung bestimmt. Die Sequenz ist das einfachste und gleichzeitig grundlegendste Steuerungselement in der Programmierung.

Verzweigung (Konditionalstrukturen)

Verzweigungen ermöglichen es Programmen, Entscheidungen zu treffen und abhängig von Bedingungen unterschiedliche Abschnitte des Codes auszuführen. Typische Formen sind if-Anweisungen, die eine oder mehrere Bedingungen prüfen und basierend auf dem Wahrheitswert der Bedingungen bestimmte Anweisungen ausführen oder überspringen. Dadurch können Programme flexibel auf unterschiedliche Situationen reagieren.

Wiederholung (Schleifen)

Wiederholungen erlauben es, Programmteile mehrfach auszuführen, solange eine oder mehrere Bedingungen erfüllt sind. Es gibt verschiedene Schleifenarten, darunter üblicherweise:

- **while-Schleife (fußgesteuert):** Die Bedingung wird am Ende jeder Schleifendurchführung geprüft, sodass der Schleifenrumpf mindestens einmal ausgeführt wird.
- **while-Schleife (kopfgesteuert):** Die Bedingung wird zu Beginn jeder Schleifendurchführung geprüft, sodass der Schleifenrumpf nie ausgeführt wird, falls die Bedingung nicht erfüllt ist.
- **for-Schleife**: Häufig genutzt für eine definierte Anzahl an Wiederholungen, iteriert über einen Wertebereich.

Schleifen sind wichtig, um wiederholt auszuführende Aufgaben effizient zu lösen.

Funktionen/Methoden

Funktionen oder Methoden sind in sich geschlossene Programmeinheiten, die eine bestimmte Aufgabe erfüllen und bei Bedarf mehrfach im Programm aufgerufen werden können. Sie ermöglichen die Modularisierung des Codes, fördern die Wiederverwendbarkeit und verbessern die Übersichtlichkeit eines Programms. Funktionen können Eingabewerte (Parameter) entgegennehmen, Berechnungen durchführen und optional Rückgabewerte an den Aufrufer liefern. Durch die Verwendung von Funktionen lernen Programmierende, Probleme in kleinere Teilprobleme zu zerlegen und diese systematisch zu lösen, was einen zentralen Aspekt des algorithmischen Denkens darstellt.

Rückgabewerte

Rückgabewerte sind Werte, die eine Funktion oder Methode nach ihrer Ausführung an den Aufrufer zurückgibt. Sie ermöglichen es, berechnete Ergebnisse weiterzuverwenden und die Programmstruktur modular und übersichtlich zu gestalten.

Variablen

Variablen sind Speicherplätze für Daten, die während der Programmausführung verändert werden können. Sie dienen zur Speicherung von Eingabewerten, Zwischenergebnissen oder Zuständen und sind zentral für die Weiterverwendung und Manipulation von Informationen im Programm.

2.4 Game-based Learning

Game-based Learning bezeichnet den gezielten Einsatz von digitalen oder analogen Spielen als Lernumgebungen, um Wissen, Fertigkeiten und Kompetenzen zu vermitteln. Dabei werden Lerninhalte so in das Spiel integriert, dass Spieler:innen durch aktive Teilnahme am Spielprozess lernen. Die intrinsische Motivation und das Engagement, das Spiele typischerweise hervorrufen, helfen dabei, den Lernprozess zu unterstützen.

Zentrale Merkmale von Game-based Learning sind eine klare Zielorientierung, bei der Lernziele explizit oder implizit durch Spielaufgaben adressiert werden, sowie unmittelbares Feedback, das den Spieler:innen Rückmeldungen zu ihrem Lernfortschritt gibt und Lernanpassungen ermöglicht (Kiili, 2005). Eine der häufig im Kontext des Game-based Learnings genannte Theorie ist das Prinzip der „Flow"-Erzeugung, das heißt einer Balance zwischen Herausforderung und den Fähigkeiten der Lernenden, um optimale Lernzustände zu fördern (Abbildung 1). Steigende Komplexität und variable Schwierigkeitsgrade ermöglichen es, die Lernenden schrittweise und ihren Fähigkeiten entsprechend an anspruchsvollere Inhalte heranzuführen. So verfolgt auch der Prototyp das Ziel, Spieler:innen auf dieser Grenze zwischen Über- und Unterforderung zu führen, um Motivation zu erzeugen und Frustration zu vermeiden.

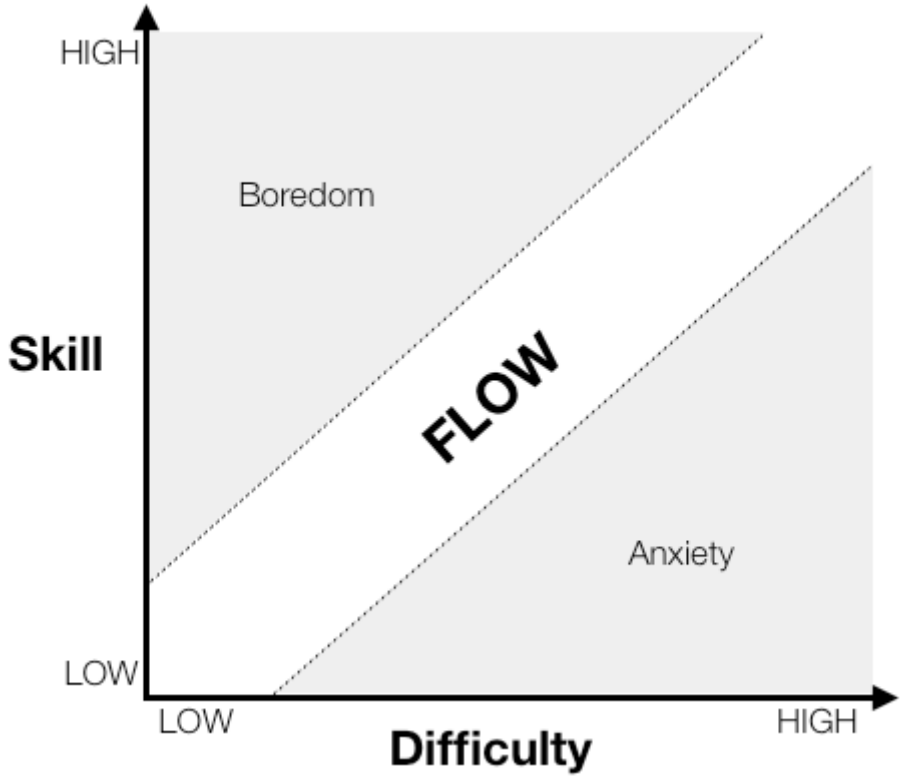


Abbildung 1: Modell des Flow-Zustands nach Csikszentmihalyi (1999) (Quelle: Ishitani, 2012)

Empirische Studien belegen die Wirksamkeit von Game-based Learning bei komplexen und kognitiv anspruchsvollen Themen, zum Beispiel in den Bereichen Mathematik und Biologie (Wouters et al., 2013). Die Einbettung von Lerninhalten in spielerische, handlungsorientierte Kontexte fördert das aktive Problemlösen, die Exploration und die nachhaltige Verankerung des Wissens. Dabei kann Game-based Learning nicht nur Wissen vermitteln, sondern auch kritisches Denken, Kreativität und kooperative Fähigkeiten stärken (Alotaibi, 2024).

2.5 Playful Learning

Playful Learning erweitert Game-based Learning um affektive und soziale Aspekte wie Spaß, Kreativität und Kooperation. Gleichzeitig sind die Grenzen zwischen beiden Ansätzen fließend, da beide auf ähnlichen theoretischen Grundlagen beruhen (Plass, Homer & Kinzer, 2014, 2015). Während Game-based Learning häufig stärker zielorientierte Aufgabenformate in den Vordergrund stellt, betont Playful Learning eher offene und explorative Aktivitäten.

Plass et al. verdeutlichen sowohl im Kontext von Playful Learning als auch von Game-based Learning anhand des Integrated Design Frameworks, dass nachhaltiges Lernen in spielerischen Umgebungen durch die Verbindung mehrerer Perspektiven ermöglicht wird:

Kognitive Perspektive: Informationsverarbeitung, Problemlösen und Wissenskonstruktion.

Motivationale Perspektive: Förderung intrinsischer und extrinsischer Motivation, Interesse und Zielorientierung.

Affektive Perspektive: Emotionen wie Freude, Neugier und Interesse, die Lernprozesse begünstigen.

Soziokulturelle Perspektive: Kooperation, Kommunikation und die Schaffung von gemeinsamem Wissen.

2.6 Gamification

Gamification bezeichnet den Einsatz spieltypischer Elemente und Mechaniken in nicht-spielerischen Kontexten, um Motivation, Engagement und Verhaltensänderungen zu fördern (Deterding et al., 2011). Im Bildungsbereich werden beispielsweise Punktesysteme, Levelaufstiege, Abzeichen oder Ranglisten in Lernumgebungen integriert, ohne dass notwendigerweise ein vollständiges Spiel entsteht. Dadurch sollen intrinsische und extrinsische Motivationsfaktoren angesprochen werden, um die Lernenden stärker zu aktivieren und die Lernziele effektiver zu erreichen (Hamari et al., 2014). Während Game-based Learning den Fokus auf das Lernen durch das Spielen selbst legt, liegt der Fokus der Gamification vor allem darauf, spielerische Elemente in Lernumgebungen einzubetten und so spielerische Erfahrungen und Lernerfolg miteinander zu verbinden.

2.7 Cognitive Load Theory

Ein weiterer zentraler Aspekt beim Erlernen von Programmierkonzepten ist die kognitive Belastung, die während des Lernprozesses auftritt. Die Cognitive Load Theory nach Sweller et al. (1998) beschreibt, dass Lernprozesse durch die begrenzte Kapazität des Arbeitsgedächtnisses beeinflusst werden. Besonders beim Programmieren lernen kann eine Überlastung schnell entstehen, da Lernende gleichzeitig mit abstrakten Konzepten, komplexer Syntax, Fehlersuche und Problemlösestrategien konfrontiert sind. Dies führt dazu, dass die eigentliche inhaltliche Auseinandersetzung mit den grundlegenden Konzepten

wie Sequenz, Verzweigung oder Wiederholung durch die kognitive Beanspruchung erschwert wird. Didaktische Ansätze, die darauf abzielen, diese Belastung zu reduzieren, schaffen daher bessere Voraussetzungen für nachhaltiges Lernen.

2.8 Konstruktivismus im Kontext von Gamification und Lernen

Der Konstruktivismus ist eine Lerntheorie, die davon ausgeht, dass Wissen nicht einfach vermittelt, sondern von Lernenden aktiv konstruiert wird (Piaget, 1970; Vygotsky, 1978). Lernen wird als individueller Prozess verstanden, bei dem Lernende neue Informationen mit ihrem bestehenden Wissen verknüpfen und dadurch eigene Bedeutungen und Zusammenhänge konstruieren.

Im Kontext von Gamification und spielbasiertem Lernen wird der konstruktivistische Ansatz vor allem dadurch relevant, dass Spiele und spielerische Lernumgebungen den Lernenden ermöglichen, aktiv und selbstgesteuert zu explorieren, Probleme zu lösen und ihre Kenntnisse in sinnvollen Kontexten anzuwenden. Durch das Feedback und die Interaktionen im Spiel werden Lernprozesse gefördert, die auf Erfahrung, Entdeckung und sozialer Interaktion basieren (Gee, 2003).

Konstruktivistische Lernumgebungen unterstützen das autonome Lernen und fördern Motivation, da Lernende ihre eigenen Lernwege wählen und durch unmittelbare Rückmeldungen zu Erfolgen und Misserfolgen im Spiel Zusammenhänge erkennen können und Bedeutungssysteme konstruieren können.

2.9 Self-Determination Theory

Ein zentraler Aspekt im Kontext von Gamification und spielbasiertem Lernen ist die Motivation der Lernenden. Die Self-Determination Theory von Deci und Ryan (2000) liefert hierfür einen wichtigen theoretischen Rahmen. Sie geht davon aus, dass intrinsische Motivation dann entsteht, wenn drei grundlegende psychologische Bedürfnisse erfüllt werden: Kompetenz, Autonomie und soziale Eingebundenheit. Lernumgebungen, die diesen Bedürfnissen Rechnung tragen, fördern nicht nur kurzfristiges Engagement, sondern auch nachhaltiges Lernen.

Gamification und Game-based Learning können genau hier ansetzen, indem sie durch klare Rückmeldungen und ansteigende Schwierigkeitsgrade das Erleben von Kompetenz ermöglichen, durch Entscheidungsfreiheiten Autonomie unterstützen und durch kollaborative Spielmechaniken soziale Eingebundenheit stärken. Verschiedene Studien zeigen, dass Lernende motivierter und ausdauernder arbeiten, wenn diese Bedingungen erfüllt sind (Li et al., 2024; Zainuddin, 2018).

2.10 Blockbasierte Programmierung

Blockbasierte Programmierumgebungen wie Scratch (MIT Media Lab, 2007) (Abbildung 2) oder Blockly (Google, 2012) bieten einen intuitiven Zugang zu Programmierkonzepten für Anfänger:innen.

Durch das visuelle Kombinieren von Codebausteinen entfallen syntaktische Hürden, wodurch der Fokus auf das logische Denken gelenkt wird. Studien zeigen, dass diese Form der Programmierung sehr gut geeignet ist, um erste Programmiererfahrungen zu sammeln (Bau et al., 2017).

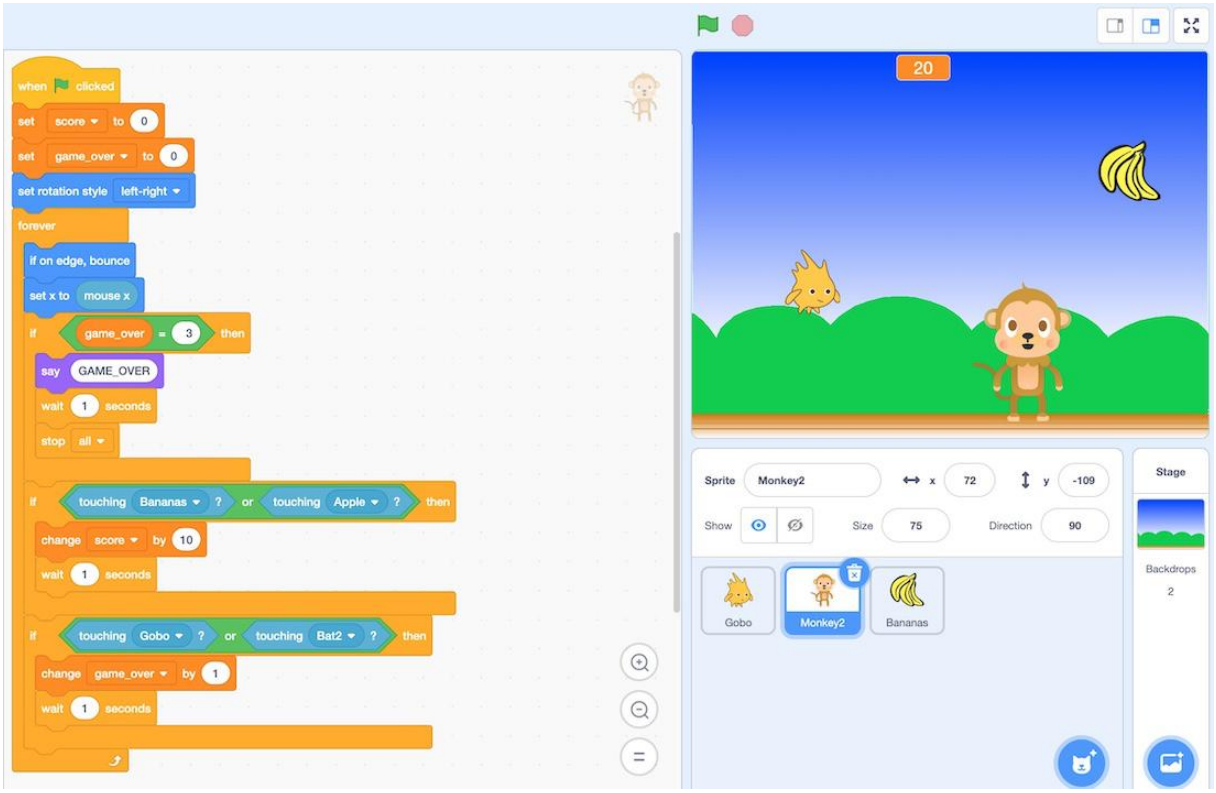


Abbildung 2: Screenshot aus Scratch, Quelle: GeeksforGeeks, 2023, abgerufen am 09.09.2025, von https://www.geeksforgeeks.org/computer-science-fundamentals/how-to-create-a-game-in-scratch-step-by-step-tutorial-for-begineers/

2.11 Automatisierungsspiele

Automatisierungsspiele sind eine Spielkategorie, bei der der Spielprozess stark auf der Planung, Programmierung oder Steuerung von automatisierten Abläufen beruht. Typischerweise geht es darum, Ressourcen systematisch zu sammeln, Prozesse zu optimieren und Produktionsketten aufzubauen, die weitgehend selbstständig laufen, während die Spielenden strategisch eingreifen, um Effizienz und Fortschritt zu steigern. Diese Spiele zeichnen sich durch einen wiederkehrenden Gameloop aus, in der Spieler:innen Programme oder Regeln definieren, die Einheiten oder Mechanismen steuern, um bestimmte Ziele zu erreichen, wie z. B. den Ausbau einer Basis oder die Automatisierung von Arbeitsschritten. Daraufhin werden neue Einheiten oder Mechanismen freigeschaltet, die eine Steigerung der Effizienz oder das Erreichen neuer Ziele ermöglichen und der Gameloop wird wiederholt.

Im Kontext von Lernspielen bietet sich das Genre der Automatisierungsspiele an, da diese von Spielenden verlangen, systematisch zu denken, komplexe Abläufe zu aufzubauen und Fehler darin zu beheben.

Dies sind Fähigkeiten, die eng mit Computational Thinking Fähigkeiten zusammenhängt. Die spielerische Umgebung ermöglicht es den Lernenden, abstrakte Logikkonzepte praktisch anzuwenden und zu experimentieren, ohne die Angst vor Fehlern oder Misserfolg, da diese Spiele häufig keine klassischen "Game Over"-Situationen bieten, wie z. B. Satisfactory (Coffee Stain Studios, 2024) oder Human Resource Machine (Tomorrow Corporation, 2015). Automatisierungsspiele können so als Brücke zwischen theoretischem Wissen und praktischer Anwendung dienen und eignen sich gut für die Förderung von algorithmischem Denken in einem motivierenden, spielerischen Rahmen.

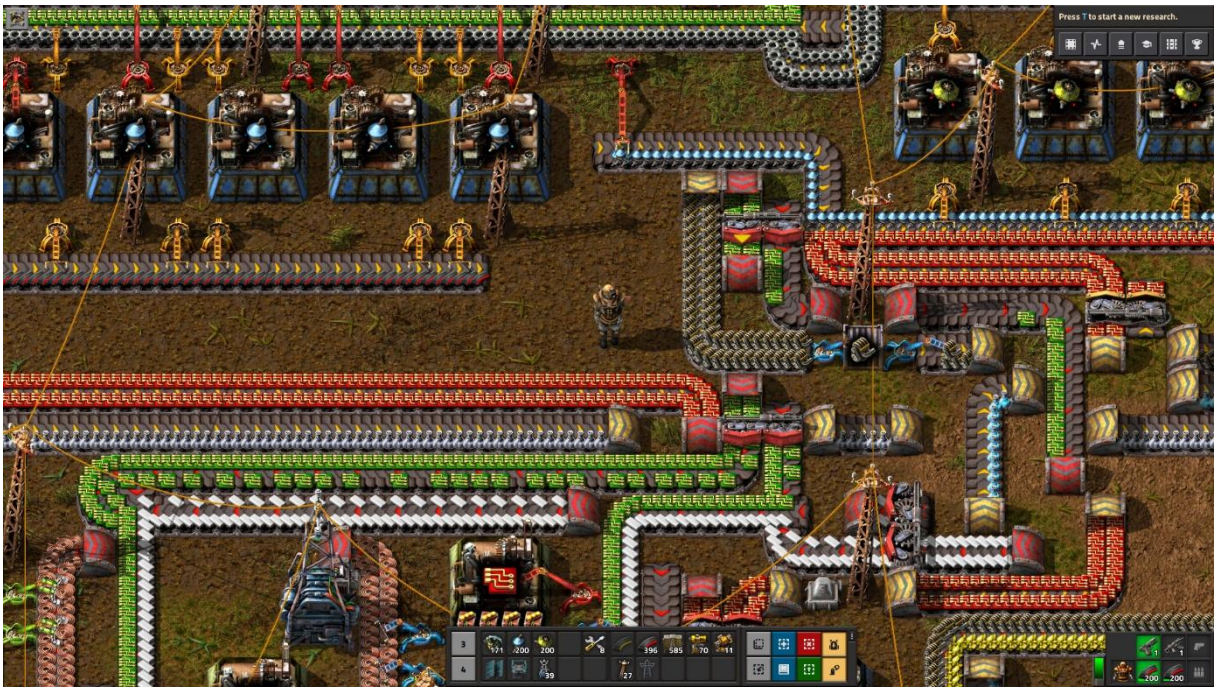


Abbildung 3: Komplexe Produktionsketten im Automatisierungsspiel Factorio (Wube Software, 2020). Quelle: Factorio. Abgerufen am 09.09.2025, von https://www.factorio.com/game/screenshots

3 Forschungsstand

Die Auseinandersetzung mit bestehenden Studien und Anwendungen im Bereich gamifizierter Programmierlernspiele liefert wichtige Erkenntnisse für die Einordnung und Bewertung des im Projekt *Learning by Playing* entwickelten Prototyps. In diesem Kapitel wird ein Überblick über den aktuellen Stand der Forschung gegeben. Es werden zunächst Meta-Analysen zur Wirksamkeit von Gamification allgemein sowie speziell im Kontext der Programmierausbildung vorgestellt. Daraufhin werden Einzelstudien zu ausgewählten Programmierspielen aus dem wissenschaftlichen Kontext und zuletzt Programmierspiele aus dem kommerziellen Kontext betrachtet.

3.1 Positive Effekte von Gamification

Verschiedene Meta-Analysen setzen sich mit den tatsächlichen Effekten von Gamification auseinander. Die Ergebnisse sind heterogen hinsichtlich der Wirksamkeit und zeigen oft offene Forschungsdesiderata auf.

Hamari et al. (2014) untersuchten in ihrer Meta-Analyse den aktuellen Stand der empirischen Forschung zu Gamification. Sie entwickelten ein Rahmenmodell, das die motivierenden Merkmale von Gamification, psychologische und Verhaltensfolgen strukturiert. Die Mehrheit der Studien zeigt positive Effekte von Gamification auf Motivation und Engagement. Allerdings beschreiben die Autor:innen auch Einschränkungen: Die Effekte sind meist nur teilweise signifikant, und es zeigen sich starke Abhängigkeiten vom jeweiligen Kontext sowie von den Eigenschaften der Nutzer. Außerdem weisen sie darauf hin, dass die Komplexität des Phänomens Gamification in vielen Studien noch unzureichend abgebildet wird und dass in zukünftigen Studien methodisch strengere Designs notwendig sind, um die Wirksamkeit besser zu erfassen. Insgesamt bestätigen Hamari et al. (2014) die Potenziale von Gamification, betonen jedoch die Notwendigkeit differenzierter Betrachtungen und weiterer Forschung.

Sailer et al. (2019) führten eine Meta-Analyse empirischer Studien durch und fanden heraus, dass Gamification signifikant positive Effekte auf kognitive, motivationale sowie verhaltensbezogene Lernergebnisse haben kann. Besonders stabil zeigten sich die Effekte auf kognitive Lernziele, während die Effekte auf Motivation und Verhalten in ihrer Ausprägung variieren. Die Analyse hebt hervor, dass adaptive Feedbackmechanismen sowie die Kombination spielerischer Elemente mit individuellen Lernstrategien förderlich für nachhaltiges Lernen sind. Zudem betonen die Autor:innen, dass soziale Interaktion und die Kombination von Wettbewerb mit Kooperation wichtige Faktoren für den Lernerfolg sein können. Trotz der positiven Resultate bleiben viele Fragen zur genauen Wirkweise von Gamification offen, und die Wirksamkeit hängt stark vom Kontext und den Eigenschaften der Lernenden ab. Sailer et al. fordern daher weitere Forschung, um die komplexen Mechanismen der Gamification besser zu verstehen und weiterzuentwickeln.

Diese Meta-Analysen verdeutlichen, dass Gamification nicht nur motivationale Effekte entfaltet, sondern auch messbare Lernerfolge erzielt, was insbesondere für die Vermittlung komplexer Inhalte wie der Programmierung von großer Bedeutung ist. Es wird jedoch auch klar, dass zusätzliche Forschung nötig ist.

3.2 Wirksamkeit gamifizierter Lernumgebungen in der Programmierausbildung

In den letzten Jahren wurde eine Vielzahl an Studien zur Effektivität von Gamification in der Programmierausbildung veröffentlicht. Um eine Übersicht über die verschiedenen Ergebnisse unterschiedlicher Studien zu erhalten, betrachtete eine Meta-Analyse von Costa (2023) mehrere Studien. Dabei wurde der Effekt der Anwendung von Punkten, Abzeichen, Leveln, Avataren und Ranglisten analysiert. Die Analyse kommt zu dem Ergebnis, dass Gamification einen signifikanten positiven Effekt auf Lernmotivation und Lernerfolg haben. Auf den Lernerfolg scheint sich allerdings unter den analysierten Aspekten ausschließlich die Nutzung von Leveln auszuwirken, während die Nutzung von Abzeichen, Punkten, Avataren und Ranglisten sich scheinbar hauptsächlich auf die Motivation auswirken. Besonders wirksam sind diese Elemente bei Anfänger:innen, da sie sowohl das Interesse steigern als auch den Lernprozess strukturieren.

Auch eine systematische Übersicht von Ishaq et al. (2024) zeigt, dass gamifizierte Lernformate, die kognitive Strategien mit personalisierten Feedbackmechanismen verknüpfen, zu besseren Ergebnissen im Erwerb von Programmierkenntnissen führen. Die Kombination von Gamification mit adaptiven Lerntechnologien erweist sich als besonders vorteilhaft.

Mellado et al. (2024) vergleichen in ihrer Studie zu Data-Structure-Kursen die Ergebnisse von Student:innen, in deren Kursen Gamification angewandt wurde, sowie von Student:innen, in deren Kursen dies nicht der Fall war. Die Untersuchung stellt fest, dass Gamification zu einem größeren Lernzuwachs führte. Zudem wird angemerkt, dass die positiven Effekte bei steigender Komplexität der Lerninhalte nicht verringert werden.

Eine Meta-Analyse von Li et al. (2023) fasst 41 Studien mit über 5.000 Teilnehmenden zusammen und zeigt einen signifikant großen positiven Gesamteffekt von Gamification auf Lerneffekte. Moderatoranalysen weisen darauf hin, dass Faktoren wie der Typ der Lernenden, der Fachbereich, die Gestaltung der Gamification-Elemente, die Dauer der Nutzung sowie die Lernumgebung die Wirksamkeit beeinflussen. Dies unterstreicht, dass Gamification besonders dann erfolgreich sein kann, wenn sie gezielt auf die Bedürfnisse und Kontexte der Lernenden abgestimmt ist. Spielbasierte Einstiegsformate wie visuelle Programmiersysteme oder narrative Lernumgebungen könnten dabei speziell für Anfänger:innen vorteilhaft sein, da sie eine motivierende und zugängliche Lernumgebung schaffen.

Ergänzend dazu liefert eine systematische Literaturübersicht von Giannakoulas und Xinogalos (2023) weitere Erkenntnisse. Basierend auf 61 Studien zu Serious Games in der Primarstufe zeigt sich, dass bildungsorientierte Spiele die Entwicklung von Computational Thinking (CT)-Fähigkeiten und das Verständnis grundlegender Programmierkonzepte signifikant fördern können. Darüber hinaus berichten die Autor:innen von einer generell positiven Haltung der Schüler:innen gegenüber dem Einsatz solcher Spiele, die als motivierend und lernförderlich wahrgenommen werden. Gleichzeitig verweisen die Befunde jedoch auf methodische Einschränkungen und Forschungsdesiderate, etwa die Notwendigkeit längerer und kontrollierter Studien. Insgesamt verdeutlicht diese Übersicht, dass spielbasierte Ansätze gerade im Grundschulalter einen wichtigen Beitrag zur Entwicklung grundlegender Programmier- und Denkfähigkeiten leisten können.

3.3 Evaluation konkreter Programmierspiele mit wissenschaftlichem Hintergrund

Verschiedene Studien haben die Wirksamkeit einzelner Programmierspiele untersucht. Ein prominentes Beispiel ist Light-Bot (Yaroslavski, 2008), ein Puzzle-basiertes Spiel, das Programmierlogik durch visuelles Problemlösen vermittelt (Abbildung 4). Gouws et al. (2013) zeigen, dass Light-Bot besonders geeignet ist, grundlegende Konzepte des Computational Thinking zu fördern. Dabei werden sowohl Stärken als auch Schwächen einzelner Konzepte erkannt. Als potenzielles Problem wird die Übertragbarkeit der im Spiel erlernten Konzepte auf echte Programmieraufgaben genannt, die für Anfänger:innen nicht immer offensichtlich ist.

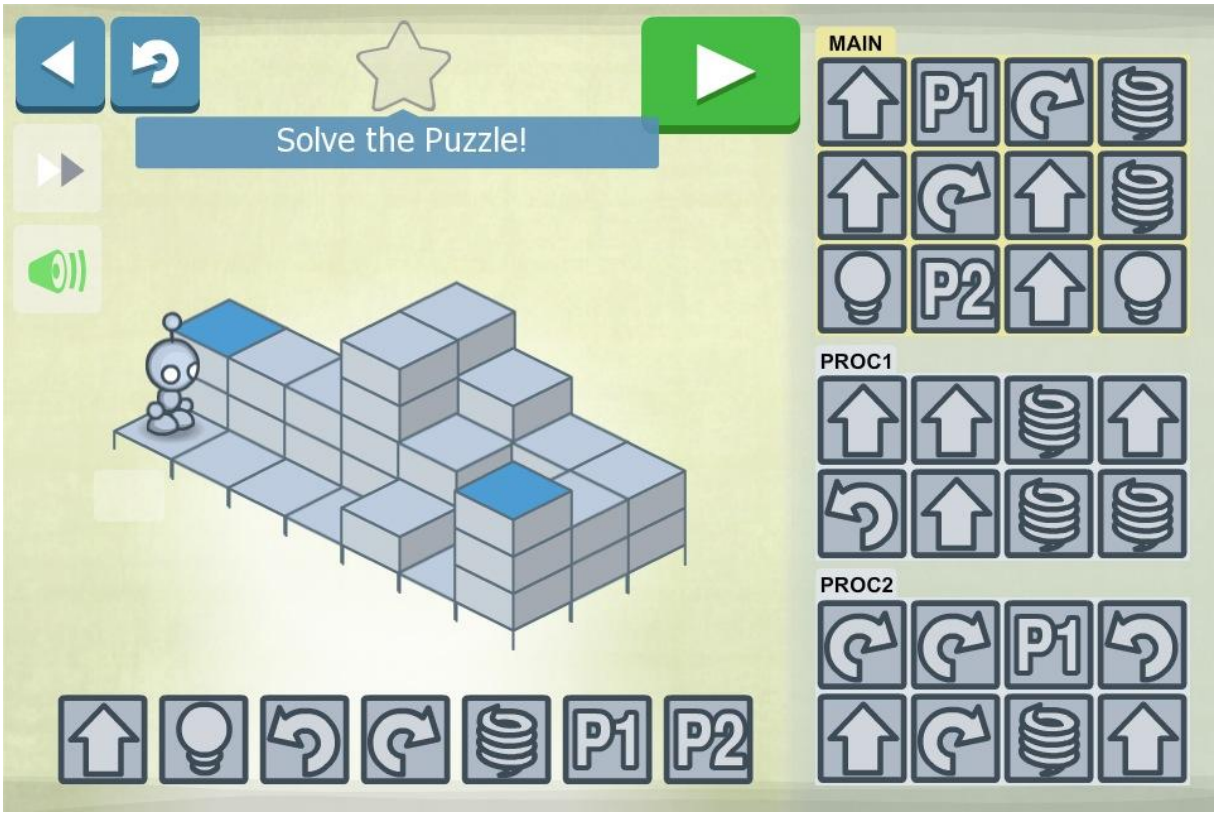


Abbildung 4: Screenshot aus dem Spiel LightBot (LightBot Inc., 2008). Quelle: StrategyWiki. Abgerufen am 09.09.2025, von https://strategywiki.org/wiki/Lightbot/Challenge_Levels

Auch die visuelle Programmierumgebung Scratch (MIT Media Lab, 2007) wurde vielfach untersucht. Resnick et al. (2009) betonen, dass Scratch nicht nur das Verständnis für Programmierlogik, sondern auch kreative und kooperative Kompetenzen fördert. In ihrer Untersuchung zeigen sie, wie durch das Teilen von Projekten in der Online-Community ein sozialer Lernraum entsteht, der für konstruktivistische Ansätze besonders passend ist.

Code.org (2013) ist eine Online-Plattform, die interaktive Tutorials und spielerische Aufgaben anbietet, um Kindern und Jugendlichen Programmieren anschaulich näherzubringen (Abbildung 5). Eine Studie von Choi und Cho (2024) zeigt, dass der Einsatz von Code.org in der schulischen Programmierausbildung sowohl die Computational Thinking-Fähigkeiten als auch die Lernmotivation und die Einstellung der Schülerinnen und Schüler gegenüber dem Programmieren deutlich verbessert. Dabei wird betont, dass neben dem Fachwissen auch eine motivierende Lernumgebung und individuelle Fördermaßnahmen wichtig sind, um unterschiedliche Lernvoraussetzungen zu berücksichtigen. Die Ergebnisse unterstrei-

chen somit die Bedeutung einer ganzheitlichen und motivierenden Herangehensweise im Programmierunterricht.

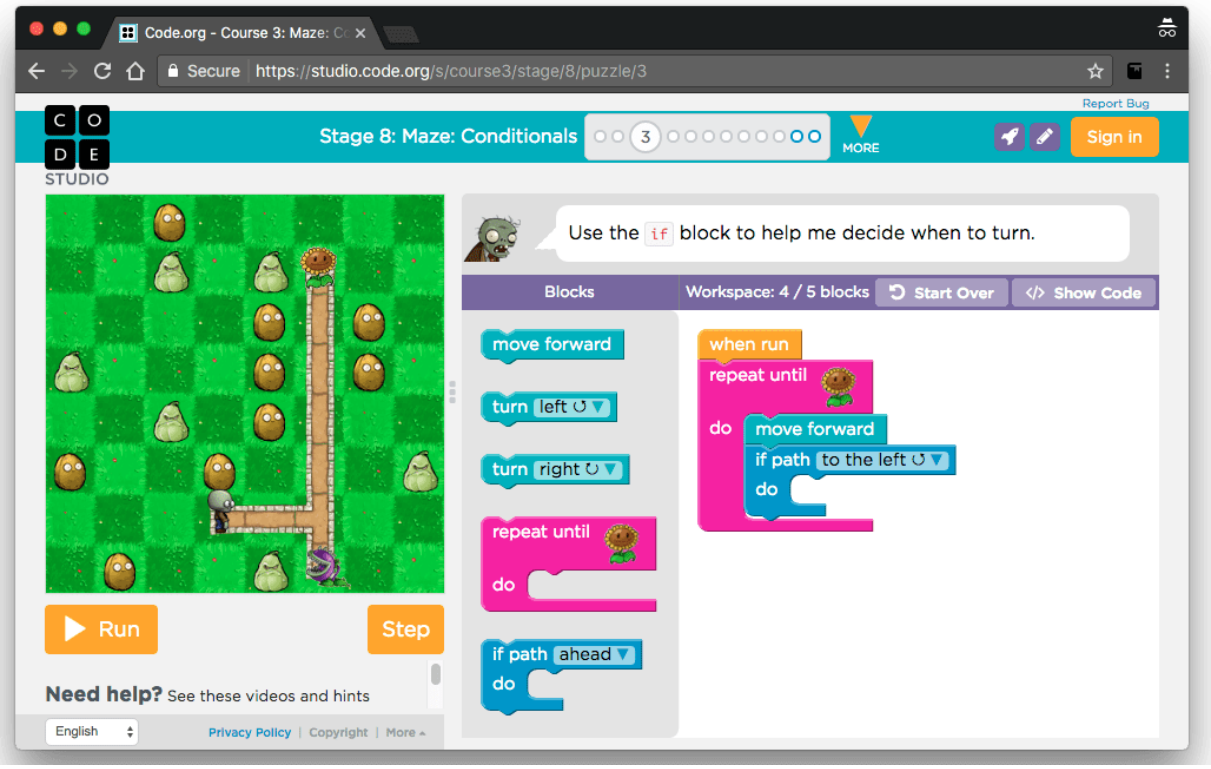


Abbildung 5: Screenshot aus der Lernumgebung Code.org. Quelle: Code.org, abgerufen am 09.09.2025, von https://code.org/images/proficiency

Ein weiteres Beispiel ist das blockbasierte Programmier-Lernspiel Penguin Go, das im Rahmen einer Studie von Zhao und Shute (2019) speziell zur Förderung von Computational-Thinking-Fähigkeiten entwickelt wurde (Abbildung 6). Die Autor:innen zeigen, dass bereits nach weniger als zwei Stunden Spielzeit signifikante Verbesserungen der CT-Kompetenzen bei Schüler:innen der Mittelstufe messbar waren. Interessant ist, dass eine im Spiel integrierte Zusatzbedingung – die Begrenzung der Blockanzahl in den Lösungen – keinen positiven Einfluss auf das Lernen hatte und sogar zu einer Verschlechterung der Einstellung gegenüber Informatik führte. Dies verdeutlicht, dass zwar spielbasierte Lernumgebungen effektiv zur Kompetenzförderung beitragen können, bestimmte Designentscheidungen jedoch unbeabsichtigte negative Effekte auf die Motivation und Einstellung der Lernenden haben können.

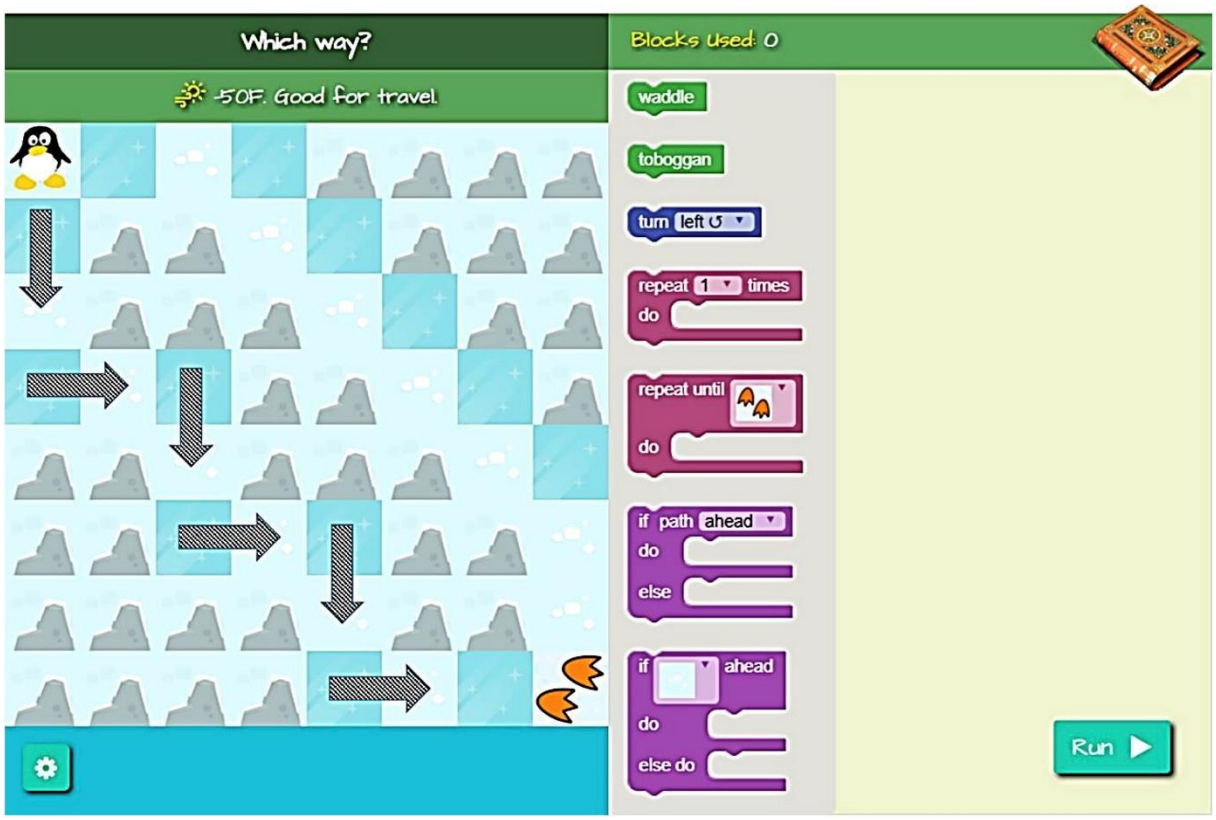


Abbildung 6: Visualisierung der Spielmechanik in Penguin Go. Quelle: Zhao & Shute, 2019

Neben digitalen Umsetzungen existieren auch analoge oder hybride Ansätze, die Programmierkonzepte vermitteln. Ein Beispiel hierfür ist Program Wars (Anvik et al., 2019), ein im Rahmen einer Studie entwickeltes, kartenspielbasiertes Lernspiel, das grundlegende Programmier- und Cybersicherheitskonzepte ohne die Notwendigkeit einer konkreten Programmiersprache vermittelt (Abbildung 7). Anvik, Cote und Riehl (2019) zeigen, dass die Spielenden in der Lage sind, die Spielkonzepte erfolgreich mit realen Programmiersprachen zu verknüpfen. Allerdings bleibt unklar, ob dadurch auch ein tiefergehendes Verständnis von Programmierung gefördert wird. Aufbauend darauf wurde im Rahmen einer Studie Program Wars v.2.0 entwickelt, das durch verbesserte Spielmechaniken, eine überarbeitete Benutzeroberfläche sowie die Einführung neuer Inhalte wie Such- und Sortieralgorithmen erweitert wurde. Tareque et al. (2024) berichten in ihrer Nutzerstudie, dass die zweite Version deutlich effektiver als die ursprüngliche war. So zeigten 60 % der Teilnehmenden Wissenszuwächse bei Variablen, 56 % bei Schleifen und 44 % bei Methoden. Zudem bestätigten qualitative Ergebnisse die hohe Motivation und das Engagement, die durch den spielbasierten Ansatz hervorgerufen wurden.

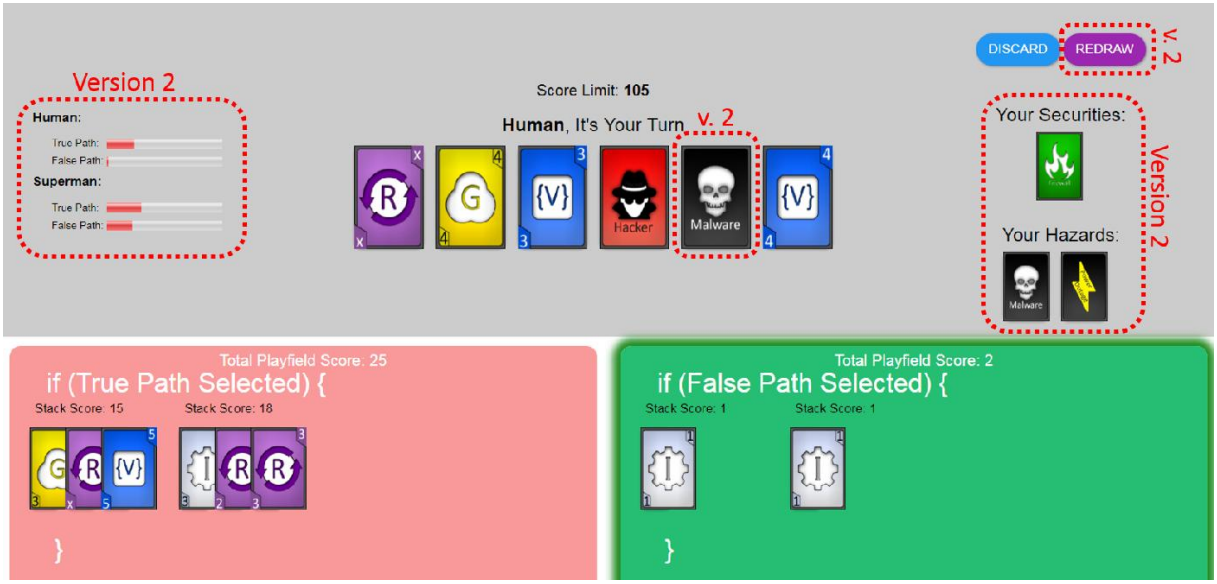


Abbildung 7: Screenshot aus Program Wars. Quelle: Anvik, Cote & Riehl, 2019

Obwohl einzelne Beispiele wie Light-Bot, Program Wars oder Penguin Go zeigen, dass vollständig als Spiel gestaltete Lernanwendungen im Programmierunterricht wirksam sein können, konzentriert sich ein großer Teil der wissenschaftlichen Literatur auf hybride Formate wie Scratch oder Code.org. Diese Plattformen kombinieren spielerische Elemente mit offenen Lernumgebungen, sind jedoch nicht als abgeschlossene Spiele mit festem Spielziel konzipiert. Entsprechend ist die Forschung zu klassischen Serious Games für das Erlernen von Programmierfähigkeiten relativ gering. Dies zeigt, dass es an systematischen Untersuchungen zu Lernspielen bedarf, die in ihrer Struktur und ihrem Spielfluss stärker traditionellen Videospielen ähneln, um das Potenzial dieser Art von Spielen im Programmierunterricht fundiert bewerten zu können.

3.4 Kommerzielle Lernspiele mit Gamification-Ansatz

Neben wissenschaftlich begleiteten Projekten existieren auch zahlreiche kommerzielle Spiele zur Programmierbildung. Ein Beispiel ist CodeCombat (2013), entwickelt von CodeCombat Inc., das klassische RPG-Elemente mit textbasierter Programmierung kombiniert (Abbildung 8). Die Lernenden steuern Spielfiguren durch Codebefehle, während sie Rätsel lösen und Gegner bekämpfen. Laut einer firmeneigenen Evaluationsstudie (*Ozaria Efficacy Report Mountain Ridge Middle School*, 2021; *Ozaria Efficacy Report Summary McIntosh Middle School*, 2021) berichten Lehrkräfte von signifikant gestiegenem Engagement und verbesserten Lernergebnissen, insbesondere bei jüngeren Lernenden. Diese Ergebnisse sind jedoch wenig wissenschaftlich und als firmeneigene Studien als graue Literatur entsprechend kritisch zu sehen.

Ergänzend dazu untersuchte allerdings auch eine wissenschaftliche Studie von Kroustalli und Xinogalos (2021) mit Sekundarschüler:innen den Einsatz von CodeCombat im Vergleich zu traditionellem

Unterricht mit Python (n = 59, quasi-experimentelles Design mit Kontrollgruppe, Pre-/Post-Test). Dabei zeigte sich, dass beide Gruppen Fortschritte in grundlegenden Programmierkonzepten erzielten, die Spielgruppe im Post-Test jedoch leicht bessere Ergebnisse erreichte. Dieser Unterschied war allerdings nicht signifikant, was die Autor:innen auf die kurze Interventionsdauer, die geringe Stichprobengröße und die enge Anbindung an das Curriculum zurückführen. Positiv bewertet wurde CodeCombat insbesondere hinsichtlich Benutzerfreundlichkeit, Nützlichkeit und Einstellung zur Nutzung, während die Verhaltensintention zur weiteren Verwendung neutral ausfiel. Insgesamt unterstreicht die Studie das Potenzial textbasierter Serious Games zur Unterstützung beim Programmieren lernen, verweist jedoch zugleich auf die Notwendigkeit längerer Einsatzzeiträume und größerer Stichproben für belastbarere Wirksamkeitsnachweise.

Abbildung 8: Screenshot aus CodeCombat. Quelle: bladee, 22.01.2021, Help: Ogre Invaders [Online-Forum-Post]. CodeCombat-Forum, abgerufen am 09.09.2025, von https://discourse.codecombat.com/t/help-ogre-invaders/26872

Swift Playgrounds (Apple Inc., 2016), eine von Apple entwickelte App für iPad und Mac, vermittelt Programmierkenntnisse in der Sprache Swift durch interaktive Aufgaben und spielerische Elemente (Abbildung 9). Eine Studie von Cheng und Chen (2021) untersuchte den Einsatz der App in einem Computational-Thinking-Kurs an taiwanischen Grundschulen. Die Autor:innen berichten, dass Lehrkräfte den Ansatz besonders zur Förderung logischen Denkens und inferentieller Fähigkeiten schätzen. Während grundlegende Aufgaben von den Schülerinnen und Schülern problemlos gemeistert wurden, bereiteten komplexere Konzepte wie Funktionen und Schleifen noch größere Schwierigkeiten. Die Studie hebt hervor, dass ergänzende Instruktionen und erweiterbare Kursmaterialien den Lernerfolg weiter steigern könnten.

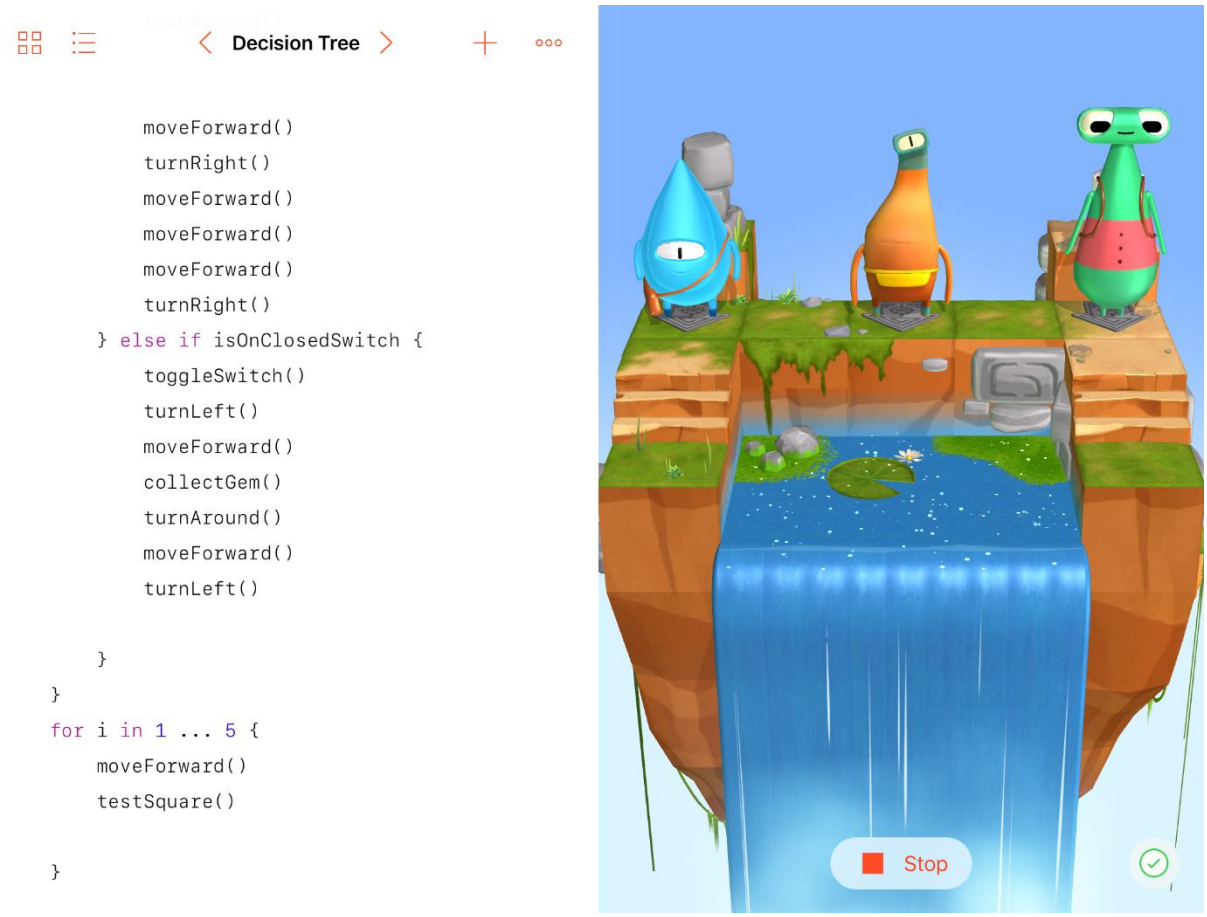


Abbildung 9: Screenshot aus dem Lernspiel Swift Playgrounds. Quelle: Ochs, 2016

Human Resource Machine (Tomorrow Corporation, 2015) ist ein Programmierspiel, das Programmierkonzepte wie Schleifen, Bedingungen und Speicheroperationen in einer Büro-ähnlichen Umgebung vermittelt (Abbildung 10). Spieler:innen programmieren mithilfe einfacher Befehle Büroangestellte, die Aufgaben wie das Sortieren und Verschieben von Zahlen erledigen müssen, um komplexere Arbeitsprozesse zu automatisieren. In einer Analyse von Heithausen (2020) wird das Spiel im Kontext von Seymour Paperts konstruktivistischen Lernprinzipien betrachtet. Demnach fördert Human Resource Machine aktives Problemlösen und selbstständiges Konstruieren von Wissen, was als zentral für effektives Lernen gilt. Die Autorin hebt hervor, dass das Spiel durch seine schrittweise steigenden Aufgaben und motivierende Gestaltung grundlegende Programmierfähigkeiten vermittelt, während soziale Interaktion und kollaboratives Lernen weniger berücksichtigt werden. Insgesamt wird das Spiel als gelungenes Beispiel für die Umsetzung konstruktivistischer Ansätze in digitalen Lernspielen eingeschätzt.

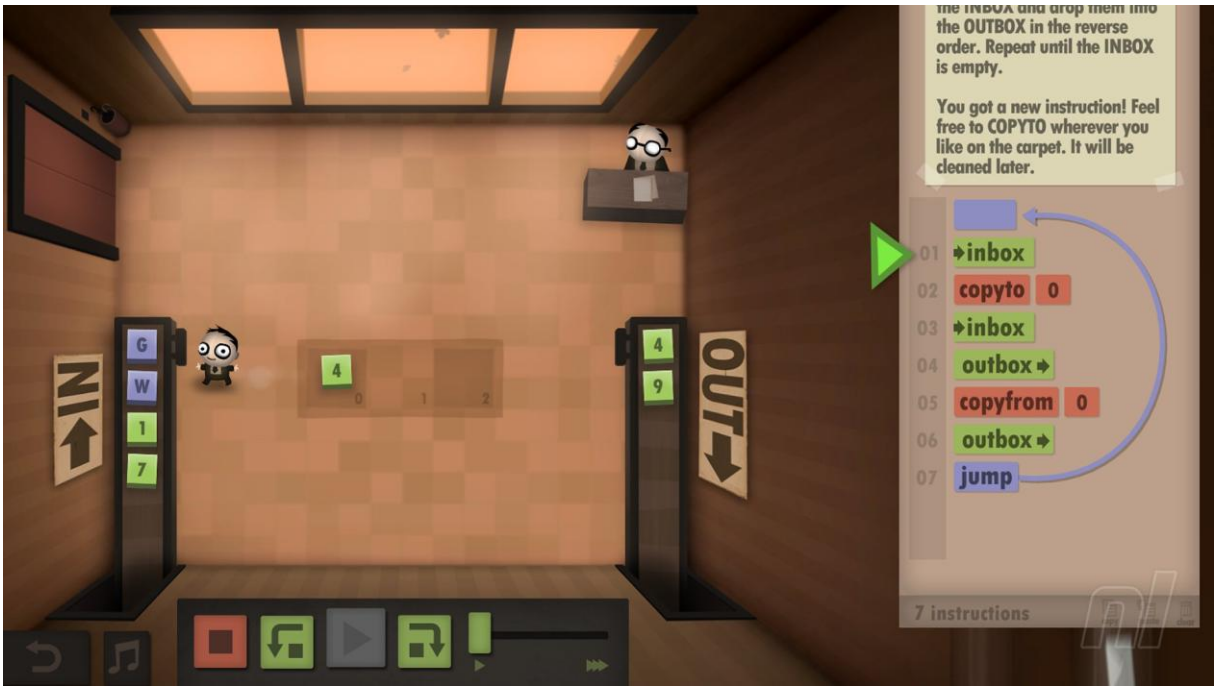


Abbildung 10: Screenshot aus dem Spiel Human Resource Machine. Quelle: nintendolife, abgerufen am 09.09.2025, von https://www.nintendolife.com/games/switch-eshop/human_resource_machine

Neben wissenschaftlichen Untersuchungen geben auch Nutzerbewertungen auf Spieleplattformen Hinweise auf die Akzeptanz von Programmierspielen. So wird Human Resource Machine mit über 3.000 Rezensionen auf Steam überwiegend positiv bewertet (Stand: 05.09.2025). Ein weiteres Beispiel ist The Farmer Was Replaced (Timon Herzog, 2020), das bei mehr als 1.600 Rezensionen sogar eine äußerst positive Bewertung erreicht (Stand: 05.09.2025). Solche Bewertungen deuten darauf hin, dass gut gestaltete Programmierspiele in der Praxis durchaus auf hohe Zustimmung und Spielfreude bei Nutzer:innen stoßen können.

3.5 Zusammenfassung

Abschließend zeigt die Auseinandersetzung mit dem Forschungsstand, dass gamifizierte Lernspiele und Lernumgebungen großes Potenzial für die Vermittlung von Programmierkenntnissen besitzen. Während Meta-Analysen und Einzelstudien überwiegend positive Effekte von Gamification auf Motivation, Engagement und Lernerfolg belegen, ist die empirische Forschung zu vollständig spielbasierten Programmierlernspielen bislang vergleichsweise gering. Ein großer Teil der untersuchten Anwendungen sind hybride Formate oder offene Lernplattformen wie Scratch oder Code.org, die zwar spielerische Elemente integrieren, jedoch nicht als klassische Spiele mit klar definierten Spielzielen konzipiert sind. Kommerzielle Spiele wie CodeCombat, Swift Playgrounds oder Human Resource Machine bieten motivierende und interaktive Zugänge zur Programmierung, werden jedoch in der Wissenschaft bisher nur punktuell untersucht. Dies weist auf weiteren Bedarf an systematischen und methodisch fundierten Stu-

dien, die Lernspiele mit stärkerer spieltypischer Struktur analysieren, um deren Potenziale für die Programmierbildung umfassend zu erfassen und weiterzuentwickeln. Dies bildet die Grundlage für die Einordnung und Bewertung des im Projekt *Learning by Playing* entwickelten Prototyps.

4 Prototyp *Learning by Playing*

Der Prototyp *Learning by Playing* ist ein rasterbasiertes 3D-Top-down-Automatisierungsspiel, in dem Spieler:innen das Verhalten von Einheiten in einer blockbasierten Programmieroberfläche programmieren, um diese selbstständig Rohstoffe sammeln zu lassen. Der Prototyp ist unter folgendem Link spielbar: https://jonasmack.itch.io/terracode.

Ziel des Projekts *Learning by Playing* war die Entwicklung eines Prototyps, anhand dessen das Potential des Konzepts eines Automatisierungsspiels zum Programmieren lernen demonstriert werden kann, um als Grundlage zur Förderung und Entwicklung eines vollständigen Spiels zu dienen.

Der Prototyp wurde mit Unity 6000.0.25f1 (Unity Technologies, 2024) entwickelt.

4.1 Spielkonzept

Im Zentrum des Spiels steht die Programmierung autonomer Einheiten, die Ressourcen auf einem rasterbasierten Spielfeld sammeln. Das Hauptziel besteht darin, Einheiten zunächst zum Ressourcensammeln zu programmieren und anschließend die Programmierung zu verbessern und zu erweitern, um schneller Ressourcen zu sammeln und den Automationsgrad weiter zu erhöhen. Zusätzlich können Ressourcen ausgegeben werden, um neue Gebäude, Einheiten und Programmbausteine freizuschalten. Diese bieten Spielenden anschließend neue Möglichkeiten, den Abbau der Ressourcen zu automatisieren oder bestehende Automatisierung effizienter zu gestalten. Spielende können dabei nicht verlieren, wenn sie Fehler machen. Stattdessen stagniert der Fortschritt oder Einheiten stehen still, bis Fehler behoben wurden.

Die nötigen Programmierkenntnisse werden im Rahmen eines Tutorials spielerisch vermittelt. Die Programmierung erfolgt blockbasiert und abstrahiert auf der Grundlage von Java. Das Spiel soll sich primär an Schüler:innen der Oberstufe mit wenig bis keinen Programmierkenntnissen richten und sekundär an Student:innen im ersten Semester.

4.2 Einordnung des Spielkonzepts

Während viele wissenschaftlich evaluierte Programme wie Scratch oder Code.org hybride Lernumgebungen mit spielerischen Elementen darstellen, nähert sich der Prototyp einem klassischen Spiel mit klar definierten Spielzielen und einer geschlossenen Spiellogik an. Damit bewegt sich das Spiel konzeptionell näher an kommerziellen Programmierspielen wie CodeCombat oder Human Resource Machine. Gleichzeitig ist das Ziel des Projekts *Learning by Playing* ein Spiel als Hochschulprojekt zu finanzieren und zu entwickeln, was es von kommerziellen Produkten unterscheidet und die Möglichkeit bietet, wissenschaftliche Qualitätsstandards und didaktische Erkenntnisse in den Fokus zu rücken. Diese

Kombination aus spieltypischem Design und akademischer Fundierung eröffnet neue Chancen, das Potenzial klassischer Spielelemente für die Programmierbildung fundiert zu erforschen und weiterzuentwickeln.

4.3 Gameloop und Progressionsmechanik

Der grundlegende Spielablauf folgt einer iterativen Struktur, sodass der geschilderte Ablauf erneut beginnt, wenn der letzte Schritt durchgeführt wurde.

- Einheiten programmieren (z. B. zur Ressourcensuche),
    - Programme testen und ggf. Fehler beheben,
- Ressourcen automatisiert sammeln lassen,
- Mithilfe gesammelter Ressourcen Fortschritt erzielen
    - Zusätzliche Gebäude und Einheiten bauen,
    - Neue Arten von Gebäuden und Einheiten bauen,
    - Neue Arten von Komponenten zur Programmierung freischalten.

4.4 Spielfeld und Umgebung

Das Spielfeld besteht aus einem räumlich begrenzten, quadratischen Raster (Abbildung 11). Jedes Feld kann ein Gebäude, eine Ressource oder eine Einheit enthalten. Da Einheiten fliegen, können sie sich auf demselben Feld wie ein Gebäude oder eine Ressource befinden. Die Himmelsrichtungen sind am Spielfeldrand sichtbar markiert (die Himmelsrichtungen werden zur Programmierung der Bewegung der Späheinheit verwendet).

Ressourcen (Schrott und Erz) erscheinen zufällig und werden nach dem Einsammeln wieder neu platziert, wodurch sie jederzeit verfügbar bleiben. Diese Ressourcen werden benötigt, um neue Komponenten freizuschalten sowie um neue Gebäude und Einheiten zu bauen.

Abbildung 11: Ansicht der Spielwelt zum Zeitpunkt des Spielstarts, 1: Kraftwerk, 2: Depot, 3: Arbeitseinheit, 4: Schrott, 5: Erz.

4.5 Interaktive Benutzeroberfläche

Die Benutzeroberfläche (Abbildung 12, 13) unterstützt das Programmieren und Ressourcenmanagement durch:

- Ressourcenleiste mit Anzeige der verfügbaren Energie, Erz und Schrott,
- Forschungsmenü zur Freischaltung neuer Komponenten,
- Baumenü zum Errichten von Gebäuden und Einheiten,
- Editorfenster, in dem der Programmierprozess erfolgt,
- Tutorialfenster, das die Spielenden schrittweise an die Funktionen heranführt und das nächste Ziel anzeigt (z. B. Programmierung der Bewegung der Arbeitseinheit).

Abbildung 12: Interaktive Benutzeroberfläche, 1: Tutorialfenster, 2: Ressourcenleiste, 3: Baumenü.

Abbildung 13: Forschungsmenü.

4.6 Editor

Der Editor stellt die Programmieroberfläche dar (Abbildung 14). In diesem befinden sich auf der linken Seite alle Komponenten, die für die Programmierung der ausgewählten Art von Einheit verfügbar sind.

Die verfügbaren Komponenten können mehrfach verwendet werden. Auf der rechten Seite befindet sich das für die ausgewählte Einheit programmierte Verhalten. Um dieses zu erstellen, werden Komponenten von der linken Seite auf die Rechte gezogen und dort untereinander angeordnet. Um die Einheit das programmierte Verhalten ausführen zu lassen, muss auf Ausführen geklickt werden.

Komponenten, die Teil des Programms auf der rechten Seite des Editors sind, kosten Energie. Die Energiekosten der Programme aller Einheiten werden dabei zusammengezählt. Wird die verfügbare Energie überschritten, bleiben die Einheiten stehen, bis Komponenten aus Programmen entfernt werden oder die Energiekapazität durch Bau zusätzlicher Kraftwerke vergrößert werden. Somit soll vermittelt werden, dass die Ausführung von Befehlen Rechenleistung kostet und Programme somit unterschiedlich effizient sein können.

Abbildung 14: Ansicht des Editors einer Arbeitseinheit, links: verfügbare Komponenten, rechts: Programmierung der Einheit.

4.7 Einheiten und Gebäude

Es gibt zwei zentrale Einheitentypen: *Späheinheiten*, die das Spielfeld absuchen und Rohstoffe markieren sowie *Arbeitseinheiten*, die Rohstoffe sammeln und sie an Depots liefern.

Gebäudetypen umfassen *Depots* zur Lagerung von Ressourcen, *Kraftwerke* zur Erhöhung der Energiekapazität und *Forschungslabore* zum Freischalten neuer Programmierkomponenten.

4.8 Programmierumgebung und Komponenten

Die Programmierung erfolgt über eine blockbasierte Oberfläche, ähnlich bekannten Systemen wie Scratch oder Blockly. Die Blöcke repräsentieren Funktionen (z. B. Bewegung oder Bedingung) und können beliebig kombiniert werden.

Die folgende Tabelle zeigt, welche Komponenten für die jeweilige Einheit im Tutorial benötigt werden und liefert Erklärungen zur Funktionsweise der Komponenten.

Tabelle 1: Beschreibung der im Tutorial benötigten Komponenten

Komponente	**Funktionsweise**	**Einheiten, die über die Komponente verfügen**
Solange	Wiederholt die Ausführung der darin platzierten Komponenten, solange eine Bedingung wahr ist. Benötigt ein boolesches Argument, dass die Dauer der Ausführung bestimmt. Das Argument ist per Default *wahr*.	Späheinheit Arbeitseinheit
Bewegen (in Richtung)	Benötigt ein Vector2 Argument, das die Richtung angibt, in die die Bewegung erfolgen soll. Die Ausführung hat eine Bewegung in die angegebene Richtung um ein Rasterfeld zur Folge.	Späheinheit
Norden, Osten, Süden, Westen	Ein Vector2 Argument für die Bewegen Komponente der Späheinheit, das die entsprechende Richtung darstellt.	Späheinheit
Wenn	Erlaubt die Platzierung anderer Komponenten innerhalb der *Wenn*-Bedingung, die nur ausgeführt werden, wenn das an die *Wenn*-Komponente übergebene Argument *wahr* ist. Benötigt ein boolesches Argument als Ausführungsbedingung für die in der *Wenn*-Bedingung platzierten Komponenten.	Späheinheit

Über Erz	Gibt *wahr* zurück, wenn die Einheit sich zum Zeitpunkt der Ausführung über Erz befindet. Wenn sie es nicht tut, gibt die Komponente *unwahr* zurück.	Späheinheit
Erz markieren	Platziert eine Erzmarkierung an der Position, an der sich die Einheit zum Zeitpunkt der Ausführung befindet. Wird unabhängig davon ausgeführt, ob die Einheit sich tatsächlich über Erz befindet.	Späheinheit
Bewegen (zu Koordinaten)	Benötigt ein Vector2 Argument, das die Position angibt, zu der die Bewegung erfolgen soll. Die Ausführung hat eine Bewegung bis zum Ziel zur Folge.	Arbeitseinheit
Erz abbauen	Lässt die Einheit Erz einsammeln, wenn sie sich zum Zeitpunkt der Ausführung über Erz befindet.	Arbeitseinheit
Abliefern	Lässt die Einheit Erz abliefern, wenn sie zuvor welches eingesammelt hat und sich zum Zeitpunkt der Ausführung über einem Depot befindet. Das Abliefern von Erz an ein Depot macht dieses für Spieler:innen verfügbar, um z. B. weiter Einheiten zu bauen.	Arbeitseinheit
Nächste Erzmarkierung	Gibt die Vector2 Position der nächsten Erzmarkierung auf dem Spielfeld zurück.	Arbeitseinheit
Nächstes Depot	Gibt die Vector2 Position des nächsten Depots auf dem Spielfeld zurück.	Arbeitseinheit

4.9 Tutorial

Im Rahmen der Studie soll das Tutorial absolviert werden. Im Tutorial programmieren Spieler:innen in aufeinander aufbauenden Missionen zunächst eine Arbeitseinheit und anschließend eine Späheinheit.

Dabei werden verschiedene Komponenten und Verständnis grundlegender Programmierkonzepte benötigt, die spielbezogen durch Text, Videos und Exploration erlernt werden können. Die Videos zeigen beispielhaft die korrekte Implementation der jeweiligen Komponenten im Editor.

Zu Beginn des Tutorials starten Spielende auf einem nahezu leeren Spielfeld. Zu diesem Zeitpunkt gibt es bereits eine Arbeitseinheit, ein Kraftwerk und ein Depot. Das Tutorial beginnt mit der Erklärung des Ziels, die Einheiten zu programmieren, selbstständig Rohstoffe abzubauen. Zunächst wird die Aufgabe der Arbeitseinheit, Rohstoffe zu sammeln und an das Depot zu liefern, erläutert. Erzmarkierungen werden zu diesem Zeitpunkt manuell von der spielenden Person platziert, was später durch die Späheinheit erledigt wird. Darauf folgt eine Einführung in die Funktionsweise des Editors (Abbildung 15), in dem zunächst nur die Bewegung zur nächstliegenden Erzmarkierung programmiert wird (Abbildung 16).

Abbildung 15: Erste Programmierung der Arbeitseinheit im Tutorial.

Abbildung 16: Ausführung der programmierten Bewegung zur Erzmarkierung (pink) durch die Arbeitseinheit.

Nachdem *Bewegen* erfolgreich ausgeführt wurde, erfolgt die Programmierung des restlichen Prozesses inklusive Abbauen des Erzes, Bewegung zum nächstliegenden Depot und Abliefern des Erzes an das Depot (Abbildung 17, 18).

Abbildung 17: Fortgeschrittene Programmierung der Arbeitseinheit im Tutorial.

Abbildung 18: Ausführung des programmierten Abbaus von Erz und darauffolgende Bewegung zum Depot und Abliefern des Erzes an das Depot durch die Arbeitseinheit.

Nachdem die Arbeitseinheit erfolgreich programmiert wurde und auf Knopfdruck von Spielenden markiertes Erz abbauen und an das Depot liefern kann, bauen Spielende ein Forschungslabor, das die Erforschung neuer Komponenten im Forschungsmenü erlaubt (Abbildung 19). Zunächst wird somit die *Solange*-Komponente erforscht.

Abbildung 19: Freischaltung der *Solange*-Komponente im Forschungsmenü im Tutorial.

Mithilfe dieser Komponente kann nun das Programm der Arbeitseinheit automatisiert werden, wodurch das Verhalten von nun an wiederholt ausgeführt wird, anstatt wie zuvor jeweils auf Knopfdruck (Abbildung 20).

Abbildung 20: Automatisierung des Verhaltens der Arbeitseinheit mithilfe der neu erworbenen *Solange*-Komponente.

Im letzten Teil des Tutorials bauen Spielende eine Späheinheit, die programmiert werden kann, selbstständig Erz für die Arbeitseinheit zu markieren, das Spielende zuvor selbst markieren mussten (Abbildung 21). Dies wird Spielenden zunächst demonstriert, indem sie eine Erzmarkierung setzen lassen und anschließend die Einheit weiterbewegen, um den Effekt sichtbar zu machen (Abbildung 22).

Abbildung 21: Erste Programmierung der Späheinheit im Tutorial.

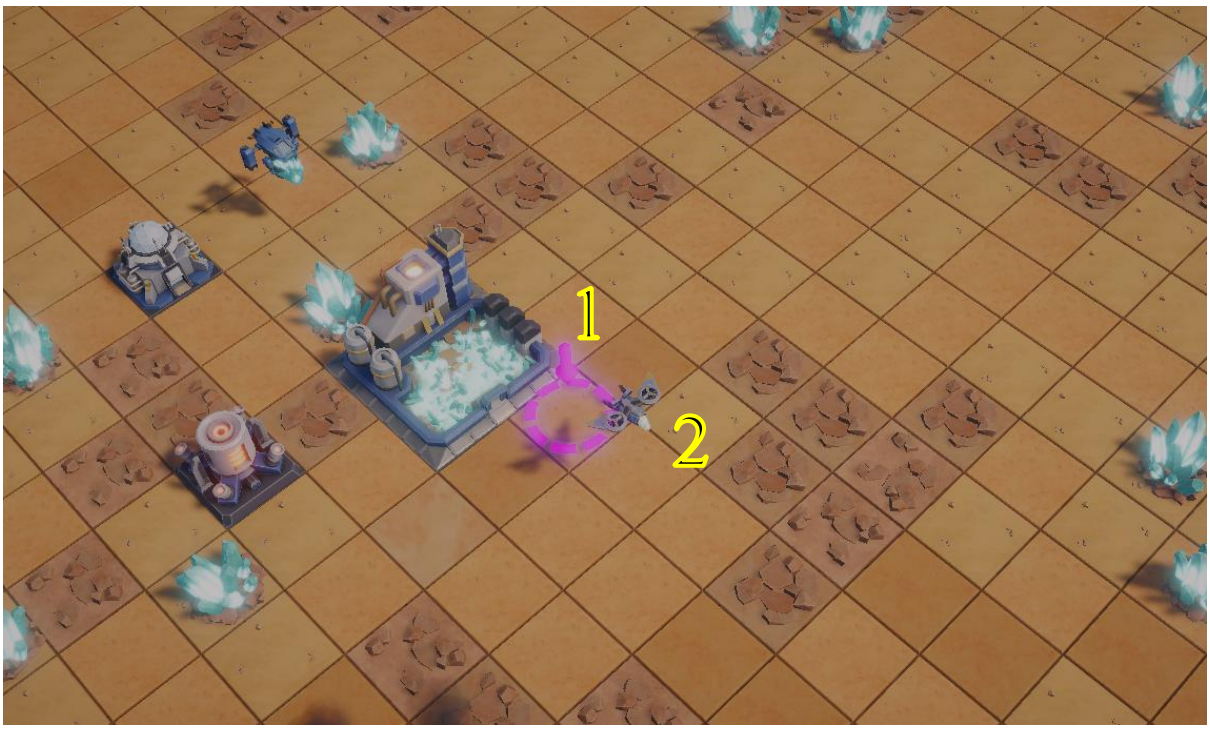


Abbildung 22: Ausführung der programmierten Platzierung einer Erzmarkierung und darauffolgender Bewegung der Späheinheit, 1: Erzmarkierung, 2: Späheinheit.

An dieser Stelle wird die *Wenn*-Komponente eingeführt. Mit dieser kann überprüft werden, ob sich an der aktuellen Position der Späheinheit Erz befindet. Das erlaubt es, nur an solchen Positionen Markierungen für die Arbeitseinheit zu platzieren. Zuletzt wird auch das Verhalten der Späheinheit mithilfe der *Solange*-Komponente automatisiert (Abbildung 23, 24).

Abbildung 23: Erweiterte Programmierung der Späheinheit mithilfe einer Bedingung und einer Schleife.

Abbildung 24: Ausführung der programmierten wiederholten Bewegung und Überprüfung auf Erzvorkommen durch die Späheinheit mit entsprechender Platzierung von Erzmarkierungen.

Anschließend erfolgt der Hinweis darauf, dass der Post-Test nun durchgeführt werden kann.

4.10 Programmierkonzepte

Die folgenden Programmierkonzepte werden im beschriebenen Tutorial explizit erklärt.

- Sequentielle Ausführung von Programmcode
- Methodenargumente und -parameter (z. B. Position als Argument für *Bewegen*-Komponente)
- Fußgesteuerte Schleife (*Solange*-Schleife)
- Verzweigung (*Wenn*-Anweisung)

Zudem ist implizit das Konzept der Rückgabewerte enthalten. So verwenden Spielende z. B. die Komponente *Über Erz*, die einen Bool zurückgibt, in Verbindung mit der *Wenn*-Komponente.

Das Konzept der Fehlersuche und -behebung ist implizit enthalten. Spielenden werden im Tutorial Aufgaben erteilt. Erfüllt die programmierte Einheit die jeweilige Aufgabe nicht, müssen Spielende Fehler in dem Programm der Einheit suchen und beheben, um im Spiel fortzuschreiten.

4.11 Designprinzipien

Die Gestaltung des Spiels folgte mehreren didaktischen und spielbasierten Designprinzipien, die sich in der Forschung zu Lernspielen bewährt haben.

4.11.1 Konstruktivistisches Lernen

Spieler:innen lernen Programmierkonzepte, indem sie direkt mit ihnen interagieren und deren Auswirkungen im Spiel beobachten können. Beispielsweise wird das Verhalten der Arbeitseinheit programmiert und anschließend ausgeführt. Somit sehen Spielende unmittelbar, wie sich die Einheit entsprechend der Programmierung bewegt und können davon ableiten, welche Wirkung einzelne Komponenten haben und wo gegebenenfalls Fehler in der Programmierung liegen. Dieses Prinzip orientiert sich am *Experiential Learning* nach Kolb (1984) und an Paperts Konstruktionismus, bei dem Lernen durch aktives Konstruieren eigener Lösungen stattfindet (Papert, 1980).

4.11.2 Progressive Disclosure

Komplexität wird schrittweise eingeführt. Am Beispiel der Späheinheit lernen Spielende zunächst, eine Erzmarkierung zu setzen. Erst danach folgen die Programmierung der Bedingung („nur markieren, wenn Erz darunter liegt“) sowie die Wiederholung des Verhaltens mit einer Schleife. Dieses Prinzip reduziert kognitive Überlastung indem neue Inhalte erst dann eingeführt werden, wenn Spielende bereit dafür sind.

4.11.3 Immediate Feedback

Das Verhalten der programmierten Einheiten liefert den Spielenden eine unmittelbare Rückmeldung über die Qualität ihres Programms. Spielende können durch Beobachtung feststellen, ob das Programm zum gewollten Verhalten führt oder nicht. Diese Form des prozessnahen Feedbacks entspricht dem, was Hattie & Timperley (2007) als besonders lernförderlich beschreiben, da Lernende direkt nach der Handlung Rückmeldung erhalten und diese zur Anpassung ihrer Strategien nutzen können.

4.11.4 Gamification und Zielstruktur

Das Ressourcensystem dient als Fortschrittsanzeige und Motivationsstruktur. Spieler:innen sammeln Ressourcen, um neue Einheiten, Gebäude und Komponenten freizuschalten. Dies entspricht gängigen Gamification-Ansätzen, bei denen Spielmechaniken zur Steigerung von Motivation und Engagement verwendet werden (Deterding et al., 2011; Hamari et al., 2014).

4.11.5 Unterstützungsangebote (Scaffolding)

Spielende erhalten umfangreiche Hilfestellungen: Tooltips im Editor und in der Spielwelt, ein Tutorialfenster mit Missionstiteln und situativer Erklärung sowie kontextbezogene Hilfe direkt im Spiel. Diese Hilfestellungen entsprechen dem Konzept des Scaffolding (Wood et al., 1976) und Just-in-Time Help (*Instructional-design theories and models*, 1999), das gleichzeitig eigenständiges Denken und kognitive Entlastung fördert während Frustration reduziert wird.

4.12 Mögliche Probleme des Prototyps

Die folgenden Merkmale des Prototyps sind möglicherweise problematisch und bieten Potenzial für Verbesserungen. Die mit der *Solange*-Komponente erstellten Programmschleifen funktionieren im Spiel wie fußgesteuerte Schleifen. In der Regel sind der Beginn sowie das Ende einer fußgesteuerten Schleife durch eine Anweisung gekennzeichnet, während im Spiel nur das Ende der Schleife durch eine Anweisung markiert wird. Dies könnte syntaktisch verwirrend wirken. Rückgabewerte werden im Spiel nicht explizit erklärt. Es besteht das Risiko, dass Spielende die Funktionsweise der im Prototyp verwendeten Rückgabewerte nicht von selbst durch Rückschlüsse verstehen. Variablen und Operatoren sind kein Teil des Prototyps, wobei sie ein zentraler Bestandteil der gängigen Programmiersprachen sind. Somit fehlt nicht nur ein wesentlicher Teil der grundlegenden Programmierkonzepte, die es zu vermitteln gilt, es schränkt auch die Möglichkeiten zur Programmierung im Spiel sowie der Fragestellungen in der Studie ein. Die visuelle Programmieroberfläche des Spiels bietet Verbesserungspotenzial in Hinsicht auf Lesbarkeit und Darstellung der Zusammenhänge zwischen genutzten Komponenten.

4.13 Anpassungen des Prototyps für die Studie

Der Prototyp wurde für die bestimmten Anforderungen der Studie angepasst. Zur besseren Spielbarkeit wurden einige Bugs behoben, die für die Programmierung der Bewegungsrichtungen benötigten Himmelsrichtungen am Spielfeldrand entsprechend markiert und der freie Spielmodus ohne Tutorial wurde deaktiviert.

Da bei einem Test des Spiels zu Beginn der Studienentwicklung deutlich wurde, dass die Verwendung der englischen Sprache im Spiel für manche Spieler:innen ein zusätzliches Hindernis darstellt, wurde das gesamte Spiel auf Deutsch übersetzt.

Da das Tutorial linear abläuft, sollte verhindert werden, dass Spieler:innen das Spiel nicht fortführen können, weil ihnen ein Schritt des Tutorials nicht gelingt. Aus diesem Grund wurde ein Lösungsblatt erstellt. Dieses war explizit nur in dem Fall zu verwenden, dass ohne das Lösungsblatt auch nach längeren Versuchen ein Schritt des Tutorials nicht lösbar war. Die im Tutorial enthaltenen erklärenden Texte wurden überarbeitet, um die Inhalte möglichst präzise und leicht verständlich zu vermitteln.

Zusätzlich wurde ein Hinweis auf das Erreichen des Endes des für die Studie notwendigen Teils des Tutorials ergänzt. Spieler:innen, die weiter die Programmierung ihrer Einheiten verbessern und erweitern oder mehr Spielinhalte kennenlernen wollen, werden jedoch zum Weiterspielen ermutigt.

5 Methodik

In diesem Kapitel wird das methodische Vorgehen der Studie beschrieben. Dazu wird zunächst die Zielsetzung erläutert, bevor das Studiendesign, die eingesetzten Erhebungsinstrumente sowie die Auswertungsmethoden im Detail dargestellt werden. Die Methodik bildet die Grundlage für die spätere Auswertung und Diskussion der Ergebnisse und soll die Nachvollziehbarkeit der Studie sicherstellen.

5.1 Ziel der Studie

Ziel der Studie ist es, den Lernerfolg durch das Spielen des Prototyps zu evaluieren, um die Forschungsfrage zu beantworten, ob das Spiel effektiv für den Erwerb grundlegender Programmierkonzepte ist. Dazu wird das Verständnis grundlegender Programmierkonzepte, die im entwickelten Spiel vermittelt werden, in einem Pre-Post-Test geprüft. Die Untersuchung fokussiert sich auf folgende Konzepte:

- Reihenfolge der Ausführung von Programmcode
- *Solange*-Schleife
- *Wenn*-Anweisung
- Rückgabewerte

Andere Programmierkonzepte werden nicht berücksichtigt, um den Zeitaufwand für die Studienteilnahme zu begrenzen und/oder weil sie nicht ausreichend im Spiel erläutert werden.

5.2 Herausforderungen bei der Testentwicklung

Die Nutzung bestehender Tests stellte sich aus den folgenden Gründen als schwierig heraus. Alle Programmierkomponenten im Spiel sind auf Deutsch übersetzt (z. B. *While*-Schleife als *Solange*-Schleife), was unüblich ist. Um Studienteilnehmer:innen nicht zu verwirren, sollten die Bezeichnungen der Komponenten im Test und im Prototyp gleich sein. Der Umfang der vermittelten Programmierkonzepte ist im Spiel relativ begrenzt, weshalb keiner der untersuchten Tests exakt die im Spiel behandelten Konzepte abdeckt. Zuletzt musste der Zeitaufwand für die Studie begrenzt werden, um sicherzustellen, dass genügend Teilnehmer:innen diesen abschließen. Aus diesen Gründen wurde ein eigener Pre-Post-Test entworfen.

5.3 Eigener Testentwurf

Der Pre- und Post-Test wurde jeweils so gestaltet, dass:

- Operatoren eingesetzt werden, um sinnvolle Aufgaben zu ermöglichen, jedoch auf einfache und schulübliche Operatoren beschränkt (/, +, -, <, >, *). Spezielle, in Programmiersprachen übliche Operatoren wie ==, *=, += etc. wurden ausgeschlossen.

- Variablen eingesetzt werden, um sinnvolle Aufgaben zu ermöglichen, obwohl diese im Spiel nicht ausreichend erklärt werden. Die Verwendung von Variablen in den Tests entspricht allerdings der aus dem Mathematikunterricht bekannten Verwendung und wird somit als ausreichend verständlich eingeschätzt.
- Items im Pre- und Post-Test sind nur parallel und nicht identisch, um Gedächtniseffekte zu vermeiden.
- Antworten dichotom auswertbar sind.
- Kein Wissen über Spielmechaniken benötigt wird, um die Vergleichbarkeit der Ergebnisse sicherzustellen da im Pre-Test das Spiel noch nicht bekannt ist.
- Die verwendete Terminologie ausschließlich auf Deutsch ist, analog zur Spielumgebung.
- Abstrahierter Java-Code als Basis für die Aufgaben genutzt wird.
- Jede Programmierkonzept-Kategorie mit der gleichen Anzahl an Items vertreten ist.
- Alle Items dieselbe Anzahl an Antwortmöglichkeiten besitzen, um Vergleichbarkeit zu gewährleisten.
- Die erste Testfrage bewusst einfach gehalten ist, um den Teilnehmer:innen eine gewisse Sicherheit zu geben und frühe Motivationsverluste zu vermeiden.
- Die Zeit pro Frage auf etwa eine Minute veranschlagt wird.

Die Studie wird als Pilotierung verstanden, da es sich um die erstmalige Nutzung selbst entwickelter Items ohne vorherige Validierung handelt. Für die Studienteilnahme inklusive Pre-Test, Spielen des Tutorials und darauffolgendem Post-Test wird eine Dauer von ca. 30-40 Minuten veranschlagt.

5.4 Teilnehmer:innen

Um Teilnehmer:innen zu werben, wurde die Emil Krause Schule kontaktiert. An dieser wurde die Studie im Rahmen des Informatikunterrichts von zwölf Schüler:innen der Stufe 12 und 13 begonnen und von acht der zwölf Schüler:innen abgeschlossen. Zusätzlich wurden innerhalb der Hochschule Studienteilnehmer:innen geworben, von denen sieben Teilnehmer:innen mit wenig bis keinen Programmierkenntnissen die Studie abgeschlossen haben.

5.5 Erhebungsinstrumente

Das Spiel wurde über die Plattform Itch bereitgestellt und konnte dort von den Teilnehmer:innen gespielt werden. Vor dem Spielen füllten die Teilnehmenden einen Pre-Test-Fragebogen aus, um die vorhandenen Programmierkenntnisse zu erfassen. Nach dem Spielen erfolgte der Post-Test, um den Lernerfolg durch das Spiel zu messen. Die Fragebögen wurden mit Google Forms realisiert.

5.6 Auswertungsmethoden

Die Auswertung erfolgt quantitativ mittels SPSS (Version 31.0.0.0). Es sollen deskriptive Statistiken und inferenzstatistische Tests durchgeführt werden, um signifikante Unterschiede zwischen Pre- und Post-Test sowie zwischen den Gruppen (Schule, Hochschule) zu ermitteln. Da es sich um eine Pilotierung handelt, sollen Reliabilität der Items sowie Itemschwierigkeit ermittelt werden.

Vor der Auswertung wurden die Datensätze bereinigt. Antworten von Personen, die ausschließlich den Pre-Test, nicht jedoch den Post-Test bearbeitet hatten, wurden entfernt. Ebenfalls ausgeschlossen wurden Antworten von Personen, die bei der Selbsteinschätzung der Programmierkenntnisse auf einer Skala von 1 bis 5 die Stufen 3, 4 oder 5 angegeben hatten, da der Test sich an Personen mit wenig bis keinen Programmierkenntnissen richtet. Bei mehrfach ausgefüllten Tests wurde jeweils nur die erste Antwort der jeweiligen Person berücksichtigt.

5.7 Hypothesen

Es wird die folgende Hypothese aufgestellt. Diese soll mithilfe der Studienergebnisse belegt oder widerlegt werden, um die Forschungsfrage zu beantworten, ob der Prototyp effektiv für den Erwerb grundlegender Programmierkenntnisse ist.

Hypothese 1 – Vergleich der Ergebnisse im Pre- und Post-Test:

- H0: Es gibt keinen Unterschied zwischen Pre- und Post-Test-Ergebnissen in der Gesamtstichprobe sowie in den Gruppen Schule und Hochschule.
- H1: Die Post-Test-Ergebnisse sind signifikant höher als die Pre-Test-Ergebnisse in der Gesamtstichprobe sowie in den Gruppen Schule und Hochschule.

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Untersuchung dargestellt. Zunächst erfolgt eine deskriptive Auswertung der Daten, um einen Überblick über zentrale Lage- und Streuungsmaße sowie Verteilungen zu geben. Anschließend werden die inferenzstatistischen Analysen berichtet, die die Hypothesenprüfung ermöglichen. Abschließend wird die Item-Analyse durchgeführt, um zusätzliche Einblicke in die Eignung und Aussagekraft der eingesetzten Testinstrumente zu erhalten.

6.1 Deskriptive Statistik

Für die Analysen wurden ausschließlich dichotome Variablen der Pre- und Post-Tests verwendet. Zusätzlich wurde die Gruppenzugehörigkeit der Teilnehmenden (Schule/Hochschule) erfasst. Die Auswertung erfolgte somit für drei Datensätze: die Gesamtstichprobe, die Gruppe Schule und die Gruppe Hochschule.

6.1.1 Gesamtstichprobe

Die folgenden Tabellen geben die zentralen Lage- und Streuungsmaße der Summenscores aus Pre- und Post-Test wieder. Zusätzlich wurden Schiefe und Kurtosis berechnet, um Hinweise auf Abweichungen von der Normalverteilung zu erhalten. Boxplots ergänzen die grafische Darstellung (Abbildung 25, 26).

Tabelle 2: Deskriptive Statistiken der Summenscores der Gesamtstichprobe sowie der Gruppen Schule und Hochschule im Pre-Test. Quelle: Eigene Darstellung

	Gesamt	**Schule**	**Hochschule**
Mittelwert	6,87	5,63	8,29
Median	7,00	5,50	9,00
Standard-Abweichung	1,73	1,19	0,95
Minimum	4,00	4,00	7,00
Maximum	9,00	8,00	9,00
Interquartilbereich	4,00	1,00	2,00
Schiefe	-0,05	0,97	-0,76
Kurtosis	-1,40	1,87	-1,69

Tabelle 3: Deskriptive Statistiken der Summenscores der Gesamtstichprobe sowie der Gruppen Schule und Hochschule im Post-Test. Quelle: Eigene Darstellung

	Gesamt	**Schule**	**Hochschule**
Mittelwert	6,00	3,63	8,71
Median	5,00	4,50	10,00
Standard-Abweichung	3,36	2,13	2,21
Minimum	0,00	0,00	4,00
Maximum	10,00	6,00	10,00
Interquartilbereich	6,00	3,50	2,00
Schiefe	-0,21	-0,88	-2,08
Kurtosis	-1,05	-0,55	4,40

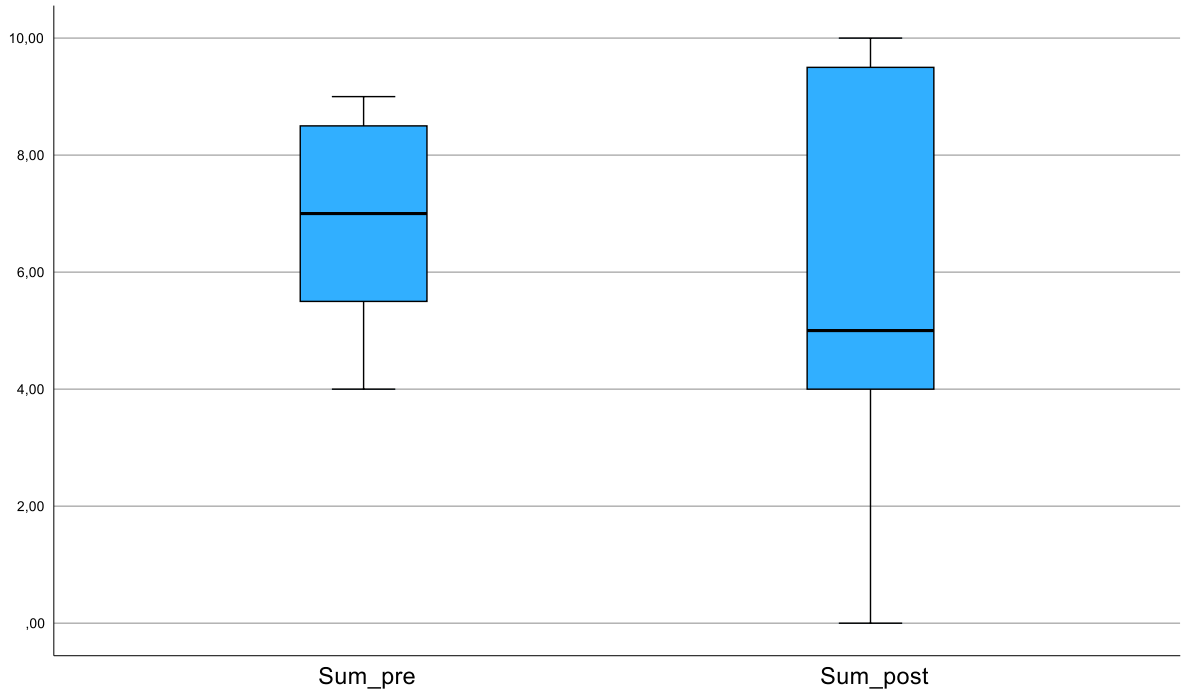


Abbildung 25: Box-Plot der Summenscores im Pre- und Post-Test der Gesamtstichprobe. Quelle: Screenshot aus SPSS

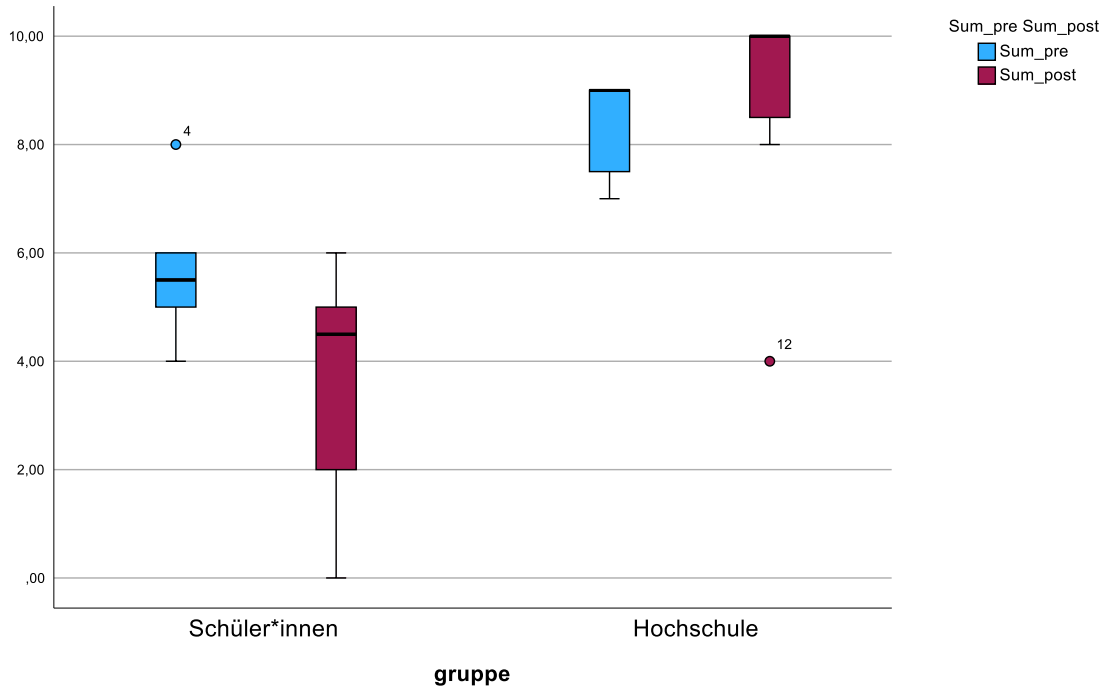


Abbildung 26: Box-Plot der Summensores im Pre- und Post-Test der Gruppen Schule und Hochschule. Quelle: Screenshot aus SPSS

6.1.2 Prüfung der Normalverteilung

Da die Stichprobengrößen unter 50 lagen, wurde der Shapiro–Wilk-Test (Shapiro & Wilk, 1965) zur Prüfung der Normalverteilung angewandt ($\alpha = 0{,}05$). Grundlage für die Berechnung waren die Differenzwerte zwischen Post- und Pre-Test.

Tabelle 4: Ergebnisse des Shapiro-Wilk-Tests für die Gesamtstichprobe sowie für die Gruppen Schule und Hochschule. Quelle: Eigene Darstellung

	Gesamtstichprobe	**Schule**	**Hochschule**
W	0,93	0,96	0,89
p	0,31	0,85	0,26

Die Überprüfung der Normalverteilungsannahme mittels Shapiro-Wilk-Test ergab für alle Variablen keine signifikanten Abweichungen ($W = 0{,}89$ bis $0{,}93$; $p = 0{,}26$ bis $0{,}85$). Werte nahe 1 der Teststatistik W, die Werte von 0 bis 1 annehmen kann, deuten laut Shapiro und Wilk (1965) auf eine geringe Abweichung von der Normalverteilung hin. Zudem liegen alle p-Werte deutlich über dem Signifikanzniveau von $\alpha = 0{,}05$, weshalb die Nullhypothese der Normalverteilung eindeutig nicht verworfen werden konnte. Die Differenzen wurden somit als normalverteilt betrachtet, weshalb für die weiteren Analysen ein t-Test für gepaarte Stichproben durchgeführt wurde.

6.2 Inferenzstatistik

Um zu überprüfen, ob die im Pre-Post-Vergleich beobachteten Unterschiede in den Mittelwerten der Summenscores nicht nur zufällig, sondern statistisch signifikant sind, wurden inferenzstatistische Verfahren angewendet. Die Berechnung der Effektstärke erfolgte mittels Cohens d. Cohens d gibt an, um wie viele Standardabweichungen sich die Mittelwerte der beiden Messzeitpunkte unterscheiden (Cohen, 1988). Die Korrektur nach Hedges wurde aufgrund der geringen Stichprobengröße ($n < 20$) angewendet, um eine Verzerrung der Effektgröße zu korrigieren (Hedges, 1985).

Tabelle 5: Ergebnisse des t-Tests für gepaarte Stichproben für die Gesamtstichprobe sowie für die Gruppen Schule und Hochschule. Quelle: Eigene Darstellung

	Gesamtstichprobe	**Schule**	**Hochschule**
***p* (zweiseitig)**	0,21	0,07	0,56
Cohens d mit Hedges' Korrektur	0,27, 95%-KI [−0,18; 0,81]	0,30, 95%-KI [−0,06; 1,35]	0,21, 95%-KI [−0,85; 0,46]

Zur Einordnung der Effektstärken wurde die gängige Klassifikation nach Cohen (1988) herangezogen: $d \approx 0{,}20$ (klein), $d \approx 0{,}50$ (mittel), $d \approx 0{,}80$ (groß). Dieser Einordnung nach sind die Effekte als klein zu bewerten. Die Ergebnisse sind zudem nicht signifikant ($p > 0{,}05$).

6.3 Item-Analyse

6.3.1 Pre-Test

Die Reliabilität im Pre-Test war mit Cronbach's Alpha = 0,37 gering und nach George & Mallery (2003) als inakzeptabel zu bewerten. Die Itemschwierigkeiten (Mittelwert der Lösungshäufigkeit) lagen zwischen 0,07 und 0,93 und variierten somit stark. Die Trennschärfen der Items (korrigierte Item-Skala-Korrelation) variierten stark (−0,57 bis 0,40). Eine vollständige Übersicht der Item-Werte findet sich in folgender Tabelle.

Tabelle 6: Übersicht der Itemstatistiken für Items des Pre-Tests. Quelle: Eigene Darstellung

Item (Pre-Test)	**M (Itemschwierigkeit)**	**Trennschärfe**	**Cronbachs Alpha, wenn Item weggelassen**
1	0,93	0,16	0,35
2	0,47	0,40	0,21
3	0,07	-0,57	0,50

4	0,27	-0,13	0,45
5	0,73	0,26	0,29
6	0,73	0,37	0,24
7	0,47	0,30	0,27
8	0,80	0,36	0,26
9	0,80	0,36	0,26
10	0,80	-0,36	0,51
11	0,80	0,24	0,31

6.3.2 Post-Test

Im Post-Test ergab sich eine deutlich höhere und nach George & Mallery (2003) als gut zu bewertende Reliabilität (Cronbach's Alpha = 0,83). Die Itemschwierigkeiten lagen zwischen 0,40 und 0,73 und variierten somit leicht. Die Trennschärfen waren überwiegend im akzeptablen bis guten Bereich und variierten (0,10 bis 0,76).

Eine vollständige Übersicht der Werte zu Items, Itemschwierigkeiten und Trennschärfen ist in folgender Tabelle dargestellt.

Tabelle 7: Übersicht der Itemstatistiken für Items des Post-Tests. Quelle: Eigene Darstellung

Item (Post-Test)	***M* (Itemschwierigkeit)**	**Trennschärfe**	**Cronbachs Alpha, wenn Item weggelassen**
1	0,73	0,67	0,80
2	0,40	0,24	0,84
3	0,47	0,41	0,82
4	0,60	0,57	0,81
5	0,60	0,10	0,85
6	0,53	0,51	0,82
7	0,73	0,51	0,82
8	0,40	0,57	0,81

9	0,47	0,61	0,81
10	0,47	0,76	0,79
11	0,60	0,63	0,81

6.4 Zusammenfassung der Ergebnisse

Die Analysen basierten auf den bereinigten Daten der Gesamtstichprobe sowie getrennt für die Gruppen „Schule“ und „Hochschule“. Es wurden ausschließlich dichotome Itemantworten (0 = falsch, 1 = richtig) berücksichtigt. Für die Gesamtstichprobe zeigte sich im Pre-Test ein Mittelwert von 6,87 richtigen Antworten, im Post-Test ein Mittelwert von 6,00. In der Gruppe Schule sank der Mittelwert von 5,63 auf 3,63, in der Gruppe Hochschule stieg der Mittelwert von 8,29 auf 8,71. Die Prüfung der Normalverteilung der Differenzen mittels Shapiro–Wilk-Test ergab in allen drei Stichproben keine signifikante Abweichung von der Normalverteilung ($p > 0,05$). Daher wurde jeweils ein t-Test für gepaarte Stichproben durchgeführt.

Die Ergebnisse des t-Tests zeigten für die Gesamtstichprobe keinen signifikanten Unterschied zwischen Pre- und Post-Test ($p = 0,21$). Auch innerhalb der Gruppen Schule ($p = 0,07$) und Hochschule ($p = 0,56$) ergaben sich keine signifikanten Unterschiede. Die berechneten Effektstärken nach Cohen’s d mit Hedges’ Korrektur aufgrund geringer Stichprobengröße waren klein bei $d = 0,27$ für die Gesamtstichprobe, $d = 0,30$ für die Gruppe Schule und $d = 0,21$ für die Gruppe Hochschule.

Die Itemanalysen ergaben für den Pre-Test ein Cronbach’s Alpha von 0,37 und für den Post-Test ein Cronbach’s Alpha von 0,83. Damit war die interne Konsistenz im Pre-Test als inakzeptabel einzustufen, während sie im Post-Test im guten Bereich lag. Die Items unterschieden sich hinsichtlich ihrer Schwierigkeit sowie ihrer Trennschärfe.

7 Diskussion der Ergebnisse

In diesem Kapitel werden die im vorherigen Abschnitt dargestellten Ergebnisse kritisch eingeordnet und im Hinblick auf die aufgestellte Hypothese sowie den theoretischen Rahmen dieser Arbeit diskutiert. Ziel ist es, die Befunde nicht nur statistisch, sondern auch inhaltlich zu interpretieren und mögliche Ursachen, Limitationen sowie Implikationen für die weitere Forschung aufzuzeigen.

7.1 Einordnung der Ergebnisse

Die Untersuchung hatte die Erwartung, dass die Ergebnisse des Post-Tests im Vergleich zum Pre-Test signifikant besser ausfallen würden (H1 – signifikante Verbesserung). Diese Hypothese konnte jedoch nicht bestätigt werden. Für die Gesamtstichprobe zeigte sich keine signifikante Verbesserung (p = 0,21). In der Teilgruppe Schule fielen die Ergebnisse im Post-Test tendenziell sogar schlechter aus (p = 0,07), wenn auch knapp nicht signifikant. In der Hochschulgruppe blieben die Ergebnisse stabil (p = 0,56) und bewegten sich auf einem insgesamt hohen Niveau. Die berechneten Effektstärken (Cohen's d mit Hedges' Korrektur) lagen zwar im kleinen bis mittleren Bereich, zeigten jedoch keine statistische Signifikanz. Dies deutet darauf hin, dass zwar Unterschiede messbar sind, diese aufgrund der geringen Fallzahl nicht zuverlässig auf die Grundgesamtheit übertragen werden können.

Die interne Konsistenz der Testinstrumente wurde mittels Cronbach's α überprüft. Während der Pre-Test eine sehr niedrige Reliabilität aufwies (α = 0,37), zeigte der Post-Test eine deutlich höhere Konsistenz (α = 0,83). Dies spricht dafür, dass die Items im Post-Test konsistenter miteinander zusammenhingen und die Ergebnisse verlässlicher sind als die Items im Pre-Test.

Zusätzlich wurde die Itemebene untersucht, um die Messqualität weiter zu prüfen. Die durchschnittliche Itemschwierigkeit im Post-Test war tendenziell höher als im Pre-Test, und die korrigierten Item-Skala-Korrelationen (Trennschärfen) zeigten überwiegend positive Werte. Das weist darauf hin, dass die Itemschwierigkeit zwischen Pre- und Post-Test besser ausbalanciert sein könnte, um eine bessere Vergleichbarkeit zu gewährleisten. Dies würde die Aussagekraft der Hypothese 1 (signifikante Verbesserung) erhöhen, da der Unterschied der Ergebnisse in Pre- und Post-Test nicht durch unterschiedlich schwere Items verzerrt werden würde. Problematische Items mit negativer Trennschärfe traten vor allem im Pre-Test auf, was auf Verbesserungspotenzial hinweist. Insgesamt zeigen die Ergebnisse, dass der Prototyp zwar gewisse Lernprozesse unterstützt, die Testinstrumente und Items jedoch weiter optimiert werden sollten, um valide Aussagen über Lernfortschritte treffen zu können.

7.2 Limitationen

Methodisch sind mehrere Einschränkungen zu berücksichtigen. Die Stichprobengröße war mit weniger als 20 Teilnehmenden gering, wodurch die statistische Power niedrig und die Konfidenzintervalle

breit ausfielen. Darüber hinaus war die Itemanzahl begrenzt, und einige Items wiesen schwache oder negative Trennschärfen auf. Hinzu kommt, dass Items im Pre- und Post-Test nicht vollständig identisch waren, was die direkte Vergleichbarkeit einschränkt. Die dichotome Auswertung berücksichtigt zudem keine graduellen Leistungsunterschiede. Die kurze Interventionsdauer von ca. 30-40 Minuten trug möglicherweise dazu bei, dass die Ergebnisse nicht signifikant und Effekte gering sind. Schließlich wurden nur jene Personen in die Analyse einbezogen, die beide Tests abgeschlossen haben, sodass mögliche Dropout-Effekte zu einer Verzerrung beigetragen haben könnten.

7.3 Interpretation möglicher Ursachen der Ergebnisse

Die Ergebnisse der deskriptiven Statistik, der Interferenzstatistik sowie der Reliabilitäts- und Itemanalyse werden folgend interpretiert.

Die tendenziell schlechteren Post-Test-Ergebnisse in der Gesamtstichprobe, aber insbesondere in der Gruppe Schule, könnten durch eine geringere Motivation, Ablenkungen im Klassenverband, erhöhte Itemschwierigkeit oder Ermüdung im zweiten Durchgang bedingt sein. Auf Motivationsverlust deutet auch die Tatsache hin, dass nur 8 von 12 Schüler:innen die Studie abgeschlossen haben. Problematische Items im Pre-Test (negative Trennschärfe) könnten zusätzlich die Vergleichbarkeit beeinträchtigt haben. Die geringen Effekte deuten zudem auf Verbesserungspotenzial der Vermittlung der Programmierkenntnisse im Prototyp hin.

Die stabileren Ergebnisse der Gruppe Hochschule deuten darauf hin, dass höhere Vorkenntnisse und eine individuellere Testsituation Schwankungen minimiert haben könnten.

Die niedrige Reliabilität im Pre-Test ($\alpha = 0{,}37$) erklärt, warum kein klarer Effekt messbar war, während der Post-Test ($\alpha = 0{,}83$) valide Informationen liefert. Die positive Entwicklung der Itemschwierigkeit und Trennschärfen im Post-Test unterstützt diese Einschätzung.

Insgesamt zeigt sich, dass sowohl die Testqualität als auch situative und motivational bedingte Faktoren die Ergebnisse beeinflusst haben könnten. Die Kombination der Hypothese H1 mit der Prüfung der Reliabilität sowie der Analyse der Itemschwierigkeit ermöglicht somit eine differenzierte Einordnung der Befunde und legt Ansatzpunkte für die Weiterentwicklung des Prototyps und der Testinstrumente nahe.

7.4 Einordnung in bestehender Forschung

Im Vergleich mit bestehenden Studien zur Wirksamkeit spielbasierter Lernumgebungen, die häufig von moderaten Lerngewinne berichten (vgl. z. B. Resnick et al., 2009; Gouws et al., 2013; Choi & Choi, 2023), konnten die erwarteten Verbesserungen hier nicht nachgewiesen werden. Eine mögliche Erklärung liegt in der geringen Reliabilität des Pre-Tests, die die Nachweisbarkeit von Lernfortschritten erheblich erschwert. Zudem entspricht es den Ergebnissen explorativer Pilotstudien, dass Effekte

bei kleinen Stichproben oft stark schwanken und nicht konsistent ausfallen (vgl. Cohen, 1988; Field, 2018).

Ein vergleichbares Ergebnis zeigt auch die Studie zu CodeCombat, einem textbasierten Lernspiel für Programmierkonzepte in der Sekundarstufe (Kroustalli, Xinagalos, 2021). In einem quasi-experimentellen Design mit Pre-/Post-Test zu Variablen und Schleifen in Python ($n = 59$) verbesserten sich sowohl die Spiel- als auch die Kontrollgruppe, wobei die Spielgruppe im Post-Test zwar leicht besser abschnitt, der Unterschied jedoch nicht signifikant war. Die Autor:innen führen dies insbesondere auf die kurze Interventionsdauer (ca. 135 Minuten), die begrenzte Stichprobengröße und die enge Anbindung an das Curriculum zurück.

Diese Begründungen lassen sich unmittelbar auf die vorliegende Pilotstudie übertragen. Auch hier führten die geringe Stichprobengröße und die kurze Einsatzdauer des Prototyps vermutlich dazu, dass trotz erkennbarer Tendenzen keine signifikanten Leistungssteigerungen nachweisbar waren. Damit bestätigt sich ein Muster, das auch in anderen Studien beobachtet wurde. Spielbasierte Lernumgebungen können das Verständnis grundlegender Konzepte unterstützen, benötigen jedoch für den Nachweis klarer Effekte größere Stichproben und längere Einsatzzeiträume.

Für die Informatikdidaktik lassen sich aus den Ergebnissen dennoch relevante Implikationen ableiten. Der entwickelte Prototyp hat grundsätzlich Potenzial, muss jedoch hinsichtlich der Motivationseffekte gezielt verbessert werden. Insbesondere die Ergebnisse der Schulgruppe deuten darauf hin, dass Schüler:innen stärker von motivationalen Faktoren abhängig sind als Studierende und daher besonders von spielerischen Verstärkungssystemen profitieren könnten. Gleichzeitig muss das Testinstrument selbst reliabler gestaltet werden, um valide Aussagen über Lernfortschritte treffen zu können.

8 Fazit

Diese Studie untersucht die Wirksamkeit eines Lernspiel-Prototyps zum Erwerb grundlegender Programmierkonzepte anhand eines Pre- und Post-Tests. Die zentrale Fragestellung lautet, wie effektiv der entwickelte Prototyp zum Erwerb grundlegender Programmierkenntnisse beitragen kann. Die Studie wurde als Pilotstudie entwickelt und von acht Schüler:innen der Stufe 12 und 13 sowie sieben Personen aus dem Hochschulkontext durchgeführt.

Die Forschungsfrage, ob der Prototyp effektiv zum Erwerb grundlegender Programmierkenntnisse beiträgt, kann nicht bestätigt werden, da signifikante Steigerungen der Testergebnisse ausblieben. Während ein stabiler Leistungsstand auf hohem Niveau in der Hochschulgruppe sichtbar wird, bleibt ein klarer Lernzuwachs insbesondere in der Schulgruppe aus. Dies verweist darauf, dass sowohl didaktische als auch motivationale Faktoren stärker berücksichtigt werden müssen, um die Wirksamkeit des Ansatzes zu erhöhen.

Ein wichtiger Befund ist die deutlich höhere Reliabilität des Post-Tests im Vergleich zum Pre-Test. Dies deutet darauf hin, dass die Qualität der eingesetzten Items verbessert wurde und künftige Untersuchungen auf einer solideren Messgrundlage aufbauen können. Zugleich verdeutlichen die Analysen auf Item-Ebene, dass die Testinstrumente noch weiter optimiert werden müssen, insbesondere durch die Anpassung problematischer Items und eine bessere Abstimmung der Schwierigkeit auf die Zielgruppe. Zukünftige Studien sollten zudem mit größeren Stichproben durchgeführt werden, um die statistische Power zu erhöhen. Zusätzlich wäre es sinnvoll, längere Interventionszeiträume zu wählen und neben reinen Testergebnissen auch das Engagement, die Motivation der Lernenden sowie ihre Einstellung gegenüber dem Programmieren systematisch zu erfassen.

Perspektivisch könnte der Prototyp so weiterentwickelt werden, dass er adaptive Rückmeldungen gibt, differenzierte Schwierigkeitsgrade berücksichtigt und stärker auf die Bedürfnisse verschiedener Gruppen eingeht. Darüber hinaus könnten Spielmechaniken stärker auf Motivationserhalt ausgerichtet werden, beispielsweise durch zusätzliche Gamification-Elemente. Designtechnische Aspekte des Prototyps könnten überarbeitet werden, um zuvor diskutierte, mögliche Probleme des Spiels zu beheben, wie z. B. das Fehlen des Konzepts von Variablen. Zudem könnte die kognitive Belastung weiter reduziert werden, um Spieler:innen den Einstieg zu erleichtern und Motivationsverlusten entgegenzuwirken.

Insgesamt liefert die Pilotstudie wertvolle Erkenntnisse über die Potenziale und Grenzen des entwickelten Spiels. Sie zeigt, dass ein spielbasierter Ansatz für den Informatikunterricht grundsätzlich vielversprechend ist, aber noch methodisch wie inhaltlich verfeinert werden muss, um sein volles didaktisches Potenzial zu entfalten.

Literaturverzeichnis

Alotaibi, M. S. (2024). Game-based learning in early childhood education: A systematic review and meta-analysis. *Frontiers in Psychology*, *15*, 1307881. https://doi.org/10.3389/fpsyg.2024.1307881

Anvik, J., Cote, V., & Riehl, J. (2019). Program Wars: A Card Game for Learning Programming and Cybersecurity Concepts. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 393–399. https://doi.org/10.1145/3287324.3287496

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable Programming: Blocks and Beyond. *Communications of the ACM*, *60*(6), 72–80. https://doi.org/10.1145/3015455

Bennedsen, J., & Caspersen, M. E. (2019). Failure rates in introductory programming: 12 years later. *ACM Inroads*, *10*(2), 30–36. https://doi.org/10.1145/3324888

Cohen, J. (with Internet Archive). (1988). *Statistical power analysis for the behavioral sciences*. Hillsdale, N.J. : L. Erlbaum Associates. http://archive.org/details/statisticalpower0000cohe_j0l3

Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011). From game design elements to gamefulness: Defining „gamification“. *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, 9–15. https://doi.org/10.1145/2181037.2181040

Gee, J. P. (2003). What video games have to teach us about learning and literacy. *Comput. Entertain.*, *1*(1), 20. https://doi.org/10.1145/950566.950595

George, D., & Mallery, P. (with Internet Archive). (2003). *SPSS for Windows step by step: A simple guide and reference, 11.0 update*. Boston : Allyn and Bacon. http://archive.org/details/spssforwindowsst00geor

Grover, S., & Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Hamari, J., Koivisto, J., & Sarsa, H. (2014). Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. *2014 47th Hawaii International Conference on System Sciences*, 3025–3034. https://doi.org/10.1109/HICSS.2014.377

Huizinga, J. (1987). *Homo ludens: Vom Ursprung der Kultur im Spiel*. Rowohlt.

Ishaq, K., Alvi, A., Haq, M. I. ul, Rosdi, F., Choudhry, A. N., Anjum, A., & Khan, F. A. (2024). Level up your coding: A systematic review of personalized, cognitive, and gamified learning in programming education. *PeerJ Computer Science*, *10*, e2310. https://doi.org/10.7717/peerj-cs.2310

Ishitani, L. (2012). *Motivational Factors for Mobile Serious Games for Elderly Users*. https://www.academia.edu/15764911/Motivational_Factors_for_Mobile_Serious_Games_for_Elderly_Users

Jordan, K. (2014). Initial trends in enrolment and completion of massive open online courses. *The International Review of Research in Open and Distributed Learning*, *15*(1). https://doi.org/10.19173/irrodl.v15i1.1651

Kiili, K. (2005). Digital game-based learning: Towards an experiential gaming model. *The Internet and Higher Education*, *8*(1), 13–24. https://doi.org/10.1016/j.iheduc.2004.12.001

Li, L., Hew, K. F., & Du, J. (2024). Gamification enhances student intrinsic motivation, perceptions of autonomy and relatedness, but minimal impact on competency: A meta-analysis and systematic review. *Educational Technology Research and Development*, *72*(2), 765–796. https://doi.org/10.1007/s11423-023-10337-7

Mayer, R. E., Fennell, S., Farmer, L., & Campbell, J. (2004). A Personalization Effect in Multimedia Learning: Students Learn Better When Words Are in Conversational Style Rather Than Formal Style. *Journal of Educational Psychology*, *96*(2), 389–395. https://doi.org/10.1037/0022-0663.96.2.389

Mühling, A., Ruf, A., & Hubwieser, P. (2015). Design and First Results of a Psychometric Test for Measuring Basic Programming Abilities. *Proceedings of the Workshop in Primary and Secondary Computing Education*, 2–10. https://doi.org/10.1145/2818314.2818320

Ochs, S. (2016, Juli 14). Meet Swift Playgrounds, the learn-to-code iPad app that feels like a puzzle game. *Macworld*. https://www.macworld.com/article/228397/meet-swift-playgrounds-the-learn-to-code-ipad-app-that-feels-like-a-puzzle-game.html

Ozaria Efficacy Report Mountain Ridge Middle School. (2021). Ozaria.

Ozaria Efficacy Report Summary McIntosh Middle School. (2021). Ozaria. https://files.ozaria.com/efficacy/Ozaria+Efficacy+Summary+Report+-+McIntosh+MS.pdf

Piaget, J. (with Internet Archive). (1970). *Science of education and the psychology of the child*. New York, Orion Press. http://archive.org/details/scienceofeducati00piag

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2014). Playful Learning: An Integrated Design Framework. https://doi.org/10.13140/2.1.4175.6969

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2015). Foundations of game-based learning. *Educational Psychologist*, *50*(4), 258–283. https://doi.org/10.1080/00461520.2015.1122533

Reigeluth, C. M. (Hrsg.). (1999). *Instructional-design theories and models: A new paradigm of instructional theory, Vol. II* (S. x, 715). Lawrence Erlbaum Associates Publishers.

Sailer, M., Hense, J., Mandl, H., & Klevers, M. (2013). Psychological Perspectives on Motivation through Gamification. *Interaction Design and Architecture(s)*, *19*, 28–37. https://doi.org/10.55612/s-5002-019-002

Shapiro, S. S., & Wilk, M. B. (1965). An Analysis of Variance Test for Normality (Complete Samples). *Biometrika*, *52*(3/4), 591–611. https://doi.org/10.2307/2333709

Tareque, M. H., Deutekom, S., Anvik, J., & Bashir, M. (2024). Program Wars v.2.0: Improving a Game-based Learning Approach for Teaching Fundamental Programming Concepts. *ACM International Conference Proceeding Series*, undefined-undefined. https://doi.org/10.1145/3660650.3660671

The University of Western Australia, & Cardell-Oliver, R. (2014). How Students Experience Learning to Program. *Education Research and Perspectives*, *41*, 196–216. https://doi.org/10.70953/ERPv41.14010

Unity Technologies. (2024). *Unity* (Version 6000.0.25f1) [Software].

Vygotsky, L. S. (1978). *Mind in Society: Development of Higher Psychological Processes*. Harvard University Press. https://doi.org/10.2307/j.ctvjf9vz4

Wing, J. M. (2006). Computational thinking. *Commun. ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215

Wood, D., Bruner, J. S., & Ross, G. (1976). The Role of Tutoring in Problem Solving. *Journal of Child Psychology and Psychiatry*, *17*(2), 89–100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x

Wouters, P., van Nimwegen, C., van Oostendorp, H., & van der Spek, E. D. (2013). A meta-analysis of the cognitive and motivational effects of serious games. *Journal of Educational Psychology*, *105*(2), 249–265. https://doi.org/10.1037/a0031311

Zainuddin, Z. (2018). Students' learning performance and perceived motivation in gamified flipped-class instruction. *Computers & Education*, *126*, 75–88. https://doi.org/10.1016/j.compedu.2018.07.003

Zhao, W., & Shute, V. J. (2019). Can playing a video game foster computational thinking skills? *Computers & Education*, *141*, 103633. https://doi.org/10.1016/j.compedu.2019.103633

Ludografie

Google. (2012). Blockly [Blockbasierte Programmierumgebung]. Google Developers. https://developers.google.com/blockly

MIT Media Lab. (2007). Scratch [Programmiersprache / Lernsoftware]. MIT Media Lab. https://scratch.mit.edu

Tomorrow Corporation. (2015). Human Resource Machine [Videospiel]. Tomorrow Corporation. Erschienen auf: Steam. https://store.steampowered.com/app/375820/Human_Resource_Machine/

Coffee Stain Studios. (2024). Satisfactory [Videospiel]. Coffee Stain Studios. Erschienen auf: Steam & Epic Games Store. https://www.satisfactorygame.com

Yaroslavski, D. (2008). LightBot [Lernspiel]. Coolio-NI (später LightBot Inc.). Erschienen als Webversion und Mobile-App. https://lightbot.com

Code.org. (2013). Code.org [Online-Lernplattform für Programmieren]. Code.org. https://code.org

CodeCombat. (2013). CodeCombat [Online-Lernspiel / Programmierlernplattform]. CodeCombat Inc. https://codecombat.com

Apple Inc. (2016). Swift Playgrounds [Lern-App / Programmierumgebung]. Apple Inc. https://apps.apple.com/app/swift-playgrounds/id908519492

Herzog, T. (2023). The Farmer Was Replaced [Videospiel]. Timon Herzog. Erschienen auf: Steam. https://store.steampowered.com/app/1293880/The_Farmer_Was_Replaced/

Wube Software. (2020). *Factorio* [Videospiel]. Wube Software. https://www.factorio.com

Anhang

Detaillierter Tutorial-Ablauf im Programmierspiel

1. Programmierung der Arbeitseinheit

- Die Aufgabe der Arbeitseinheit wird erklärt.
- Zunächst erfolgt die manuelle Markierung von Erz über einen speziellen Button.
- Der Editor sowie die Verwendung der Komponenten zur Programmierung und die Bedeutung der Reihenfolge der Ausführung werden erläutert.
- Die Komponente *Bewegen* wird eingeführt und eingesetzt.
- Die Komponente *Nächste Erzmarkierung* wird erklärt. Ihre Position wird in das Eingabefeld der Komponente *Bewegen* eingesetzt.
- Die Arbeitseinheit bewegt sich entsprechend der Programmierung zur Erzmarkierung.
- Die Funktion des Depots wird erklärt.
- Die Komponente *Erz abbauen* wird vorgestellt und eingesetzt.
- Die Komponente *Bewegen* wird erneut verwendet, diesmal mit dem Ziel *Nächstes Depot*.
- Ein Hinweis auf die Ressourcenleiste wird gegeben.
- Der Ablauf von Markierung setzen und Verhalten ausführen lassen wird wiederholt.

2. Erforschung einer neuen Komponente

- Das Baumenü wird vorgestellt.
- Ein Forschungslabor wird gebaut.
- Die Komponente *Solange* wird erforscht.
- Die Bedeutung und Nutzung der *Solange*-Komponente wird erläutert.
- Die Arbeitseinheit führt das Verhalten nun kontinuierlich in einer Schleife aus.

3. Späheinheit – Grundlagen

- Das Baumenü wird erneut zum Bau der Späheinheit verwendet
- Die Funktion der Späheinheit wird erklärt.
- Die Komponente *Erz markieren* wird vorgestellt.
- Die Komponente *Bewegen* wird mit dem Argument *Norden* eingesetzt.
- Die Späheinheit setzt automatisch Markierungen für Erzvorkommen.

4. Späheinheit – Erweiterte Funktionen

- Die *Wenn*-Komponente wird eingeführt, mit dem Argument *Über Erz*.
- Die Platzierung der Komponente *Erz markieren* innerhalb der *Wenn*-Komponente wird erläutert.

- Das vollständige Programm wird innerhalb einer *Solange*-Schleife platziert.

5. Abschluss des Tutorials

- Ein Hinweis auf den zweiten Fragebogen wird gegeben.

Pre-Test TerraCode

Dieser Pre-Test soll vor dem Spielen von TerraCode durchgeführt werden und dient dazu, deine aktuellen Programmierkenntnisse festzustellen.

* Gibt eine erforderliche Frage an

Allgemeine Daten

Alle Daten werden anonym verarbeitet.

1. Bitte gib hier deine E-Mail-Adresse an. Diese wird ausschließlich zur Verknüpfung von Pre- und Post-Test sowie zur Teilnahme an der Verlosung des 25€ Amazon-Gutscheins verwendet. *

2. Wie alt bist du? *

3. In welcher Klassenstufe befindest du dich? *

4. Was ist dein Geschlecht? *

   Markieren Sie nur ein Oval.

   - ◯ männlich
   - ◯ weiblich
   - ◯ divers
   - ◯ keine Angabe

5. Wie schätzt du deine Programmierkenntnisse ein? *

Markieren Sie nur ein Oval.

	1	2	3	4	5	
kein	◯	◯	◯	◯	◯	sehr gute Kenntnisse

Fragebogen zu Programmierkenntnissen

6. Was passiert bei folgendem Programmcode? *

```
Ausgeben("Los")
Ausgeben("Jetzt")
Ausgeben("Stopp")
```

Markieren Sie nur ein Oval.

- ◯ Es wird zuerst "Jetzt" ausgegeben.
- ◯ Die Reihenfolge ist nicht vorhersehbar.
- ◯ Es wird zuerst "Stopp" ausgegeben.
- ◯ Es wird zuerst "Los" ausgegeben.

7. Was macht diese Schleife? *

```
zähler = 0
    Ausgeben("Tick")
   zähler = zähler + 1
solange zähler < 2:
```

Markieren Sie nur ein Oval.

- ◯ Gibt nichts aus
- ◯ Gibt „Tick“ dreimal aus
- ◯ Gibt „Tick“ zweimal aus
- ◯ Gibt „Tick“ einmal aus

8. Was macht die Schleife? *

```
x = 0
  Ausgeben(„Tock“)
  x = x + 1
solange x > 2:
```

Markieren Sie nur ein Oval.

- ◯ Gibt „Tock“ unendlich oft aus
- ◯ Gibt nichts aus
- ◯ Gibt „Tock“ einmal aus
- ◯ Gibt „Tock“ zweimal aus

9. Was passiert hier? *

```
x = 5
   Ausgeben("Los geht's")
solange x < 3:
```

Markieren Sie nur ein Oval.

- ◯ "Los geht's" wird einmal ausgegeben
- ◯ "Los geht's" wird unendlich oft ausgegeben
- ◯ Es wird nichts ausgegeben
- ◯ Es gibt einen Fehler

10. Welche der folgenden Aussagen zur solange-Schleife ist korrekt? *

Markieren Sie nur ein Oval.

- ◯ Eine solange-Schleife wird genau einmal wiederholt.
- ◯ Eine solange-Schleife wiederholt sich immer unendlich oft.
- ◯ Sie wird wiederholt ausgeführt, solange die Bedingung wahr ist.
- ◯ Sie überspringt immer den ersten Durchlauf.

11. Was tut diese Bedingung? *

```
wenn Prüfen():
    Ausgeben("Aktion")
```

Markieren Sie nur ein Oval.

- ◯ Sie gibt "Aktion" immer aus.
- ◯ Sie gibt "Aktion" nur aus, wenn Prüfen() wahr zurückgibt.
- ◯ Sie gibt "Aktion" nie aus.
- ◯ Sie gibt "Aktion" in einer Schleife aus.

12. Was passiert hier? *

```
x = 3
wenn x < 5:
    Ausgeben("Treffer")
```

Markieren Sie nur ein Oval.

- ◯ Es wird einmal „Treffer" ausgegeben.
- ◯ Es gibt einen Fehler.
- ◯ Es wird zweimal „Treffer" ausgegeben.
- ◯ Es wird nichts ausgegeben.

13. Welche Aussage zur wenn-Anweisung ist richtig? *

Markieren Sie nur ein Oval.

- ◯ Sie wird immer ausgeführt.
- ◯ Sie prüft eine Bedingung und führt dann eventuell Code aus.
- ◯ Sie ersetzt eine solange-Schleife.
- ◯ Sie gibt einen Wert zurück.

14. Was ist der Zweck einer wenn-Anweisung in einem Programm? *

Markieren Sie nur ein Oval.

- ◯ Sie sorgt dafür, dass ein Codeblock immer mindestens einmal durchlaufen wird.
- ◯ Sie führt Code unabhängig von der Bedingung immer aus.
- ◯ Sie prüft, ob eine bestimmte Bedingung wahr ist, und führt den Code nur dann aus.
- ◯ Sie gibt automatisch einen Wert zurück.

15. Was kann eine Funktion zurückgeben? *

Markieren Sie nur ein Oval.

- ◯ Nur Zahlen
- ◯ Nur Wahr oder Falsch
- ◯ Unterschiedliche Arten von Werten, je nach Funktion
- ◯ Immer nur Text

16. Was bedeutet es, wenn eine Funktion „wahr“ zurückgibt? *

Markieren Sie nur ein Oval.

- ◯ Dass sie keinen Fehler hatte
- ◯ Dass etwas Bestimmtes zutrifft
- ◯ Dass eine Wiederholung gestartet wird
- ◯ Dass sie Text zurückgibt

Post-Test TerraCode

Dieser Test soll dein Verständnis grundlegender Programmierkonzepte testen, nachdem du TerraCode gespielt hast.

* Gibt eine erforderliche Frage an

1. Bitte gib hier deine E-Mail-Adresse an. Diese wird ausschließlich zur Verknüpfung von Pre- und Post-Test sowie zur Teilnahme an der Verlosung des 25€ Amazon-Gutscheins verwendet. *

2. Welche Befehle werden in folgendem Code in welcher Reihenfolge ausgeführt? *
   Bewege()

   Lade()

   DreheLinks()

   Markieren Sie nur ein Oval.

   - ◯ Lade, Bewege, DreheLinks
   - ◯ DreheLinks, Lade, Bewege
   - ◯ Bewege, Lade, DreheLinks
   - ◯ Die Reihenfolge ist nicht vorhersehbar

3. Wie oft wird der Befehl Lade() im folgenden Code ausgeführt? *

```
Lade()
   Lade()
solange falsch:
```

Markieren Sie nur ein Oval.

- ◯ Nie
- ◯ Einmal
- ◯ Zweimal
- ◯ Unendlich oft

4. Wie oft wird Ausgeben(x) ausgeführt? *

```
x = 1
    Ausgeben(x)
   x = x + 1
solange x <= 3:
```

Markieren Sie nur ein Oval.

- ◯ Zweimal
- ◯ Dreimal
- ◯ Gar nicht
- ◯ Unendlich oft

5. Was beschreibt eine solange-Schleife am besten? *

Markieren Sie nur ein Oval.

- ◯ Sie wird exakt einmal ausgeführt, egal was passiert
- ◯ Sie läuft nur, wenn die Bedingung falsch ist
- ◯ Sie wiederholt Anweisungen, solange eine Bedingung erfüllt ist
- ◯ Sie führt den Code erst ab dem zweiten Durchlauf aus

6. Was passiert in folgendem Code? *

```
x = 0
  Ausgeben(x)
  x = x + 1
solange x < 3:
```

Markieren Sie nur ein Oval.

- ◯ Gibt 0, 1, 2 aus
- ◯ Gibt 1, 2, 3 aus
- ◯ Gibt 0, 1, 2, 3 aus
- ◯ Führt eine Endlosschleife aus

7. Was wird ausgegeben? *

```
x = 10
wenn x > 5:
  Ausgeben("Groß")
Ausgeben("Klein")
```

Markieren Sie nur ein Oval.

- ◯ Groß
- ◯ Es wird nichts ausgegeben
- ◯ Klein
- ◯ Groß, Klein

8. Was macht eine wenn-Anweisung? *

Markieren Sie nur ein Oval.

- ◯ Sie wird auf jeden Fall ausgeführt, egal ob die Bedingung stimmt
- ◯ Sie überprüft eine Bedingung und führt den Code nur dann aus
- ◯ Sie wiederholt Code wie eine Schleife
- ◯ Sie liefert immer automatisch einen Rückgabewert

9. Was passiert in folgendem Code? *

```
wenn falsch:
    Ausgeben("Test")
Ausgeben("Kein Test")
```

Markieren Sie nur ein Oval.

- ◯ Nur "Test" wird ausgegeben
- ◯ "Test" und "Kein Test" werden ausgegeben
- ◯ Nichts wird ausgegeben
- ◯ Nur "Kein Test" wird ausgegeben

10. Was passiert, wenn ÜberErz() falsch zurückgibt? *

```
wenn ÜberErz():
    Lade()
```

Markieren Sie nur ein Oval.

- ◯ Lade() wird ausgeführt
- ◯ Lade() wird nicht ausgeführt
- ◯ Es gibt einen Fehler
- ◯ Der Programmcode wird von vorne ausgeführt

11. Wozu wird der Rückgabewert einer Funktion verwendet? *

Markieren Sie nur ein Oval.

- ◯ Um festzulegen, wie oft die Funktion aufgerufen wird
- ◯ Um einen Wert aus der Funktion an den Aufrufer zurückzugeben
- ◯ Um die Funktion wiederholt aufzurufen
- ◯ Um die Funktion sichtbar im Fenster anzuzeigen

12. Welche Werte kann eine Funktion generell zurückgeben? *

Markieren Sie nur ein Oval.

- ◯ Nur Zahlen
- ◯ Nur Wahr oder Falsch
- ◯ Unterschiedliche Arten von Werten, je nach Funktion
- ◯ Immer nur Text

Eigenständigkeitserklärung

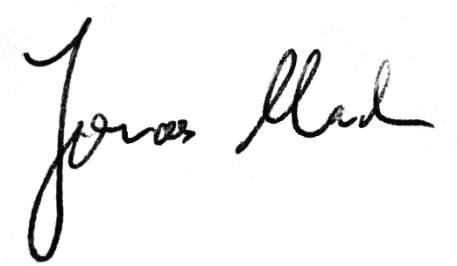
Hiermit versichere ich, dass ich die vorliegende Masterarbeit mit dem Titel:

Wie effektiv ist der für das Projekt „Learning By Playing“ entwickelte Prototyp für den Erwerb grundlegender Programmierkonzepte?

_____

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

29.09.2025



Datum

Unterschrift