MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

Die Effekte von Pair Programming im einführenden Programmierunterricht mit visuellen und textbasierten Programmiersprachen

verfasst von / submitted by

Mag. Patrick Korber, MSc, BEd

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Education (MEd)

Wien, 2020 / Vienna, 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 199 511 514 02
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Lehramt Sek (AB) Lehrverbund
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig

KURZFASSUNG

Ziel dieser Arbeit ist es, die Effekte des Einsatzes der Arbeitsform Pair Programming im einführenden Programmierunterricht einer textbasierten sowie einer visuellen Programmiersprache zu erforschen. Pair Programming wurde sowohl in praxisnahen Communities als auch in der wissenschaftlichen Forschung für unterschiedliche Zielgruppen im schulischen und außerschulischen Bildungsbereich als vielversprechende Arbeitsform für den Programmierunterricht beschrieben. Aufbauend auf den vorhandenen wissenschaftlichen Erkenntnissen für diesen Forschungsbereich sollen die spezifischen Unterschiede zwischen dem Einsatz von Pair Programming bei einer textbasierten und einer visuellen Programmiersprache empirisch untersucht werden. Dazu werden fachdidaktische Konzepte sowohl für eine visuelle Programmiersprache (MIT App Inventor) als auch für eine textbasierte Programmiersprache (Python) entwickelt und gemeinsam mit Schüler*innen einer AHS-Oberstufe umgesetzt. Im Anschluss werden die erstellten Programme, Fragebögen sowie Fokusinterviews mit ausgewählten Schüler*innen ausgewertet. Die Interpretation der empirischen Daten sowie die Einbettung der Ergebnisse in den theoretischen Diskurs sollen Rückschlüsse auf die Unterschiede zwischen textbasierten und visuellen Programmiersprachen im Hinblick auf die Effekte von Pair Programming zulassen und damit einen Beitrag zur Erforschung der Möglichkeiten, die sich durch den Einsatz von Pair Programming im Programmierunterricht bieten, liefern.

ABSTRACT

The aim of this work is to evaluate potential differences between text-based and visual programming languages in respect to the concept of pair programming as a teaching and learning method. Pair programming has been described as a promising approach for introductory programming lessons in both scientific research and practical communities. Based on existing scientific research, the specific differences between the use of pair programming in a text-based and a visual programming language are investigated. For this purpose, didactic concepts for both a visual programming language (MIT App Inventor) and a text-based programming language (Python) are developed and implemented. The empirical data is conducted through the evaluation of students' achievements, questionnaires and focus interviews with selected students. The evaluation of this data will be the basis for conclusions regarding the differences between pair programming with text-based and visual programming languages and thus contribute to the exploration of possibilities of making introductory programming lessons more successful.

INHALTSVERZEICHNIS

1. EINLEITUNG

1.1. Motivation und Forschungsproblem

Es herrscht mittlerweile ein breiter Konsens darüber, dass sich der Bildungswert der Informatik im Sinne einer neuen Kulturtechnik begründet. (vgl. Schubert und Schwill 2011: 117) Der Programmierunterricht ist ein wesentlicher Teil der informatischen Grundbildung, die vor allem auf *Algorithmisierung* als fundamentalem Konzept der Informatik basiert. (vgl. ebd. 112 ff.)

Der Einsatz von visuellen Programmierumgebungen wie *Scratch* oder *MIT App Inventor* wird oft als Garant eines positiven Lernerfolges für Programmier-Anfänger*innen genannt. (vgl. Kalogiannakis, Zaranis und Papadakis et al 2016: 230) Gleichzeitig ist es insbesondere im weiterführenden Programmierunterricht unabdinglich, auch mit textbasierten Programmiersprachen zu arbeiten, um den Anforderungen weiterführender Studien gerecht zu werden. Sowohl für textbasierte als auch für visuelle Programmiersprachen wurde das *Konzept Pair Programming*[1] als vielversprechender Ansatz sowohl in der Forschung als auch in praxisnahen Communities vielfach als Alternative oder Zusatzangebot zum *Single Programming* diskutiert. (siehe Hanks, Fitzgerald, McCauley et al 2011: 135 ff. für einen Überblick)

Pair Programming bedeutet, dass zwei Programmierer*innen kollaborativ an der Entwicklung einer Software arbeiten. (vgl. Cockburn und Williams 2000: 1 f.) Dabei nimmt eine Person die Rolle des *drivers* ein und schreibt das Programm, während die andere Person als *navigator* vielfältige Aufgaben wahrnimmt: „*One [job of the navigator] is to observe the work of the driver – looking for defects in the work of the driver. The navigator has a much more objective point of view and is the strategic, long-range thinker. […] An effective pair programming relationship is very active.*" (Williams, Wiebe, Yang et al 2002: 197 f.)

[1] Im Folgenden wird *Pair Programming* oft mit „PP", *Single Programming* mit „SP" abgekürzt.

Das Erkenntnisinteresse dieser Arbeit liegt in der Frage, ob Pair Programming in einer der beiden Bereiche (visuelle und textbasierte Programmierung) in unterschiedlichen Faktoren (z.B. Motivation, Effektivität oder Troubleshooting) gleich ausgeprägt ist oder nicht. Daraus erhoffe ich mir Erkenntnisse darüber, wie effektive und effiziente Lernszenarien für den einführenden Programmierunterricht mit Pair Programming gestaltet werden können.

1.2. Forschungsfragen und Hypothesen

Ziel meiner Arbeit ist es, die Effekte von Pair Programming im einführenden Unterricht einer visuellen Programmiersprache einerseits und einer textbasierten Programmiersprache andererseits miteinander zu vergleichen. Daraus ergibt sich folgende zentrale Forschungsfrage (ZF):

Forschungsfrage:

ZF: Wie unterscheiden sich die Effekte von Pair Programming als Arbeitsform im einführenden Unterricht einer textbasierten und einer visuellen Programmiersprache?

Um diese Forschungsfrage zu beantworten, wurden drei Teil-Forschungsfragen (TF1 bis TF3) definiert. Diese Teil-Forschungsfragen sollen das Erkenntnisinteresse aus unterschiedlichen Blickwinkeln zugänglich zu machen, um die Effekte von Pair Programming möglichst vollständig analysieren zu können.

TF1: **Inwiefern beeinflussen die spezifischen Herausforderungen von textbasierten und visuellen Programmiersprachen den erfolgreichen Einsatz von Pair Programming?**

TF2: **Wie unterscheiden sich die Erfahrungen von Schüler*innen im Hinblick auf die im Vortest erhobenen Leistungsniveaus?**

TF3: **Inwiefern unterscheiden sich die Erfahrungen im Pair Programming bei der Lösung von Problemen zwischen textbasierten und visuellen Programmiersprachen?**

Ausgehend vom Erkenntnisinteresse und den Forschungsfragen wurden Hypothesen entwickelt, die sich im Wesentlichen auf Ergebnisse vorhandener empirischer Studien zu den relevanten Forschungsbereichen (siehe Kapitel 2) beziehen:

H1: **Die Effekte von Pair Programming sind im einführenden Programmierunterricht mit textbasierten Programmiersprachen insgesamt stärker ausgeprägt als mit visuellen Programmiersprachen.**

H2: **Leistungsstarke Schüler*innen profitieren beim Pair Programming mit textbasierten Programmiersprachen stärker als bei visuellen Programmiersprachen.**

H3: **Für leistungsschwache Schüler*innen ist Pair Programming vor allem bei der textbasierten Programmierung hilfreich, um Probleme zu lösen.**

H4: **Die Effekte auf Effektivität und Effizienz von Pair Programming sind bei leistungsstarken und leistungsschwachen Schüler*innen bei der textbasierten Programmierung ähnlich ausgeprägt.**

Ich erwarte mir von den Resultaten dieser Arbeit Erkenntnisse über die Unterschiede zwischen textbasierten und visuellen Programmiersprachen in Hinblick auf die Effekte von Pair Programming im einführenden Programmierunterricht. Es ist zu erwarten, dass diese Effekte bei textbasierten Programmiersprachen deutlicher ausgeprägt sind, weil die Fehlerquellen (z.B. Syntaxfehler, Kohärenz der Variablen) vielfältiger sind. Gleichzeitig erwarte ich mir Erkenntnisse darüber, ob es divergierende Effekte in Bezug auf die Motivation der Lernenden gibt.

Insgesamt soll diese Arbeit einen Beitrag zur Erforschung der Möglichkeiten, die sich durch den Einsatz von Pair Programming im Schulunterricht bieten, beitragen und durch den Praxisbezug Erkenntnisse zur Effektivität konkreter fachdidaktischer Unterrichtskonzepte liefern.

1.3. Überblick und Ergebnisse

Diese Arbeit ist in vier Hauptteile gegliedert: Der theoretische Rahmen (Kapitel 2), die empirische Untersuchung (Kapitel 3), die Datenerhebung und -auswertung (Kapitel 4) sowie die Ergebnisse und Interpretation (Kapitel 5).

Der theoretische Rahmen in Kapitel 2 soll die theoretischen Grundlagen für die Beantwortung der Forschungsfragen aus Kapitel 1.2 liefern. Dazu wird in Kapitel 2.2 eine Systematische Literaturanalyse (SLR) durchgeführt, die einen möglichst vollständigen Überblick zum derzeitigen Forschungsstand liefern soll. In Kapitel 2.3 werden die relevanten qualitativen Forschungsmethoden zur Erhebung und Auswertung qualitativer empirischer Daten vorgestellt, anhand derer die empirische Untersuchung durchgeführt werden soll.

In Kapitel 3 wird die Durchführung der empirischen Untersuchung mit Schüler*innen eines Wahlpflichtfaches Informatik (6. Klasse AHS Oberstufe) beschrieben, die den Kern dieser Arbeit bildet. Zunächst werden in Kapitel 3.1 die Proband*innen sowie das Setting der empirischen Untersuchung beschrieben. In Kapitel 3.2 wird die Vorerhebung der Programmierkenntnisse vorgestellt, die einerseits zur Einteilung der Schüler*innen in Paare und andererseits als erster Indikator für das Leistungsvermögen der Schüler*innen dient. Die vier fachdidaktischen Konzepte zu einer visuellen Programmiersprache (MIT App Inventor) sowie einer textbasierten Programmiersprache (Python), jeweils für Single Programming und Pair Programming, werden in Kapitel 3.3 vorgestellt. Schließlich wird das methodologische Design für die Interviews in Kapitel 3.4 beschrieben.

Kapitel 4 bildet mit der Datenerhebung und Datenauswertung im Rahmen der durchgeführten Studie das empirische Fundament dieser Arbeit. Zunächst wird die Durchführung der Studie beschrieben (Kapitel 4.1). Die Auswertung der von den Schüler*innen erstellten Programme, der ausgefüllten Fragebögen sowie der Fokusinterviews wird daran anschließend in Kapitel 4.2 durchgeführt.

Die Auswertung der erhobenen Daten bildet schließlich gemeinsam mit den theoretischen Erkenntnissen aus Kapitel 2 die Grundlage für die Formulierung der Ergebnisse und der Interpretation in Kapitel 5, die in die Beantwortung der Forschungsfrage (Kapitel 5.1) sowie die Überprüfung der Hypothesen (Kapitel 5.2) unterteilt ist.

In Kapitel 6 werden diese Ergebnisse, wichtige Einschränkungen und notwendige weitere Forschungen (Kapitel 6.1) sowie Folgerungen und Auswirkungen dieser Ergebnisse (Kapitel 6.2) diskutiert.

Die Ergebnisse dieser Arbeit unterstützen die These, dass Pair Programming positive Effekte in Bezug auf das Erlernen einer Programmiersprache hat. Gleichzeitig weist die Auswertung der erhobenen Daten aus der empirischen Untersuchung auf beträchtliche Unterschiede zwischen dem Einsatz von Pair Programming bei textbasierten und bei visuellen Programmiersprachen hin, die beachtet werden müssen. Diese Unterschiede betreffen sowohl die Effekte von Pair Programming auf die Motivation als auch auf die Effektivität und die Problemlösungskompetenz der Lernenden, wobei diese Effekte bei leistungsschwachen Schüler*innen deutlich stärker ausgeprägt sind. Diese Erkenntnisse zeigen, dass die sorgsame Planung und Vorbereitung von Pair Programming Einheiten der Schlüssel zum Erfolg für gewinnbringende Lehr-Lernergebnisse ist. Diese Planung darf sich nicht nur auf fachliche Gesichtspunkte (wie etwa die Bereitstellung von Hilfsstrukturen) beschränken. Die Auswertung der Daten meiner empirischen Untersuchung zeigt, dass soziale Aspekte (wie etwa die Rolle der Kommunikation, eine positive Fehlerkultur und die Zusammensetzung der Paare) einen enormen Einfluss auf das Gelingen von Pair Programming Projekten haben.

Zusammenfassend weisen die Ergebnisse dieser Arbeit auf das große Potential von Pair Programming für den einführenden Programmierunterricht hin, wobei dies sowohl für textbasierte als auch für visuelle Programmiersprachen gilt. Um dieses Potential im Sinne der Lernenden optimal nutzen zu können, muss das Augenmerk bei der Entwicklung von Lernszenarien insbesondere auf die besondere soziale Situation gelegt werden, durch die sich Pair Programming auszeichnet.

2. THEORETISCHER RAHMEN

2.1. Einführender Programmierunterricht

Bevor ich in den nächsten Kapiteln auf die theoretischen Grundlagen für die Beantwortung der Forschungsfragen eingehen werde, soll an dieser Stelle in wenigen Worten die Bedeutung des einführenden Programmierunterrichts im Bildungswesen diskutiert werden.

Zunächst ist klar, dass der einführende Programmierunterricht für Schüler*innen die Möglichkeit bietet, das Berufsfeld Informatiker*in bzw. Programmierer*in näher kennenzulernen und somit neue berufliche Perspektiven eröffnen kann. Gerade in einer Zeit, wo in verschiedensten Branchen nach IT-Fachkräften gesucht wird, erscheint dieses Anliegen besonders bedeutend. (vgl. Geisler 2010: 198). Unabhängig von diesem Aspekt ist die Bedeutung des einführenden Programmierunterrichts als Teil einer informatischen Grundbildung aber auch für alle Schüler*innen elementar, für die das Berufsfeld Informatiker*in nicht interessant ist.

Ich möchte mich hier auf das von Wolfang Klafki formulierte Konzept *Erreichen von Allgemeinbildung durch epochaltypische Schlüsselprobleme* (siehe Klafki 2007) beziehen. Klafki schlägt vor, dass sich Bildungsinhalte an solchen Schlüsselproblemen – etwa an den Problemen Umwelt, Technikfolgen, Demokratisierung oder Menschenrechten – orientieren sollten, um so wesentliche Inhalte der Allgemeinbildung abzudecken. Für den einführenden Programmierunterricht ist besonders der Fokus auf das Schlüsselproblem der Globalisierung relevant, wo Klafki explizit auf die Bedeutung von Informationssystemen eingeht. (vgl. Klafki 2007: 80) Die Auseinandersetzung mit solchen Systemen als Teil der Schlüsselprobleme ist für Klafki die Basis dessen, was gemeinhin als Allgemeinbildung bezeichnet wird.

Auch die OECD bezieht sich in ihrer Ausarbeitung der Schlüsselkompetenzen für ein erfolgreiches Leben und eine funktionierende Gesellschaft auf den Wert der informatischen Bildung. Sie beschreibt dabei die „*Fähigkeit zur interaktiven Nutzung von Wissen und Information*“ (OECD 2015), die für den Informatikunterricht relevant ist. Die OECD hebt die Bedeutung der Rolle des Informationssektors in der heutigen Gesellschaft hervor: *„Diese Schlüsselkompetenz setzt eine kritische Reflexion über die Natur der Informationen als solche – ihre*

technische Infrastruktur sowie ihren sozialen, kulturellen und ideologischen Kontext und ihre Tragweite voraus.“ (OECD 2015: 13) Dabei sei es nicht ausreichend, sich auf grundlegende Fähigkeiten und Fertigkeiten wie etwa die Nutzung des Internets oder den Versand von E-Mails zu konzentrieren. (vgl. OECD 2015: 13) Erst darüberhinausgehendes Wissen wäre dazu geeignet, Technologie gewinnbringend einsetzen zu können. Diesem Gedanken folgend muss sich der einführende Programmierunterricht als Teil des Informatikunterrichts auch mit den Funktionsweisen von Informationssystemen beschäftigen, um dem Bildungsanspruch im Sinne von Klafki gerecht zu werden.

Für meine Arbeit zeigt die hohe Bedeutung, welche der Informatik – und des Programmierunterrichts als wichtigen Teil des Informatikunterrichts – von verschiedenen Seiten für die Allgemeinbildung zugedacht wird, wie wichtig funktionierende Konzepte für den einführenden Programmierunterricht sind. Konzepte wie Pair Programming können hierbei einen wichtigen Beitrag leisten, um diesen hohen Ansprüchen gerecht zu werden. Gerade deshalb erscheint es sinnvoll, die Faktoren für einen erfolgreichen Einsatz von Pair Programming im einführenden Programmierunterricht zu untersuchen und aus den Ergebnissen Schlüsse für die Praxis abzuleiten.

2.2. Systematische Literaturanalyse (SLR)

Ziel dieser Arbeit ist es, die Effekte von Pair Programming im einführenden Programmierunterricht von visuellen Programmiersprachen jenen von textbasierten Programmiersprachen gegenüberzustellen. Um den derzeitigen Forschungsstand der relevanten Disziplinen und Forschungsgebiete möglichst gut in diese Arbeit einzuarbeiten, eignet sich die Forschungsmethode der systematischen Literaturrecherche (*engl. systematic literature research, SLR*). Fink definiert diesen Ansatz als „[…] *a systematic, explicit and reproducible method for identifying, evaluating and synthesizing the existing body of completed and recorded work produced by researchers, scholars and practitioners* […] ". (Fink 2005: 3)

Im Folgenden werde ich die theoretischen Grundlagen, auf denen die SLR basiert sowie mögliche Probleme und Erweiterungen der Methode diskutieren. Darauf aufbauend werde ich die Methode auf mein Forschungsinteresse anwenden und abschließend einige Schlüsse für meine empirische Arbeit ableiten.

2.2.1 SLR als Forschungsmethode

Es gibt viele Gründe, SLRs in Forschungsarbeiten zur Informatikdidaktik anzuwenden. In Anlehnung an Kitchenham et al (2007) ist neben einer Zusammenfassung bereits vorhandener Forschungsergebnisse vor allem auch das Fehlen von empirischen Forschungen ein wichtiger Grund für die Durchführung einer SLR. Mit den Ergebnissen der SLR kann das geplante Forschungsvorhaben sehr viel effektiver und effizienter durchgeführt werden, weil bereits ein guter Überblick über mögliche Herausforderungen und Ressourcen geschaffen wurde. (siehe auch Fink 2005: 2-14)

Die Vorteile einer SLR (etwa gegenüber einer nicht-systematischen Recherche) liegen darin, dass durch die geplante und systematische Herangehensweise die Gefahr eines verzerrten Ergebnisses deutlich gemindert wird. (vgl. Kitchenham et al 2007) Gleichzeitig können die Ergebnisse der eigenen Forschung besser kategorisiert und kontextualisiert werden, was positive Effekte auf die Qualität der eigenen Arbeit zur Folge hat.

In diesem Zusammenhang ist es wichtig, die unterschiedlichen Terminologien zur SLR korrekt zu verwenden. Kitchenham et al (2007) weisen auf die Unterscheidung zwischen

„systematischer“ und „traditioneller/konventioneller“ Literaturrecherche hin. Dieser starren Unterscheidung der Kategorien widersprechen Okoli und Shabram (2010: 5): *„[W]e consider „systematic“ in „systematic literature review“ to be a qualitative adjective […] that describes the nature of a thing in a way that can be qualified by greater or less […]“.* Diesem Ansatz werde ich in meiner Arbeit folgen, wobei das Ziel natürlich eine möglichst systematische Vorgangsweise sein muss.

Durchführung einer SLR

Um den Anforderungen an eine SLR zu genügen, welche in Kapitel 2.1. diskutiert wurden, müssen von der Planung bis zur Durchführung im gesamten Prozess bestimmte Regeln befolgt werden. Okoli und Schabram (2010: 6) weisen darauf hin, dass es im Gegensatz zu anderen Disziplinen in der Informatik keine von allen Forschenden akzeptierte Vorgehensweise für die Durchführung einer SLR gibt, weswegen sie einen eigenen 8-Schritte-Guide vorschlagen.

Dieser Guide basiert auf den Vorarbeiten zu SLRs aus unterschiedlichen Disziplinen wie Software Engineering, Sozialwissenschaften, Management oder Organisationstheorie. (vgl. Okoli und Schabram 2010: 6) Abbildung 1 zeigt die acht Schritte in der Prozessansicht, wobei aus meiner Sicht für kleinere SLRs auch eine Unterteilung in vier Hauptschritte (links in der Graphik) ausreichend ist:

Planung

Dieser Schritt fasst die Definition der Ziele und Nicht-Ziele sowie die Vorbereitung der Mitarbeiter*innen (falls es mehrere gibt) zusammen – da es in meinem Fall keine weiteren Mitarbeiter*innen gibt, entfällt der zweite Teil.

Auswahl

In diesem Schritt geht es vor allem darum, eine möglichst vollständige und passende Auswahl der relevanten Literatur-Datenbanken zu gewährleisten. Gleichzeitig muss auch transparent gemacht werden, welche Beiträge aus welchen Gründen nicht berücksichtigt wurden.

Extrahierung

Von den ausgewählten Beiträgen werden in diesem Schritt diejenigen exkludiert, die den vorher definierten Qualitätskriterien nicht genügen. Die relevanten Informationen der ausgewählten Studien werden dann extrahiert und in passenden Formaten gespeichert.

Ausführung

Im letzten Schritt geht es um eine Analyse der extrahierten Daten mit passenden Methoden sowie um die Aufbereitung der Erkenntnisse in schriftlicher Form, um die Reproduktion der Ergebnisse für andere Forscher*innen zu ermöglichen.

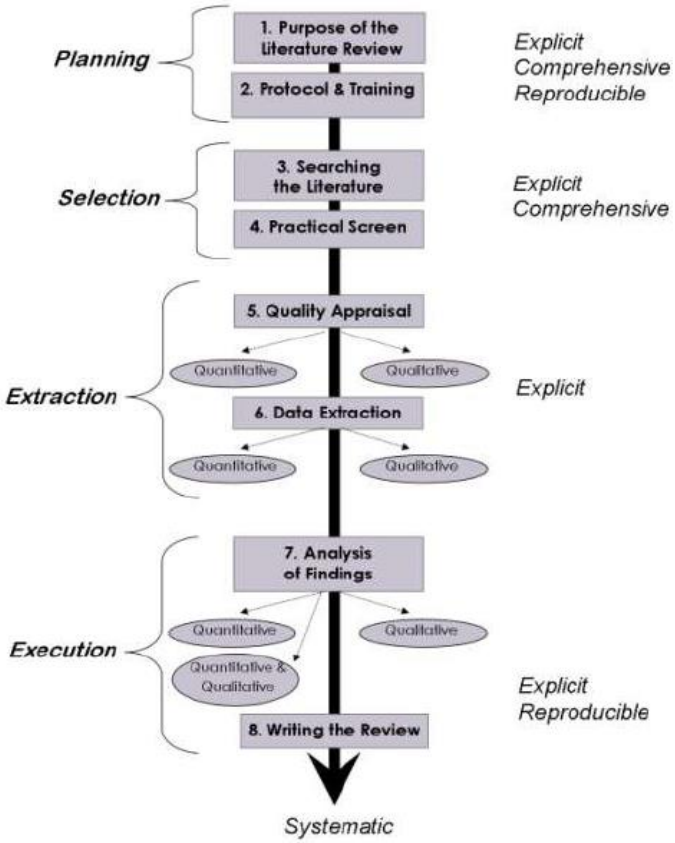


Abbildung 1: SLR in 4 Schritten

Quelle: Okoli und Schabram (2010: 9)

Aus meiner Sicht ist dieser Guide von Okoli und Schabram (2010: 6 ff.) sehr gut dafür geeignet, eine fundierte SLR zu meiner Fragestellung durchzuführen. Ich werde mich deshalb im Folgenden auf diese Vorgangsweise fokussieren, wobei auch Erkenntnisse anderer Arbeiten zur Durchführung von SLRs wie z.B. Kitchenham et al (2007) sowie Fink (2005) miteinbezogen werden.

2.2.2 SLR zu Pair Programming in Education

Um den Ansprüchen an den Prozess der Identifizierung, Auswahl und Interpretation relevanter Literatur zu meiner Fragestellung zu genügen, wurden die vier wesentlichen Prozesse nach Okoli und Schabram (2010) auf die SLR angewandt. Im Folgenden werde ich die Vorgangsweise in den jeweiligen Schritten darlegen, um im Anschluss ein Fazit für die Beantwortung meiner Forschungsfrage zu ziehen.

Planung der SLR

Der Fokus dieser Arbeit liegt auf den Effekten von Pair Programming im einführenden Programmierunterricht von visuellen Programmiersprachen sowie von textbasierten Programmiersprachen auf unterschiedliche Faktoren des Lernprozesses wie Motivation, Effektivität, Kommunikation oder Troubleshooting. Diese wesentlichen Aspekte müssen sich demnach natürlich in den Kriterien der Search-Strings für die ausgewählten Datenbanken wiederfinden.

Um möglichst vollständige Resultate zu erhalten, wurden verschiedene Schreibweisen und Kombinationen in einer Test-Suche ausprobiert und einige relevant erscheinende Artikel auf mögliche zusätzliche Schlagworte untersucht. So entstand eine Liste aus Schlagwörtern und Synonymen, die mittels der gängigen Boole'schen Operatoren verknüpft wurden. Verschiedene Search-Strings wurden dann getestet und folgender ausgewählt:

(Pair-Programming OR Pair Programming) AND (Classroom) AND (Efficiency OR Motivation)

Die Auswahl der passenden Datenbanken ist natürlich besonders wichtig für eine erfolgreiche SLR. Nach verschiedenen Test-Suchen und der Analyse bereits durchgeführter SLR zu ähnlichen Forschungsfragen (siehe z.B. Salleh et al 2011) wurden vier Datenbanken ausgewählt:

- ACM Digital Library
- IEEE Xplore / Electronic Library
- Lecture Notes in Computer Science (Springer)
- Google Scholar

Auswahl

Für die erste Auswahl und Eingrenzung der Literatur wurde der Search-String in jede der Datenbanken eingegeben. Dabei hat sich gezeigt, dass bei bestimmten Datenbanken eine weitere Eingrenzung notwendig ist, um eine überschaubare Anzahl an Ergebnissen zu erzielen. Durch das Kriterium „Abstract“ wird nach dem Vorkommen des Search-Strings im Abstract der Ergebnisse gefiltert, was bei den Datenbanken ACM, IEEEXplore und Google Scholar die Anzahl der Ergebnisse erheblich reduziert und – nach einer stichprobenartigen Überprüfung – die Relevanz der Ergebnisse enorm erhöht. In der Datenbank Lecture Notes in Computer Science musste der Search-String leicht verändert werden, da der Algorithmus den Begriff „Pair Programming“ nur mit Bindestrich als zusammengehörenden Begriff erkennt.

Auf diese Weise brachte eine erste Suche insgesamt 585 Ergebnisse, die in der folgenden Tabelle zusammengefasst sind:

Datenbank	**Such-String**	**Resultate**
ACM	[[Abstract: pair Programming] OR [Abstract: pair-Programming]] AND [Abstract: classroom] AND [[Abstract: success] OR [Abstract: motivation] OR [Abstract: efficiency]]	245
IEEEXplore	(("Abstract":Pair Programming) OR "Abstract":Pair-Programming) AND (Motivation)	29
Google Scholar	[[Abstract: pair Programming] OR [Abstract: pair-Programming]] AND [Abstract: classroom] AND [[Abstract: success] OR [Abstract: motivation] OR [Abstract: efficiency]]	66
Lecture Notes in Computer Science (Springer)	(Pair-Programming) AND (Classroom) AND (Efficiency OR Motivation)	142

Die weitere Auswahl erfolgte dadurch, offensichtlich nicht relevante Literatur auszuschließen sowie durch eine erste oberflächliche Analyse der Abstracts die Anzahl der Werke erheblich einzuschränken. Dazu wurden empirische Studien, die vor dem Jahr 2000 erschienen sind, ausgeschlossen sowie jene, die PP nicht im Bildungs-Kontext (sondern etwa im Software Engineering) untersuchten.

Durch diese Vorgangsweise wurde die Anzahl der Artikel auf 41 reduziert, die im Folgenden einer genaueren Analyse unterzogen wurden.

Extrahierung

In dieser Phase geht es darum, die ausgewählten Artikel hinsichtlich relevanter wissenschaftlicher Erkenntnisse zu unserer Forschungsfrage zu durchsuchen und gleichzeitig die Qualität der empirischen Ergebnisse zu hinterfragen. (vgl. Salleh et al 2011) Um die wissenschaftliche Qualität der Studien zu beurteilen, verwende ich eine einfache Punkte-Matrix, die an Salleh et al (2011: 513) angelehnt ist und einige wesentliche Merkmale wissenschaftlicher Qualität abdeckt:

Frage	**Antwort**	**Punkte**
Handelt es sich um ein „Peer-Reviewed Journal“?	Ja/Nein	1/0
Wird die Methodologie schlüssig erklärt?	Ja/Nein/Teilweise	1/0/0.5
Wird Durchführung der Studie schlüssig erklärt?	Ja/Nein/Teilweise	1/0/0.5
Sind die Ergebnisse bzw. deren Präsentation glaubwürdig?	Ja/Nein/Teilweise	1/0/0.5

Durchführung

Anhand der Qualitäts-Matrix wurden alle 41 Arbeiten systematisch analysiert, wobei der Fokus in dieser Phase nicht auf die Relevanz, sondern allein auf die wissenschaftlichen Qualität gelegt wurde. Dadurch ergibt sich folgende Übersicht der Ergebnisse:

Qualität	**Sehr schlecht (0-0.5)**	**Schlecht (0.5-1)**	**Mittelmäßig (1.5-2)**	**Gut (2.5-3.5)**	**Sehr gut (4)**	**Gesamt**
	0	1	6	26	8	41
	0%	2%	15%	63%	20%	100%

Thematisch wurden die analysierten Studien in vier Hauptgruppen eingeteilt, wobei der Übergang zwischen diesen Gruppen oftmals fließend ist. Anhand der Abstracts wurde eine Einteilung vorgenommen, in welchen Themenbereich die betreffende Studie am besten passt. Diese Hauptgruppen wurden aus der Forschungsfrage abgeleitet und stellen insofern relevante Kategorien für deren Beantwortung dar.

Ausgangspunkt waren die unterschiedlichen Effekte von Pair Programming, welche in folgende Kategorien untergliedert wurden:

- Effektivität und Effizienz
- Motivation
- Soziale Faktoren
- Lehren und Lernen

Das folgende Diagramm zeigt die Ergebnisse dieser Kategorisierung:

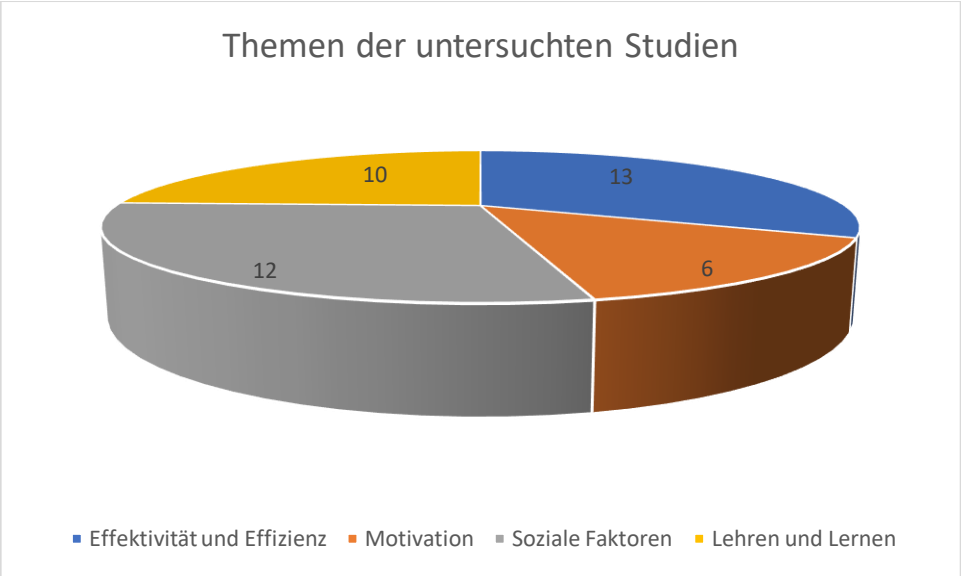


Abbildung 2: Themen der untersuchten Studien

Es zeigt sich, dass die ausgewählte Literatur die für die Forschungsfragen relevanten Themenbereiche relativ gleichmäßig behandelt. Am wenigsten Studien fanden sich zum Themenbereich „Motivation“, wobei viele Studien in der Kategorie „Soziale Faktoren“ auch (zumindest teilweise) den Faktor „Motivation“ miteinbeziehen.

Im Folgenden werden die Resultate für die einzelnen Kategorien überblicksartig zusammengefasst, um insgesamt Rückschlüsse für die Beantwortung der Forschungsfrage treffen zu können.

2.2.3 Ergebnisse der SLR

Effektivität und Effizienz

Der Großteil der analysierten Studien (8 von 13 oder 62%) sieht einen sehr deutlichen oder deutlichen positiven Effekt von PP auf die Effizienz und / oder Effektivität des Programmier-Unterrichts im Pair Programming. Diese Ergebnisse betreffen Studien zu älteren Lernenden (S10) und auch zu ganz jungen Lernenden (S1), verschiedene Schultypen (S3, S5, S9) sowie Querschnitts- und Langzeitstudien (S11). Eine einzige Studie (S1) sieht einen negativen Effekt auf die Effektivität des Programmierunterrichts. Interessant sind vor allem jene Studien (4 von 13 oder 31%), die einen differenzierten Blick auf die Effektivität und / oder Effizienz von PP wahrnehmen. Manche Autor*innen warnen etwa vor zu viel Enthusiasmus in Bezug auf PP im Schul-Kontext (S13, S2) oder zeichnen ein eher positives Bild, ohne die Effekte sehr signifikant zu bewerten (S13). Ein anderer Aspekt ist der Zusammenhang zwischen wahrgenommener und tatsächlicher Effektivität, wo es eine Diskrepanz geben kann (S6). So merkt etwa Faja (2014) folgendes an: „*Despite good intentions, these activities [PP] may not always have the desired outcomes of encouraging peer learning, increasing students' social skills, and enhancing student achievement.*" (Faja 2014: 41)

Folgende Tabelle zeigt zusammenfassend die Ergebnisse der analysierten Studien auf die Effekte von PP im Bereich Effektivität und Effizienz:

Positiv	**Negativ**	**Kein Einfluss**	**Unklares Ergebnis**
8	1	0	4

Diese Ergebnisse unterstützen im Wesentlichen jene Stimmen in der Forschung, die PP als zukunftsweisendes Konzept für den Programmierunterricht ansehen. Für diese Arbeit sind die analysierten Studien wichtige Quellen, um die Ergebnisse meiner Forschung mit vorhandenen wissenschaftlichen Studien zu kontextualisieren.

Einerseits zeigen die analysierten Studien insgesamt das große Potential, welches im PP als Arbeitsform für einen effektiven und effizienten Programmierunterricht liegt. Andererseits zeigen gerade die kritischen Stimmen, dass es zu keinem „Automatismus" im Sinne eines

erfolgreichen Programmierunterrichts durch PP kommt. Vielmehr müssen verschiedene (soziale, didaktische und technische) Voraussetzungen geschaffen werden, um die Potentiale von PP in der Praxis möglichst gut nutzen zu können.

Motivation

Die Literatur zum Themenbereich „Motivation" zeigt zum größten Teil klar positive Effekte von PP im Bildungs-Kontext. Vier von sechs Studien (67%) kommen in den verschiedenen Untersuchungs-Settings zum Schluss, dass sich PP deutlich positiv auf die Motivation von Lernenden auswirkt. Besonders positive Effekte auf die Motivation wurden bei älteren und fortgeschrittenen Lernenden mit Programmiererfahrung gezeigt (S16). Aber auch für Programmier-Anfänger*innen wurde etwa beim Einsatz von PP für graphische Programmierumgebungen gezeigt, dass sich PP positiv auf die Motivation auswirkt (S14).

Positiv	**Negativ**	**Kein Einfluss**	**Unklares Ergebnis**
4	0	0	2

Tabelle 4: Motivation der Lernenden und PP

Die Motivation von Lernenden wurde nicht nur in vergleichenden Ansätzen (mit Single Programming als Alternative) untersucht, sondern auch durch Messung der Drop-Out Raten von Programmierkursen, wo in S19 eine positive Korrelation zwischen PP und geringerer Drop-Out Rate nachgewiesen wird.

Besonders relevant erscheint mir die Untersuchung von PP aus dem Blickwinkel von Gender im Bildungskontext: In S18 wird in einer qualitativen Studie untersucht, wie PP die Programmiererfahrung von Mädchen beeinflusst. Die Ergebnisse postulieren einen sehr positiven Effekt von PP: „*They especially enjoyed the socialization and communication brought about by pair programming. The assistance, support, motivation, focus and encouragement they received from partners when stuck or while fixing errors made the programming experience more enjoyable for them.* ". (Liebenberg, Mentz und Breed 2012: 219)

In zwei von sechs Studien (33%) wird zwar kein negativer Effekt auf die Motivation festgestellt, jedoch zeigen diese Studien ein differenziertes Bild. In S15 wird keine klare Korrelation von PP und der Motivation der Telnehmer*innen gezeigt, allerdings wird darauf hingewiesen, dass noch weitere Studien in diesem Bereich notwendig seien. In S17 wird ein eher positiver Effekt von PP auf die Motivation festgestellt, wobei allerdings darauf hingewiesen wird, dass die Ergebnisse nicht eindeutig sind, wobei anekdotisch durchaus positive Effekte gesehen werden: *„The first major theme was that students liked having another programmer around with which to verbalize approaches to problem solving."* (Nagappan und Williams 2003: 18)

Für meine Masterarbeit zeigen diese Studien im Wesentlichen das Potential von PP im einführenden Programmierunterricht im Hinblick auf die Motivation der Lernenden, das für verschiedenste Bereiche des Bildungswesens nachgewiesen wurde. Für meine konkrete Forschung möchte ich aufbauend auf diese Ergebnisse mögliche Unterschiede zwischen textbasierten und graphischen Programmierumgebungen näher untersuchen.

Lehren und Lernen

In fast allen Publikationen zu Pair Programming kommen die Themen Lehren und Lernen in irgendeiner Form vor, da es bei PP ja in allen Kontexten um Problemlösungen geht. In den zehn Studien, die diesem Bereich zugeordnet wurden, geht es darüber hinaus ganz konkret um verschiedene Lehr-Lernaspekte von PP, die ich für meine Arbeit als besonders relevant empfunden habe.

In S20 wird PP ganz grundsätzlich auf das didaktische Potential im Hinblick auf die Anforderungen an Bildungsinstitutionen (z.B. Heterogenität, Kompetenzorientierung oder Employability) untersucht. Die Autor*innen stellen fest, dass PP insgesamt dazu geeignet ist, diesen Anforderungen gerecht zu werden, können aber in bestimmten Bereichen wie *Employability* oder *Motivation* keine klaren Schlüsse aus den Untersuchungen ziehen.

PP wird oft gemeinsam mit anderen derzeit propagierten Lehr-Lernstrategien wie kooperatives Lernen (S23) oder Flipped Classroom (S28) untersucht, wobei beide Studien die Kombination der Konzepte positiv bewerten. Die kooperative Methode Think-Pair-Share (TPS) wird in S24 als Strategie für das konzeptuelle Verstehen von Algorithmen gemeinsam

mit PP untersucht, wobei die Autor*innen eine positive Bilanz ziehen. In S27 wird besonders auf das Potential von PP für den weiterführenden Programmierunterricht und das Erlernen von komplexer struktureller Programmierung hingewiesen. Die Autor*innen stellen fest, dass PP gerade für jene Schüler*innen, die besondere Herausforderungen verlangen, positive Effekte auf die Motivation und die Fähigkeit haben kann.

Eine wichtige Dimension bei den didaktischen Potentialen von PP ist zweifellos die Zusammensetzung der Teams, die auch in einigen Studien im Bereich „Soziale Faktoren" untersucht wird. Im Bereich Lehren und Lernen wird in S29 untersucht, welchen Einfluss die Kompatibilität der Paare auf die Lernenden hat, wobei die Autor*innen festhalten, dass vor allem das proaktive Vorgehen der Lehrenden im Sinne der Formierung von kompatiblen Paaren essentiell sei. Im professionellen Kontext wird in S22 der Einfluss von dynamischen Teams als Erfolgsfaktor für PP gesehen, ein Aspekt, der durchaus auch in der Schule genauer untersucht werden sollte. Erfolgversprechend kann nach der Studie S25 auch die Formierung von *distributed pairs* sein, sofern es die technischen Voraussetzungen erlauben. In S26 werden ähnliche Ergebnisse, diesmal aber anhand einer visuellen Programmiersprache, dargestellt. Hier könnte eine Kombination mit *Flipped Classroom* besonders erfolgversprechend für den Einsatz in der Schule sein.

In S21 wird in einer Studie mit Studierenden untersucht, inwiefern PP das Teilen von gemachten Erfahrungen und Erkenntnissen (Knowledge Sharing) begünstigt, wobei positive Schlüsse gezogen werden: *„[P]air programming can be a useful approach […] to facilitate effective knowledge sharing among the students.*" (Kavitha und Irfan 2013: 319).

Für meine Masterarbeit schließe ich aus diesen Ergebnissen, dass insgesamt das didaktische Potential von PP für den Einsatz im einführenden Programmierunterricht eigentlich unbestritten ist, wobei auf bestimmte Faktoren wie die Zusammensetzung der Paare besonders geachtet werden muss. Auch die Verknüpfung von PP bekannten kooperativen Lehr-Lernformen erscheint mir besonders relevant für die Erkenntnisse, die ich mir von dieser Arbeit erhoffe.

Soziale Faktoren

Dem Themenbereich Soziale Faktoren wurden jene Studien zugeordnet, deren Erkenntnisinteresse sich hauptsächlich um die sozialen Bedingungen drehte, um PP im pädagogischen Kontext erfolgreich durchzuführen. Auch hier gab es natürlich zahlreiche Überschneidungspunkte mit anderen Bereichen, die ich im Einzelnen diskutieren werde.

Mehrere Studien untersuchten die Rolle der Kommunikation im PP-Prozess: In S31 wird in einer Untersuchung von Studierenden festgestellt, dass klare Richtlinien für die Kommunikation hilfreich sind, um zu positiven Ergebnissen zu kommen. Diese Erkenntnis scheint für jüngere Lernende noch wichtiger zu sein. In S35 wurde PP in einer Mittelschule untersucht, dabei merken die Autor*innen durchaus kritisch an, dass die Interaktionen zwischen den Lernenden nicht immer so waren, wie sich das die Lehrenden vorgestellt haben: „*The interactions were primarily not disruptive, but also not likely to involve the kinds of interactions that promote learning.*" (Campe, Denner, Green und Torres 2009: 20) Diese Erkenntnis wird auch in S36 geteilt, wo mit noch jüngeren Lernenden in einer Grundschule gearbeitet wurde. Die Autor*innen stellten Koordinationsprobleme sowie Herausforderungen für den kollaborativen Dialog fest – die Produktivität sei unter diesen Voraussetzungen unklar. In diesem Zusammenhang sind die Erkenntnisse aus S39 relevant, wonach diejenigen Lernenden die besten Ergebnisse erzielen, die es schaffen, Unsicherheit und Unklarheiten im Prozess klar auszudrücken und viel miteinander zu diskutieren. Aus dem gleichen Blickwinkel werden in S33 in einer Untersuchung die persönlichen Bedingungen für erfolgreiches PP untersucht, wobei die Autor*innen vielfältige Beiträge der Partner*innen (verbale und steuerungstechnische) untersuchen und zum Schluss kommen, dass bei einem Großteil der Paare die Arbeit gleichmäßig aufgeteilt wird. Dies erscheint deshalb besonders wichtig, weil es umgekehrt demotivierend sein könnte, wenn PP zu sehr ungleich verteilten Lasten innerhalb der Paare führt.

In fast allen Studien wird auf die Bedeutung der Team-Zusammensetzung hingewiesen. Für den Einsatz im einführenden Programmierunterricht erscheint vor allem S34 wichtig, wo festgestellt wird, dass Paare mit vergleichsweise wenig Erfahrung besonders erfolgreich sind. In S30 wird für eine Gruppe von Studierenden in diesem Zusammenhang der positive Einfluss von PP auf das Vertrauen in die eigenen Programmier-Fähigkeiten herausgearbeitet: Auch hier profitieren insbesondere jene Lernende, die vorher wenig Vertrauen in sich selbst

hatten. Auch wenn S30 keine Veränderung der Leistungen feststellt, wird PP insbesondere deshalb unterstützt, weil weibliche Studierende eher profitieren.

Ein relativer Konsens herrscht darüber, dass PP keine negativen Effekte für bestimmte Persönlichkeitstypen hat. In S40 wird dieser Umstand explizit formuliert, in S32 konnte durch qualitative und quantitative Untersuchungen festgestellt werden, dass Lernenden – unabhängig vom Vorwissen – durch PP mehr Vertrauen in die eigenen Fähigkeiten entwickeln konnten. In S41 wird in diesem Zusammenhang nochmals auf die Bedeutung der Zusammensetzung der Paare hingewiesen. In S37 wird zusätzlich festgestellt, dass auch der „Partnertausch" kein Problem darstellt und sogar – anders als man intuitive meinen könnte – positive Effekte haben kann: „*In fact, this aspect of the study seemed to strengthen some of the more difficult lessons related to code readability, using a coherent design, and using appropriate code documentation."* (DeClue 2003: 55)

Die Rolle von Gender beim Einsatz von PP wird durchaus kontrovers diskutiert: In S38 wird festgestellt, dass dieser Faktor derzeit noch nicht genügend untersucht wurde. Gleichzeitig wird festgehalten, dass die Rolle von Freundschaft und Gemeinschaft signifikante Auswirkungen hat: Je besser die Paare auf einer freundschaftlichen Basis arbeiten, desto besser sind die Resultate: „*On the one hand, good partnership helps to improve compatibility of PP as mentioned above. On the other hand, PP tightens up the partnership within a pair.*" (Zhong, Wang und Chen 2016: 429)

Für meine Masterarbeit schließe ich aus den Ergebnissen der SLR zu den sozialen Faktoren, dass für gelingendes PP die richtige Zusammensetzung der Paare essenziell ist. Gleichzeitig stimmt es zuversichtlich, dass es einen relativen Konsens zu positiven Effekten unabhängig von der Persönlichkeit der Lernenden gibt. Dies scheint mir insbesondere für den Entwurf der Lehr-Lernszenarios für textbasierte und visuelle Programmierung relevant.

2.2.4 Fazit der SLR

Ziel der SLR war es, vorhandene wissenschaftliche Erkenntnisse zu den Effekten von Pair Programming im Programmierunterricht systematisch zu sammeln und hinsichtlich ihrer Relevanz für meine Forschungsfrage kritisch zu analysieren. Dabei zeigte sich, dass das Thema grundsätzlich sehr gut erforscht ist: Für verschiedene relevante Effekte von PP gibt es

wissenschaftliche Studien für unterschiedliche pädagogische Kontexte. In einigen Bereichen herrscht ein relativer wissenschaftlicher Konsens, andere Bereiche werden kontroverser diskutiert oder sind noch ungenügend erforscht. Im Folgenden werde ich zusammenfassend einige Erkenntnisse aus der Durchführung der SLR beschreiben und die Relevanz für meine Arbeit herausarbeiten.

PP ist effektiv, aber nicht unbedingt effizient:

Die Analyse der Literatur aus dem Themenbereich Effektivität und Effizienz zeichnet insgesamt ein positives Bild vom Einsatz von PP für den Programmierunterricht. Bei genauerem Hinsehen (insbesondere auch bei Studien im Themenbereich *Soziale Faktoren*) zeigt sich aber, dass gerade bei jüngeren Lernenden PP zwar gut im Sinne der Zielerreichung (*Effektivität*) funktioniert, es aber stark auf die Kommunikation innerhalb der Paare ankommet, ob die Ziele auch mit angemessenem Aufwand (*Effizienz*) erreicht werden. Für mein Forschungsinteresse scheint dies insbesondere deshalb relevant, weil die Fehlerquellen von textbasierten Programmiersprache vielfältiger sind als bei visuellen Programmiersprachen (z.B. durch die Problematik von Syntaxfehler), wodurch dem Faktor Effizienz eine größere Bedeutung zukommt, um tatsächlich ein Programmierziel zu erreichen. Gleichzeitig gilt es zu beachten, dass die Messung der „Effizienz" von Lernprozessen sehr komplex ist und die Aussagekraft dementsprechend je nach Studiendesign uneindeutig sein kann.

PP hat positive Effekte auf Motivation und Selbstvertrauen:

Eines scheint als Fazit dieser SLR klar: Die große Stärke von PP liegt in der gesteigerten Motivation der Lernenden und dem höheren Vertrauen in die eigenen Fähigkeiten. Dieses Ergebnis wird auch in einer ähnlichen SLR von Salleh, Mendes und Grundy (2011) bestätigt: „*Almost all studies' findings reported that students' satisfaction was higher when using PP compared with working individually*". (Salleh, Mendes und Grundy 2011: 11)

Der kollaborative Austausch im Team, das gemeinsame Erarbeiten von Lösungsansätzen und die geteilte Freude bei Erfolgserlebnissen sind Erfolgsfaktoren dafür, dass Lernende weiterhin gerne am Programmierunterricht teilnehmen. Mich interessiert in diesem

Zusammenhang die Frage, ob sich die Erfahrungen der Lernenden in diesen Bereichen bei textbasierten und visuellen Programmiersprachen unterscheiden.

Die Zusammensetzung der Paare ist der Schlüssel zum Erfolg:

Die wichtigste Variable für den Erfolg von Pair Programming im einführenden Programmierunterricht scheint die Zusammensetzung der Paare zu sein. Während die Frage nach der Rolle von Gender bei der Zusammensetzung der Paare aus den vorhandenen Studien nicht endgültig geklärt ist, scheint die Bedeutung der persönlichen Beziehung sehr hoch zu sein: Paare, in denen sich die Lernenden gut verstehen, profitieren demnach besonders von PP. Gleichzeitig wird in mehreren Studien darauf hingewiesen, dass Lernende auch gemäß ihrem Vorwissen und ihren vorhandenen Fähigkeiten zugeteilt werden sollten. Lernende mit geringeren, aber ähnlich ausgeprägten Programmierkenntnissen von Pair Programming sind demnach besonders erfolgreich.

Hier gilt es natürlich, eine Balance zwischen den Faktoren – die ja zu sich widersprechenden Empfehlungen hinsichtlich der Einteilung in Paaren führen können – zu finden. Diese Balance versuche ich durch das Forschungsdesign, welches ich in Kapitel 3 vorstellen werde, zu erreichen. Besonders spannend ist dabei für mich die Frage, ob die Lernenden in diesem Zusammenhang das Programmieren mit einer visuellen Programmiersprache anders als mit einer textbasierten Programmiersprache erleben.

Vielfältige soziale Faktoren müssen beachtet werden:

Neben den besprochenen Faktoren gibt es noch viele andere, die in einzelnen Studien der SLR untersucht wurden und auf die ich, um den Umfang dieser Arbeit nicht zu sprengen, nicht einzeln eingehen kann. Wichtige Faktoren, die untersucht wurden, sind z.B. die Rolle von Gender in PP-Lernszenarien, die Auswirkungen von Stolz und Erfolgen, Misserfolgen, die Verteilung des Workloads innerhalb der Paare sowie der Einfluss von Partner*innenwechseln. Einen Konsens finden diese Studien in der Hinsicht, dass Pair Programming im Vergleich zu Solo Programming ein deutlich sozialerer Prozess ist, wodurch den Lehrenden eine besondere Rolle in der Steuerung dieses Prozesses zukommt. Einig sind sich fast alle Autor*innen darin, dass durch behutsame und überlegte Steuerung

(insbesondere bei der Zusammensetzung der Paare und der Förderung einer effektiven Kommunikation) zahlreiche positive soziale Effekte zu beobachten sind.

Ich werde versuchen, die Erkenntnisse aus diesen Bereichen in den Entwurf des Forschungsdesigns so einfließen zu lassen, dass die Lernenden möglichst von den genannten positiven Effekten profitieren können. Gleichzeitig erhoffe ich mir auch von Herausforderungen in der praktischen Umsetzung wertvolle Erkenntnisse für die Frage nach den unterschiedlichen Effekten von PP bei textbasierten und visuellen Programmiersprachen.

2.3. Qualitative Forschungsmethoden

Diese Arbeit legt den Fokus auf die individuellen Erfahrungen der Lernenden mit Pair Programming als Arbeitsmethode im einführenden Programmierunterricht. Die empirischen Daten in Bezug auf diese Erfahrungen sind dementsprechend vor allem qualitativer Natur. Neben den Daten zur Vorerhebung der Programmierkenntnissen sowie den Evaluierungen der Programmier-Projekte der Schüler*innen bilden teil-strukturierte Fokusinterviews den Kern des Forschungsdesigns für die empirische Untersuchung. Auf den folgenden Seiten werde ich erläutern, wie Fokusinterviews im Kontext meiner Arbeit durchgeführt werden und nach welcher Methode die Auswertung der empirischen Daten erfolgt.

2.3.1 Fokusinterviews

Leitfadengestützte Interviews sind eine sehr beliebte Methode der qualitativen Forschung: Sie sind methodologisch gut erforscht und es gibt zahlreiche Handbücher zur Erzeugung und Auswertung von qualitativen Daten mit dieser Methode. (vgl. Helfferich 2019: 669)

Als Untergruppe der leitfadengestützten Interviews bietet sich aus meiner Sicht das Fokusinterview für meine empirische Untersuchung an: Dabei wird die Offenheit der Fragestellung im Interview relativ stark eingeschränkt, um den Fokus des Interviews auf das wesentliche Erkenntnisinteresse zu lenken. Neben Filmen, Texten oder Bildern kann auch ein Experiment diesen Fokus bieten, der dann den Rahmen für die subjektiven Empfindungen des/der Interviewten bietet. (vgl. Przyborski und Wohlrab-Sahr 2008: 147) Im Kontext der Erforschung der Effekte von Pair Programming auf den einführenden Programmierunterricht bietet sich natürlich ein erstelltes Programm als Einstieg für das Fokusinterview an: Über visuelle und textbasierte Codebeispiele kann der Fokus auf die subjektive Wahrnehmung der Pair Programming Einheiten sowie der Solo Programming Einheiten gelegt werden. Diese Vorgangsweise wird in der Literatur meist als problemzentriertes Interview bezeichnet, wobei das Problem synonym zum Fokus verwendet wird. (vgl. Przyborski und Wohlrab-Sahr 2008: 146 f.)

Für den Einstieg in das Fokusinterview weist Helfferich (2019: 679) darauf hin, dass möglichst offene Fragen empfehlenswert sind, um die Äußerungen der Interviewten nicht einzuschränken. Die Vorteile dieser Interviewmethode liegt in der Möglichkeit, sehr

spezifische Informationen durch eine eher zurückhaltende, nicht-direktive Gesprächsführung zu bekommen. (vgl. Hopf 2000: 355) Gleichzeitig muss – wie bei allen qualitativen Interviews – darauf geachtet werden, dass die Strukturierung des Leitfaden genügend Raum für eigene und vielleicht unerwartete Äußerungen der Interviewten zulässt. (vgl. Helfferich 2019: 680)

Diese Balance erreicht man nach Helfferich (2019: 677), indem drei wesentliche Anforderungen von Leitfadeninterviews erfüllt werden:

1. **Offenheit**: Der gesamte Interviewprozess – inklusive Einschränkungen der Äußerungsmöglichkeiten – muss klar kommuniziert und begründet werden.
2. **Übersichtlichkeit**: Durch eine klare Struktur sollte das Interview aus Sicht der Interviewten übersichtlich bleiben, ohne die Erzählzeit zu sehr zu beschränken.
3. **Anschmiegen an den Erzählfluss**: Der Leitfaden muss so gestaltet werden, dass der erwartete Erzählfluss nicht durch Sprünge oder Themenwechsel gestört wird.

Im Hinblick auf mein Forschungsinteresse werde ich das Fokusinterview so gestalten, dass die zentrale Erzählaufforderung sich um die subjektive Wahrnehmung von PP sowie SP als Arbeitsform dreht. Die Unterschiede zwischen einer textbasierten und einer visuellen Programmiersprache werden den zweiten Fokus bilden.

2.3.2 *Qualitative Inhaltsanalyse*

Für die Auswertung qualitativer Daten stehen zahlreiche Methoden zur Verfügung, wobei darauf geachtet werden muss, dass die gewählte Methode auch zum Forschungsvorhaben passt.

Eine gut strukturierte Methode ist die qualitative Inhaltsanalyse: Hier wird das Datenmaterial in Textform kodiert und anhand klarer Regeln Schritt für Schritt kategorisiert und ausgewertet. (vgl. Mayring 2010: 114-121) Die von Philipp Mayring maßgeblich entwickelte Methode erweitert somit die quantitative Inhaltsanalyse und zeichnet sich durch eine fundierte Theorie für die Analyse und Interpretation von qualitativen Interviews aus. Durch die klare Strukturierung wird das Vorgehen übersichtlich und für andere nachvollziehbar.

Ein zentrales Element der qualitativen Inhaltsanalyse ist die Entwicklung eines Systems von Kategorien, das wie ein „Suchraster" strukturiert ist. Ziel dieses Rasters ist es, die für die Beantwortung der Forschungsfragen relevanten Aspekte aus den qualitativen Interviews zu filtern.

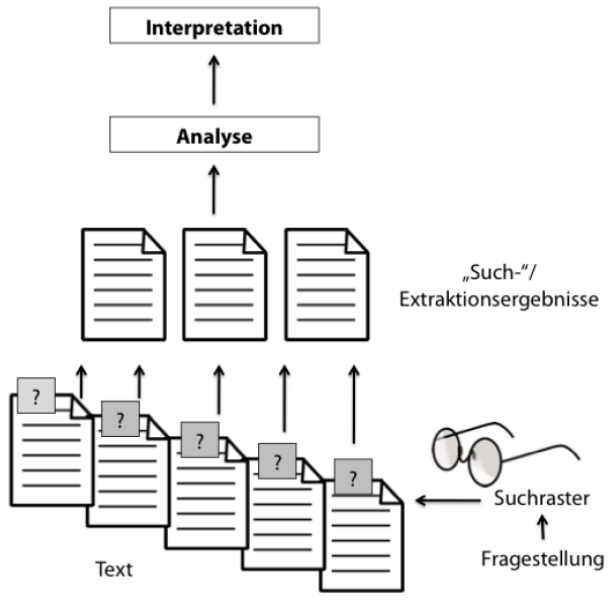


Abbildung 3: Qualitative Inhaltsanalyse

Quelle: Gläser und Laudel, 2009: S. 200

In Abbildung 1 ist der Prozess der qualitativen Inhaltsanalyse schematisch dargestellt: Anhand der Forschungsfrage(n) wird das Suchraster entwickelt und auf die transkribierten

Interviews angewandt. Daraus ergeben sich Extraktionsergebnisse, die analysiert und in weiterer Folge interpretiert werden – diese Interpretationen bilden die Basis für die Beantwortung der Forschungsfrage. (vgl. Gläser und Laudel 2009: 199 ff.)

Mayring (2002) schlägt fünf Schritte vor, um Daten anhand der qualitativen Inhaltsanalyse auszuwerten:

1. **Daten erkunden:** Vor der eigentlichen Analyse ist es empfehlenswert, die transkribierten Texten überblicksartig zu erfassen und sich Notizen zu Auffälligkeiten zu machen.
2. **Erstellung der Kategorien:** Das Kategoriensystem bildet die Basis für die Auswertung, weshalb dieser Punkt besonders wichtig ist. Nach Mayring (2010) können Kategorien sowohl deduktiv als auch induktiv gebildet werden, wobei beide Formen kombiniert werden können. Eine sinnvolle Vorgehensweise ist es oftmals, Hauptkategorien deduktiv (vor den Interviews) zu bilden und nach der Erfassung des Datenmaterials Unterkategorien induktiv zu bilden.
3. **Codierung der Interviews:** In diesem Schritt werden die Interviews Zeile für Zeile durchgearbeitet und relevante Textstellen den passenden Kategorien zugeordnet.
4. **Auswertung:** Um aus den codierten Textstellen einen wissenschaftlichen Text zu machen, werden die Ergebnisse je Kategorie ausgewertet und interpretativ in die Ergebnisse des theoretischen Kontextes eingearbeitet bzw. diskutiert.
5. **Diskussion:** Ganz zum Schluss werden die Ergebnisse zusammenfassend diskutiert. Ziel ist es dabei, die Forschungsfrage anhand einer Verknüpfung der verwendeten Theorien mit den Ergebnissen der qualitativen Inhaltsanalyse zu beantworten.

Für meine empirische Untersuchung wird die Kategorienbildung anhand der Forschungsfragen deduktiv erfolgen, wobei weitere induktive Kategorien hinzugefügt werden können, wenn das notwendig ist.

3. EMPIRISCHE UNTERSUCHUNG

Um die Forschungsfrage zu den Unterschieden von Pair Programming bei textbasierten und visuellen Programmiersprachen beantworten zu können, wurde eine empirische Untersuchung durchgeführt. Um die Vorkenntnisse und Erfahrungen der Schüler*innen einschätzen zu können, wurde zunächst eine Vorerhebung der Programmierkennisse durchgeführt (Kapitel 3.2). Dies war auch aus den Erkenntnissen der SLR (Kapitel 2.2) notwendig, um neben den freundschaftlichen Aspekten auch die Programmierkenntnisse bei der Bildung der Paare zu berücksichtigen. (vgl. Zhong, Wang und Chen 2016)

Die Basis der empirischen Untersuchung bildet ein fachdidaktisches Konzept für die Programmiersprache Python (zwei Einheiten PP und zwei Einheiten SP) sowie ein fachdidaktisches Konzept für den MIT App Inventor (zwei Einheiten PP und zwei Einheiten SP), das die Schüler*innen größtenteils selbstständig bearbeiteten (Kapitel 3.3). Nach den Einheiten füllten die Schüler*innen einen Fragebogen zu ihren persönlichen Erfahrungen mit Pair Programming und Solo Programming während dieser Einheiten aus. Mit einigen Paaren wurden im Anschluss an die letzte Einheit teilstrukturierte Interviews geführt, mit denen zusätzliche Daten zu den unterschiedlichen Erfahrungen von PP und SP erhoben werden sollten (Kapitel 3.4).

Im Folgenden werde ich auf die einzelnen Elemente der empirischen Untersuchung eingehen, um im Anschluss den Prozess der Datenanalyse zu beschreiben.

3.1. Beschreibung der Proband*innen

Die Untersuchung wurde mit einer Gruppe des Wahlpflichtfaches Informatik (6. Klasse) am BG/BRG Biondekgasse Baden im Februar und März 2020 durchgeführt. Sämtliche Forschungstätigkeiten im Rahmen des eigenen Unterrichts wurden mit der Direktion abgesprochen. Nachdem die Schüler*innen sich bewusst für dieses Wahlpflichtfach entschieden haben, kann von einem grundsätzlichen Interesse am Fach Informatik ausgegangen werden. Es handelte sich um 14 Schüler*innen zwischen 15 und 17 Jahren, von denen 10 Schüler*innen ihr Geschlecht mit „männlich“, 4 Schüler*innen mit „weiblich“ und 0 Schüler*innen mit „divers“ angegeben haben.

Alle Schüler*innen haben bereits Programmiererfahrungen mit der Programmiersprache „Scratch“ gesammelt, wobei einige sich nur wenige Stunden damit beschäftigt haben. Niemand hat angegeben, vor der Teilnahme am Wahlpflichtfach mit einer textbasierten Programmiersprache gearbeitet zu haben. Alle Schüler*innen haben zugestimmt, sich auf die Untersuchung einzulassen und freiwillig an der Befragung (Fragebogen und eventuell leitfadengestütztes Interview) teilzunehmen.

3.2. Vorerhebung der Programmierkenntnisse

Um eine möglichst valide Einschätzung der Programmierkenntnisse treffen zu können, die einerseits wichtige Daten für die Auswertung liefert und andererseits die Basis für die Einteilung der Paare beim PP bildet, nahmen die Schüler*innen an einem Algorithmus-Quiz teil. Dieser Quiz bestand aus sechs Fragen mit je vier Antwortmöglichkeiten (Single Choice) aus zwei Kategorien: Drei Fragen sind dem Online-Informatikwettbewerb „Biber der Informatik“[2] (Schulstufen 11 – 13) entnommen und behandeln allgemeines algorithmisches Denken. Die restlichen drei Fragen basieren auf einem Programm in der Programmiersprache „Scratch“, welches von mir erstellt wurde. Dabei wurde darauf geachtet, dass die drei Fragen jeweils auch drei „Schwierigkeitsstufen“ (leicht-mittel-schwer) entsprachen.

3.3. Fachdidaktisches Konzept

Um Daten für die Untersuchung der Forschungsfrage zu erhalten, wurde je ein fachdidaktisches Konzept für Python (einmal für PP, einmal für SP) sowie für den MIT App Inventor (einmal für PP, einmal für SP) ausgearbeitet. Alle Konzepte basieren auf explorativem, selbstgesteuertem Lernen. Diese didaktische Vorgangsweise basiert auf der Lerntheorie des Konstruktivismus nach J.S. Bruner, wobei es unter verschiedenen andere Bezeichnungen, die ähnliches beschreiben, in didaktischen Kontexten bekannt ist.

---

[2] Dieser Informatik-Wettbewerb wird in Österreich von der OCG durchgeführt, siehe https://www.ocg.at/de/biber-der-informatik [Abruf: 05.05.2020]

Neber (1981: 45 ff.) fasst verschiedene Lernformen, darunter *induktives Lernen*, *forschendes Lernen* und *aktives* bzw. *problemorientiertes Lernen* zu explorativem Lernen zusammen. Beim explorativen Lernen steckt sich der/die Lernende selbst das Lernziel. Er oder sie versucht, durch bestimmte Handlungen dieses Lernziel zu erreichen, dabei ist der Lernprozess aber nicht linear und es ist normal, dass während des Prozesses verschiedene Fragen auftauchen, die noch nicht geklärt sind. (vgl. Kerres 2001: 217) Ein Merkmal von explorativen Lernprozessen ist, dass besonders oft die Aktivitäten an sich – und nicht nur die Ergebnisse – von den Lernenden als befriedigend erlebt werden.

Grundsätzlich ist es gerade in explorativen Lehr-Lernsettings wichtig, die Schüler*innen zur aktiven Teilnahme zu motivieren – die vier Konzepte behandeln deshalb spielerische bzw. praxisnahe Themen, die eng mit der Lebenswelt der Schüler*innen verknüpft sind.

Die Konzepte wurden so gestaltet, dass sich die Schüler*innen grundsätzlich selbst mit den gestellten Anforderungen beschäftigen sollen. Die Lehrperson steht für Verständnisfragen und Tipps zur Verfügung, bleibt aber in der Rolle des Lernbegleiters und steuert den Unterricht nicht durch angeleitete Vorgaben. Um eine Überforderung der Schüler*innen zu vermeiden, wurden im Sinne eines *scaffolding*-basierten Programmierunterrichts (siehe dazu Nam, Kim und Lee 2010) für jedes Tutorial verschiedene Hilfsstrukturen wie Tipps oder erwartete Ergebnisse bereitgestellt. Im Rahmen der Vorerhebung der Programmier-Kenntnisse wurde ermittelt, dass die Schüler*innen bereits mit wichtigen Konzepten wie Schleifen, Bedingungen und Variablen vertraut waren. Aus diesem Grund wurden die fachdidaktischen Konzepte so gestaltet, dass diese Konzepte als bekannt vorausgesetzt wurden.

3.3.1 Visuelle Programmierung mit MIT App Inventor

Als Programmierumgebung zur visuellen Programmierung habe ich die MIT App Inventor Plattform gewählt, welche die Programmierung von mobilen Anwendungen auch Nutzer*innen ohne vorherige informatische Kenntnisse möglich machen will.

Der MIT App Inventor ist aus einer Kollaboration zwischen dem Massachusetts Institute of Technology und Google entstanden und basiert in weiten Teilen auf vorangegangenen Projekten am MIT wie etwa Logo oder Scratch sowie auf externen

Projekten wie Storytelling Alice. (vgl. Schiller, Turbak und Abelson et al 2014: 6) Bisher wurden laut Angaben der Projektverantwortlichen[3] mehr als 34 Millionen Apps von fast 8,2 Millionen registrierten Nutzer*innen aus 195 Ländern erstellt.

Nachdem die Schüler*innen bereits mit Scratch gearbeitet haben, fanden sie sich relativ schnell auf der Plattform zurecht. Die Apps konnten auf den schulinternen Tablets, welche über USB mit den Schulcomputern verbunden wurden, direkt („live“) getestet werden. Nach einer kurzen Einarbeitungszeit und dem gemeinsamen Programmieren einer „Mini-App“ (Sprachausgabe, über einen Button gesteuert) begannen die Schüler*innen direkt mit der ersten App, welche im PP-Arbeitsmodus programmiert wurde.

[3] http://appinventor.mit.edu/explore/ [Zugriff: 12.3.2020]

a) Pair Programming

Beschreibung der App

Ziel der PP Einheiten mit dem MIT App Inventor war es, eine einfache Maulwurf-App (siehe Abbildung 4) zu programmieren. Der Maulwurf bewegt sich zufällig in einem vordefinierten Zeitintervall auf der Spielfläche und muss vom User berührt werden. Jedes Mal, wenn der Maulwurf berührt wird, erhöht sich der Punktestand. Ab einer bestimmten Anzahl von Punkten taucht eine Bombe auf, die sich genauso zufällig bewegt, aber nicht berührt werden darf. Sobald die Bombe berührt wird, verliert der User und muss das Spiel neu starten.

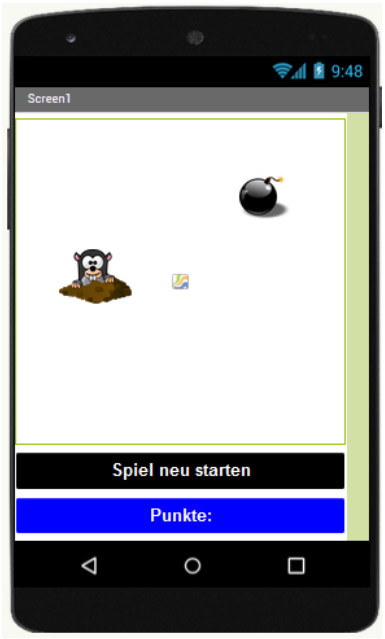
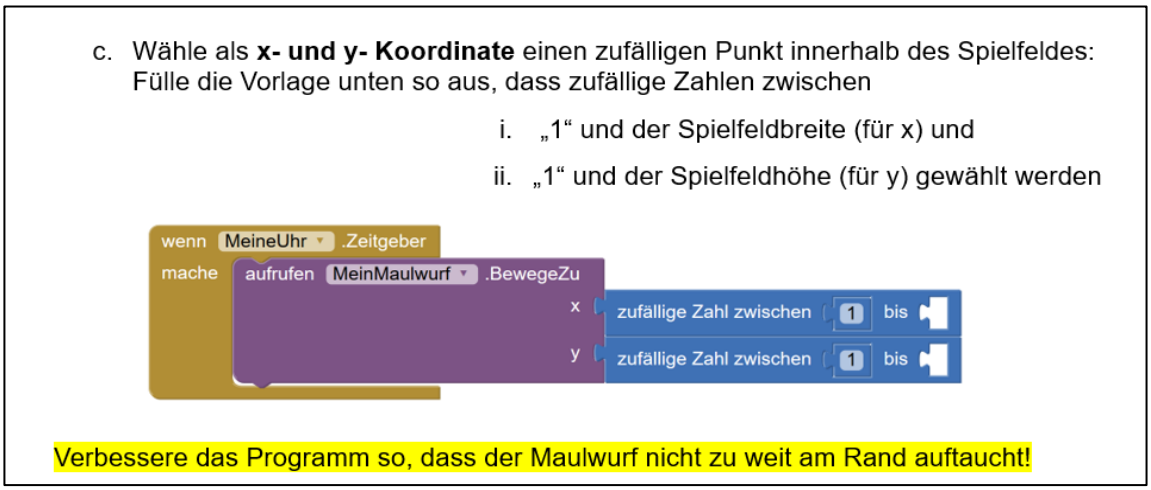


Abbildung 4: Maulwurf-App

Fachdidaktische Überlegungen

Das Tutorial für die Programmiereinheiten zur Maulwurf App ist als Ganzes in Anhang B zu finden. Einerseits wurde darauf geachtet, genügend Hilfsstrukturen im Sinne des scaffolding-Ansatzes bereitzustellen, um eine Überforderung der Schüler*innen zu vermeiden, andererseits wurden möglichst viele Elemente zur Förderung explorativen Lernens eingebaut, wie folgender Ausschnitt aus dem Tutorial illustriert:



Hier wird die zufällige Bewegung des Maulwurfs auf dem Spielfeld mit einer Uhr als Zeitgeber programmiert. Im vordefinierten Intervall wird die Bewegungs-Routine der Spielfigur *MeinMaulwurf* aufgerufen, wobei die Positionen x und y zu setzen sind.

Die Schüler*innen müssen überlegen, welche oberen Grenzen für die Zufallszahlen x und y sinnvoll sind. Die Idee war, sie dabei zum Ausprobieren verschiedener Zahlenwerte zu ermutigen sowie darauf hinzuweisen, dass die App auch auf verschiedenen Geräten (Display-Größen) spielbar sein soll. Eine gute Programmierlösung wäre, die Spielfeld-Breite und -Höhe über die entsprechende Funktion auszulesen und die Breite der Spielfigur von diesem Wert zu subtrahieren.

Die Schüler*innen können dieses Problem aber auch anders lösen. Das Hauptinteresse im Sinne meiner Forschungsfrage liegt dabei weniger auf der konkreten Lösung dieser Herausforderung als auf dem Prozess der Arbeitsform des Pair Programmings.

b) Solo Programming

Ziel der Solo Programming Einheit mit dem MIT App Inventor war es, ein einfaches Pong-Spiel (siehe Abbildung 5) zu implementieren. Das Spiel beginnt, sobald der User den Start-Button betätigt. In der Grundfunktion bewegt sich der Spielball in zufälligem Winkel über das Spielfeld und prallt von drei Spielfeld-Rändern (links, rechts und oben) ab. Sobald der Ball den unteren Spielfeld-Rand erreicht, ist das Spiel verloren. Der User versucht, das zu verhindern, indem ein Balken über die Touch-Funktion des Bildschirms nach links bzw. rechts bewegt wird und der Ball von diesem Balken abprallt.

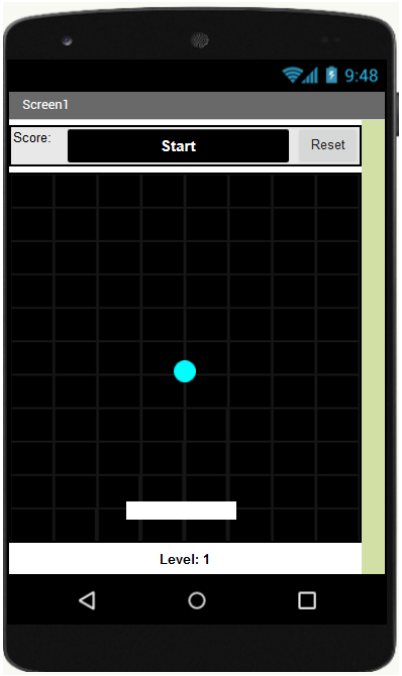
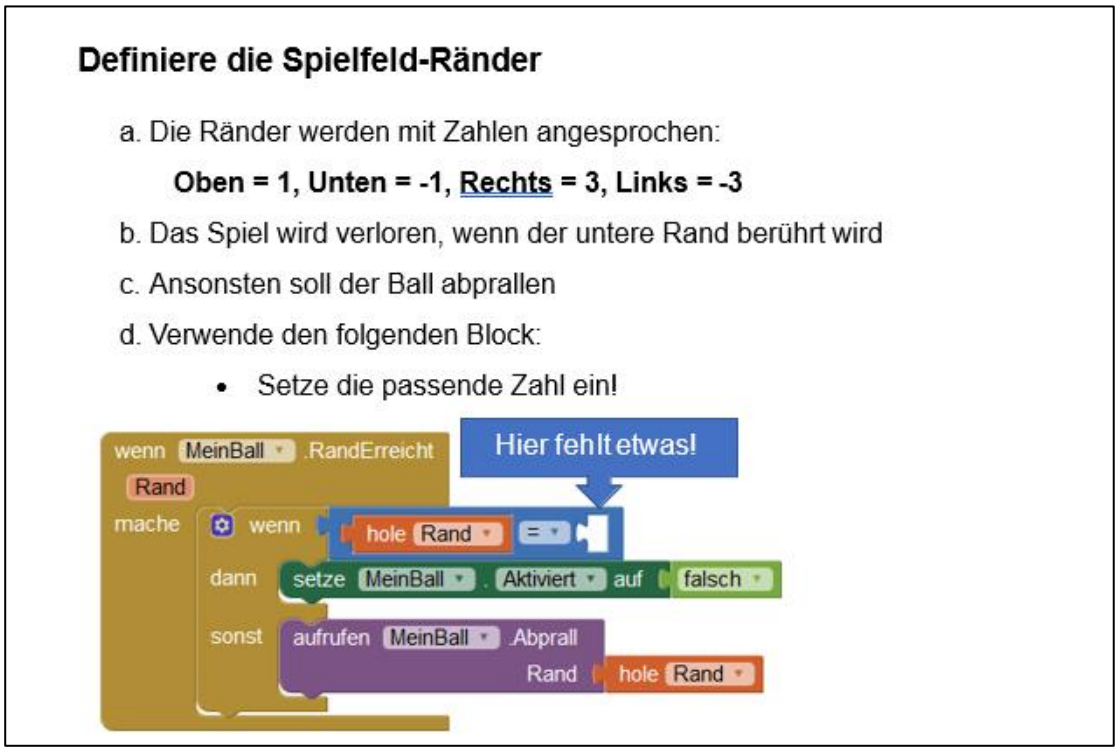


Abbildung 5: Pong-App

Fachdidaktische Überlegungen

Die Pong App eignet sich deshalb besonders gut für den Zweck meiner empirischen Untersuchung, weil im Vergleich mit der Maulwurf App keine neuen Konzepte für die Programmierung notwendig sind. Gleichzeitig unterscheiden sich die Herausforderungen bei der Programmierung aber signifikant, wodurch sichergestellt ist, dass die Schüler*innen ausreichend gefordert sind.

Ein Beispiel für die besonderen Anforderungen kann durch folgenden Ausschnitt aus dem Tutorial (siehe Anhang B, MIT App Inventor: Solo Programming) gezeigt werden:



Im Unterschied zur Maulwurf App bewegt sich die Spielfigur nicht innerhalb definierter Grenzen, sondern kontinuierlich in eine Richtung. Die Herausforderung für die Schüler*innen besteht darin, das Abprallverhalten des Balles abhängig von den verschiedenen Objekten (Ränder bzw. Balken) zu programmieren. Als Hilfestellung ist die relevante Bedingung bereits vorbereitet, die Schüler*innen können durch Ausprobieren das Verhalten des Programms bereits testen und durch Adaption der entsprechenden Bedingung das gewünschte Verhalten erreichen.

Um diese Herausforderung zu lösen, muss zuerst die passende Referenz für den unteren Rand (-1) in das freie Feld gesetzt werden. Danach kann mit unterschiedlichen Lösungswegen erreicht werden, dass die Punkte auf 0 gesetzt werden, die Bewegung des Balles gestoppt und das Spiel insgesamt in den Ursprungszustand zurückgesetzt wird. Ich erwarte mir von dieser Struktur, dass die Schüler*innen sehr schnell durch Ausprobieren die Funktion der Bedingung entdecken und davon ausgehend die weiteren Problemfelder (z.B. Rücksetzen der Variablen, Stoppen der Bewegung etc.) bearbeiten können. Davon erhoffe ich mir, eine gute Balance zwischen motivierenden

Lernfortschritten und notwendigen Herausforderungen im Programmierprozess bei den Lernenden zu erreichen.

3.3.2 Textbasierte Programmierung mit Python

Als Programmiersprache für den Einstieg in die textbasierte Programmierung wurde Python in der Version 3.8 gewählt. Python bietet für den einführenden Programmierunterricht viele Vorteile: Neben der guten Lesbarkeit der Syntax und der klaren Struktur, die etwa bei Chou (2002) hervorgehoben werden, zeichnet sich die Programmiersprache nach Stajano (2002) auch in einer weiteren Betrachtung aus: *„[…] Python, with its high level of abstraction and judicious balance of simplicity, conciseness and versatility, is an excellent choice to introduce the fundamental ideas of the art of programming."* (Stajano 2002)

Nachdem keine*r der Proband*innen Erfahrungen mit textbasierter Programmierung hatte, wurde von annähernd gleichen Vorbedingungen ausgegangen. Bevor die Schüler*innen sich mit den fachdidaktischen Konzepten in den zwei Arbeitsmodi (Pair Programming und Solo Programming) beschäftigten, wurde in einer einführenden Einheit die bisher bekannten Konzepten Bedingung, Schleife und Variable anhand einfacher Beispiele in Python gemeinsam implementiert und durch einige notwendige Konzepte (Listen, Bibliotheken) erweitert. Die Schüler*innen erhielten außerdem in einem „Cheat-Book" eine Sammlung mit Beispielen für diese Konzepte, die sie für die Bearbeitung der fachdidaktischen Konzepte verwenden konnten.

a) Pair Programming

In der Pair Programming Einheit mit Python sollte ein einfaches Kasino-Spiel für die Python-Konsole implementiert werden (siehe Abbildung 6). Der User hat einen bestimmten Geldbetrag zur Verfügung, welcher zu Beginn des Spiels angezeigt wird. Über die Tastatur wird eine Würfelzahl eingegeben, welche mit einer Zufalls-Würfelzahl verglichen wird. Bei Gleichheit der Zahlen erhöht sich der Kontostand, bei Ungleichheit wird ein Betrag abgezogen. Das Spiel endet, sobald der User kein Geld mehr zur Verfügung hat.

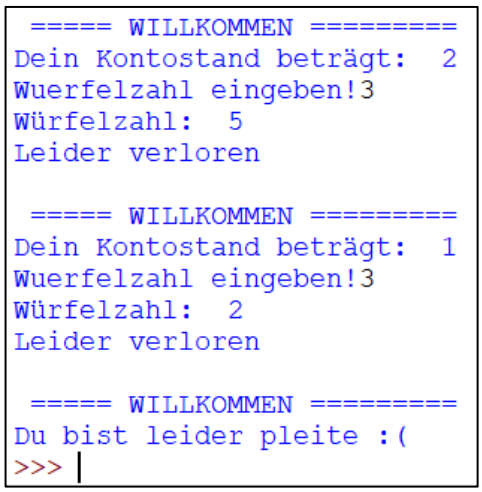


Abbildung 6: Kasino-Programm

Fachdidaktische Überlegungen

Aufgrund der geringen Erfahrung der Lernenden mit textbasierter Programmierung wurde im Tutorial (siehe Anhang B – Python: Pair Programming) der Fokus auf genügend Hilfsstrukturen gelegt, um Erfolgserlebnisse für alle Proband*innen zu ermöglichen. Gleichzeitig wurde darauf geachtet, dass die Lernenden im Sinne von explorativen Lernerlebnissen selbstständig Lösungswege erarbeiten können.

Ein Beispiel dafür ist der folgende Ausschnitt aus dem Kasino-Spiel, in dem es um die Implementierung der Zufallszahlen geht:

2. Erzeuge die Zufallszahl!

a. Importiere die *random*-Bibliothek am Anfang des Programmes: import random

b. Speichere eine **Zufalls-Würfelzahl** in einer Variable

c. Gib die Zufalls-Würfelzahl nach der User-Zahl aus

d. Das Ergebnis sollte so ausschauen:

```
Wuerfelzahl eingeben!1
Die User-Zahl ist 1
Die Zufalls-Würfelzahl ist 5
Wuerfelzahl eingeben!2
Die User-Zahl ist 2
Die Zufalls-Würfelzahl ist 5
Wuerfelzahl eingeben!3
Die User-Zahl ist 3
Die Zufalls-Würfelzahl ist 6
```

Hier wenden die Schüler*innen bekannte Konzepte (Import einer Bibliothek, Erstellung von Zufallszahlen) an, müssen diese aber in neuer, bisher unbekannter Weise (Speicherung in einer Variable und Ausgabe) verknüpfen. Die Ausgabe der vom User eingegebene Zahl einerseits und der zufälligen Würfelzahl andererseits ist außerdem bereits eine Vorbedingung für den nächsten Schritt, in dem diese beiden Zahlen verglichen und dementsprechend unterschiedliche Ausgaben gemacht werden:

3. Überprüfe, ob der User gewinnt!

a. Lösche die Ausgabe der User-Zahl

b. Erzeuge eine **If-Schleife**, in der überprüft wird, ob User-Zahl und Zufalls-Würfelzahl gleich sind

   i. Wenn ja: Gib „**GEWONNEN**“ aus

   ii. Wenn nein: Gib „**VERLOREN**“ aus

c. Das Ergebnis sollte so ausschauen:

```
Wuerfelzahl eingeben!4
Die Zufalls-Würfelzahl ist 5
VERLOREN
Wuerfelzahl eingeben!4
Die Zufalls-Würfelzahl ist 4
GEWONNEN
Wuerfelzahl eingeben!4
Die Zufalls-Würfelzahl ist 3
VERLOREN
```

Von dieser Aufgabenstellung erhoffe ich mir Rückschlüsse auf die Team-Prozesse innerhalb der Paare, die sich durch die genannten Herausforderungen ergeben sowie Daten zu den Vorgangsweisen und Lösungsstrategien, die angewandt wurden. Im Rahmen der qualitativen Interviews mit ausgewählten Proband*innen sollen diese Prozesse dann reflektiert werden.

b) Solo Programming

Ziel der Solo Programming Einheiten war es, ein Programm zur Verwaltung einer Bibliothek zu implementieren. Dabei wurde das Framework – das heißt der Programmcode, der über einen User-Input verschiedene Funktionen bereitstellt – vorgegeben. Die Schüler*innen hatten die Aufgabe, diese Funktionen dann programmieren.

Abbildung 7 zeigt die Struktur des Frameworks: in einer Liste *books* wird als erstes Listenelement eine weitere Liste mit den Daten zum ersten Buch gespeichert. In einer while-Schleife kann der User über die Eingabe einer Ziffer zwischen den Funktionen [1] Buch hinzufügen, [2] ISBN Suche, [3] Anzeige aller Bücher sowie [x] für Abbruch wählen, worauf die entsprechende Bedingung ausgeführt wird. [3] ist bereits implementiert, die Tutorials behandelten die Funktionen [1], [2] und [x] sowie Zusatzfunktionen.

```
book_nummer = 0

books = [[book_nummer,'Herr d. Ringe', 123456, 33.23, 'entliehen']]

start = True

while start == True:
    print("\n")
    userinput = input("[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! ")

    if userinput == 'x':
        print("x")
        ## Hier fehlt Code

    elif userinput == '1':
        print("1")
        ## Hier fehlt Code

    elif userinput == '2':
        print("2")
        ## Hier fehlt Code

    elif userinput == '3':
        for item in books:
            print(item)

    elif userinput == '4':
        print("4")
        ## Hier fehlt Code
    else:
        print("ERROR")


print("Programm beendet")
```

Abbildung 7: Framework Bibliothek

Abbildung 8 zeigt den erwarteten Output für Funktionen [3] und [2], nachdem über Funktion [1] zwei Bücher (Test und Superbuch) hinzugefügt wurden. Über die ISBN-Suche [2] wird eine Zahl eingegeben und das entsprechende Buch ausgegeben.

```
[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 3
[0, 'Herr d. Ringe', 123456, 33.23, 'entliehen']
[1, 'Test', '123', '12', 'verfügbar']
[2, 'Superbuch', '12345', '32', 'verfügbar']


[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 2
ISBN eingeben12345
[2, 'Superbuch', '12345', '32', 'verfügbar']
```

Abbildung 8: Bibliothek-Programm

Fachdidaktische Überlegungen

In der Planung dieser Solo Programming Einheiten wurde darauf geachtet, nicht zu viele Überschneidungen mit bereits aus der letzten Einheit (Kasino-Spiel) bekannten Konzepten zuzulassen, um möglichst viele „neue" Herausforderungen zu bieten. Aus dieser Überlegung wurde der Fokus auf den Umgang mit einer verschachtelten Liste sowie auf bisher unbekannten algorithmischen Anforderungen gelegt. Ein Beispiel ist im Ausschnitt aus dem fachdidaktischen Konzept in Abbildung 9 zu sehen: Die Schüler*innen sollen den Zugang zum Hauptprogramm durch ein Passwort schützen.

4. Implementiere einen Passwort-Schutz!

a. Erzeuge eine Variable **passwort** VOR der while-Schleife

b. Verwende **input()**, damit der User das Passwort eingeben kann

c. Solange das Passwort nicht gleich „**mama**" ist, kommt der User nicht zum „Hauptprogramm"

d. Das Ergebnis sollte so ausschauen:

```
Willkommen zur Bibliothek! Bitte Passwort hinzufügen!papa
Falsches Passwort - neuer Versuch!oma
Falsches Passwort - neuer Versuch!mama


[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch!
```

Abbildung 9: Passwort-Schutz in Python (1)

Dabei gilt es einerseits zu erkennen, dass hierfür eine while-Schleife mit dem passenden Operator (passwort != „mama“) die passende Struktur liefert. Andererseits muss überlegt werden, was innerhalb dieser Schleife passieren muss, um das geforderte Verhalten zu ermöglichen. Eine einfache Variante ist in Abbildung 10 gezeigt.

```
passwort = input("Passwort eingeben!")
while passwort != 'mama':
    passwort = input ("Passwort falsch, bitte noch einmal!")
```

Abbildung 10: Passwort-Schutz in Python (2)

Dieser Ausschnitt zeigt exemplarisch, wie im fachdidaktischen Konzept versucht wurde, einerseits eine klare „Hilfsstruktur“ zu bieten und andererseits genügend Raum für exploratives Lernen zu lassen. Ich erwarte mir von diesem Konzept, dass die Schüler*innen in den Aufgaben eine echte Herausforderung sehen und unterschiedliche Lösungswege finden, die Rückschlüsse zu den Lernprozessen im Solo Programming zulassen.

Für die systematische Auswertung der Daten, die anhand der empirischen Untersuchung zu den hier vorgestellten Konzepten gesammelt werden, sind die qualitativen Interviews mit ausgewählten Proband*innen zentral. Deshalb werde ich im Folgenden das Design für diese Interviews vorstellen.

3.4. Design der Interviews

Für die Beantwortung der Forschungsfrage ist es zentral, nach der Bearbeitung der fachdidaktischen Konzepte die Schüler*innen hinsichtlich ihrer Erfahrungen mit Pair Programming und Solo Programming als Arbeitsform zu interviewen. Ziel dieser Interviews ist es, Hinweise auf Unterschiede und Gemeinsamkeiten von PP beim einführenden Programmierunterricht in textbasierten bzw. visuellen Programmiersprachen zu finden.

Als Vorgehensweise habe ich mich für einen zweistufigen Prozess entschieden: Im ersten Schritt unmittelbar nach jedem Unterrichtsblock füllen die Schüler*innen einen Fragebogen mit geschlossenen und offenen Fragen zu ihren Erfahrungen mit PP und SP für diesen Unterrichtsblock (also einmal für textbasierte und einmal für visuelle Programmierung) aus. Dieser Fragebogen liefert qualitative und quantitative Daten, wobei durch die geringe Fallzahl die Aussagekraft der quantitativen Daten natürlich sehr gering ist. Der erstelle Fragebogen ist im Anhang C dieser Arbeit angefügt.

Im zweiten Schritt wähle ich drei Schüler*innenpaare aus, mit denen ich nach der letzten Einheit Fokusinterviews zu ihren Erfahrungen mit PP und SP führe. Die Auswahl der Interviewpartner*innen soll so erfolgen, dass möglichst diverse Ergebnisse und Erfahrungen erwartet werden können. Deshalb werden sowohl die Programmier-Vorerfahrungen als auch die Leistungen im SP und PP als Basis für die Auswahl herangezogen.

Für die Fokusinterviews wurden je ein Programm mit dem MIT App Inventor sowie mit Python erstellt. Bei beiden Programmen wurde darauf geachtet, dass die Implementierung mit dem Kenntnisstand der Schüler*innen nach den durchgeführten Einheiten grundsätzlich möglich sein sollte.

Mit dem MIT App Inventor wurde ein Minigolf-Spiel programmiert: Ziel ist es, mit einer Wisch-Bewegung am Smartphone/Tablet einen weißen Ball in ein schwarzes Loch zu schießen, wobei der Ball sich gemäß der natürlichen physikalischen Gesetze verhält

(also mit der Zeit langsamer wird) und an den Rändern sowie am blauen Hindernis abprallt (siehe Abbildung 11).

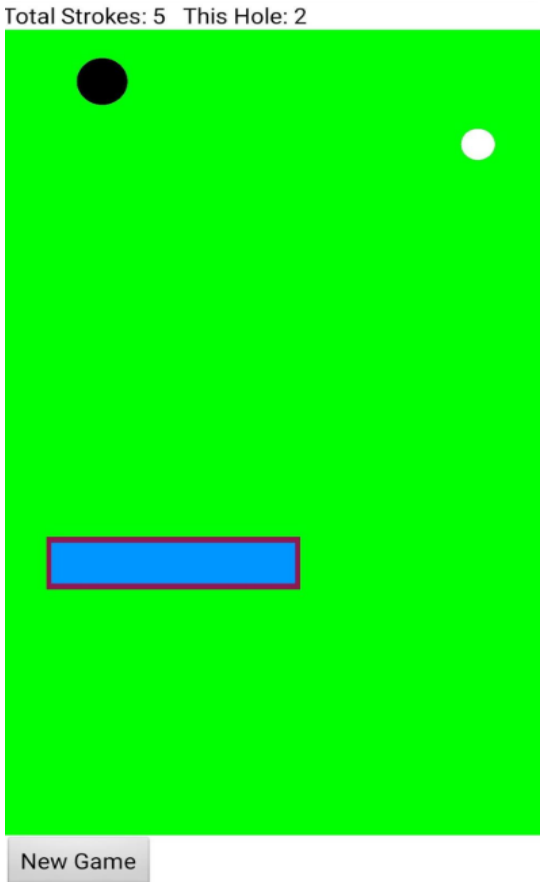


Abbildung 11: Minigolf-Spiel

Das Python-Programm bietet die Möglichkeit, „unfaire“ Noten für Schüler*innen zu berechnen. Als Basis für die Note wird eine Zufallszahl zwischen 1 und 5 erzeugt, je nach Augenfarbe und Geschlecht verbessert oder verschlechtert sich diese Note (siehe Abbildung 2). Das Programm soll durch den ironischen Zugang Spaß machen und beinhaltet mehrere Grundprinzipien der Programmierung, die auch in den fachdidaktischen Konzepten vermittelt wurden.

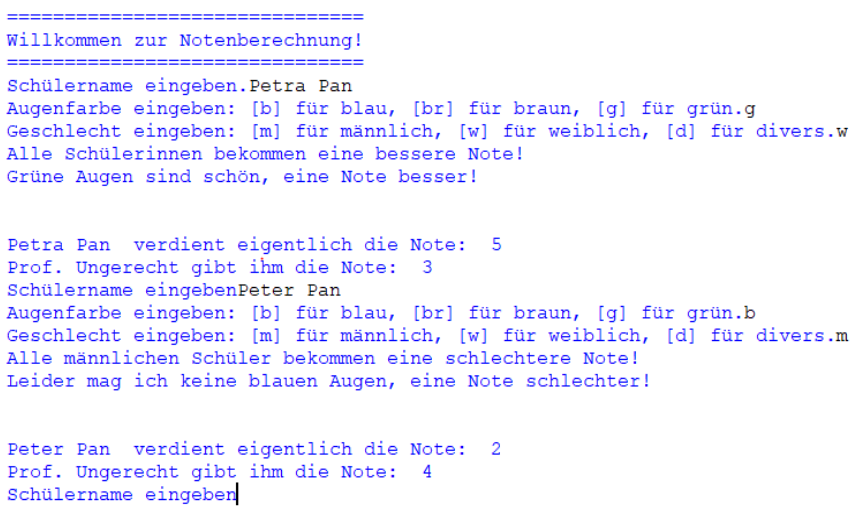


Abbildung 12: Notenberechnungs-Programm

Die Fokusinterviews begannen jeweils damit, dass der/die Interviewte die Anwendungen praktisch ausprobieren konnten. In den Erzählaufforderungen (siehe Anhang C) wurde der Fokus dann einerseits auf mögliche Herausforderungen bei der Programmierung dieser Anwendungen, andererseits natürlich auf die Perspektiven zur Arbeitsform (PP versus SP) im Hinblick auf diese Herausforderungen gelegt.

Um Daten für die zentralen Forschungsinteressen dieser Arbeit zu bekommen, werden im Fokusinterview – falls ohne explizite Erzählaufforderung keine Information dazu kommt – mehrere Nachfragen gestellt:

a. Frage zu möglichen Vorteilen von PP (Frage 3)
b. Frage zu möglichen Effekten auf die Motivation (Frage 4)
c. Frage zur Einschätzung der Qualität je nach Arbeitsform (Frage 5)

Die Kategorienbildung zur Auswertung des Fokusinterviews kann nach Mayring (2010) entweder deduktiv oder induktiv oder aber in einer Kombination aus beiden Ansätzen erfolgen. Aufgrund der zahlreichen vorhandenen Studien zum Thema PP habe ich mir für eine deduktive Kategorienbildung entschieden, wobei die Kategorien aus den Erkenntnissen der SLR abgeleitet und auf das Fokusinterview angepasst wurden.

Von dieser Vorgangsweise erhoffe ich mir einerseits ein sehr offenes Interview, in dem genug Raum für unerwartete Sichtweisen bleibt und andererseits genügend Fokus auf das eigentliche Erkenntnisinteresse dieser Arbeit.

4. DATENERHEBUNG UND -AUSWERTUNG

Die Datenerhebung erfolgte über mehrere Wochen hinweg jeweils in den Einheiten des Wahlpflichtfaches „Coding and Technology 6. Klasse“ am BG/BRG Biondekgasse Baden. Die Interviews wurden je nach Verfügbarkeit der betreffenden Schüler*innen nach der Durchführung des zweiten fachdidaktischen Konzeptes geführt. Im Folgenden werde ich die einzelnen Schritte bei der Durchführung der empirischen Untersuchung beschreiben, um anschließend eine Auswertung der gesammelten Daten vorzunehmen.

4.1. Durchführung der empirischen Untersuchung

Vorerhebung und Einteilung der Paare

Die Bearbeitung der Aufgaben aus Anhang A war der erste Teil der empirischen Untersuchung. Ziel war es einerseits, Daten zu den Programmierkenntnissen der Schüler*innen zu erhalten und andererseits, eine Grundlage für die Einteilung in Gruppen (Pair Programming) zur Verfügung zu haben. Aufgrund von Krankmeldungen haben nur 12 der 14 Schüler*innen an der Vorerhebung teilgenommen – die zwei verbleibenden Schüler*innen bildeten somit in den PP-Einheiten das siebte Paar.

Die Schüler*innen hatten 15 Minuten Zeit, die Aufgaben zu bearbeiten, wobei nach 12 Minuten alle das Quiz beendet hatten. Das verbale Feedback der Schüler*innen zum Quiz war durchwegs positiv, es gab keine Verständnisprobleme. In der allgemeinen Auswertung (Abbildung 13) zeigt sich, dass es sich um eine relativ homogene Gruppe handelt: Die meisten Schüler*innen erreichten 10 Punkte, 3 Schüler*innen 8 und nur zwei Schüler*innen erreichten 6 Punkte. Eine einzige Schülerin hat alle Fragen richtig beantwortet, das durchschnittliche Ergebnis lag bei 9 Punkten.

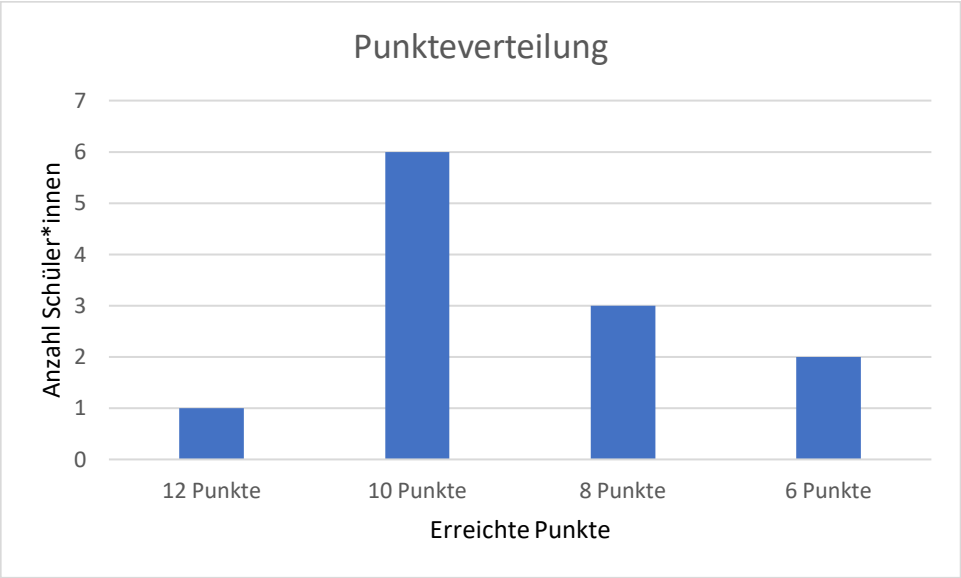


Abbildung 13: Ergebnisse Programmier-Vorerhebung

In der Detailansicht (Abbildung 14) zeigt sich, dass die Schüler*innen mit den Scratch-Aufgaben (Aufgaben 4-6) gar keine bzw. kaum Probleme hatten: Aufgabe 4 und 5 haben alle Schüler*innen richtig beantwortet, Aufgabe 6 90% der Schüler*innen. Bei den Aufgaben zum informatischen Denken (Aufgabe 1-3) hatten die Schüler*innen eine viel höhere Fehlerquote: Nur knapp über 40% der Schüler*innen haben Aufgabe 1 und 2 richtig beantwortet, etwas weniger als 80% Aufgabe 3.

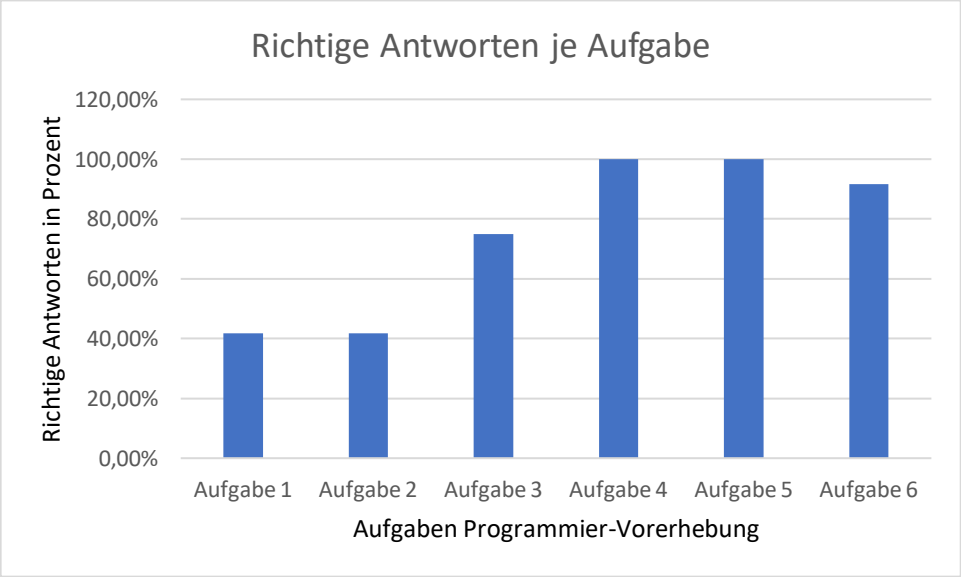


Abbildung 14: Richtige Antworten Programmier-Vorerhebung

An dieser Stelle muss festgehalten werden, dass diese Erhebung an sich keine wissenschaftliche Relevanz hat, weil zahlreiche unbekannte Faktoren das Ergebnis beeinflussen können. Aus zwei Gründen ist diese Vorerhebung trotzdem hilfreich für mein Forschungsvorhaben: Erstens bietet sie die Basis für die Einteilung der Paare beim Pair Programming – hier sollte darauf geachtet werden, Schüler*innen mit ähnlichen Vorerfahrungen zusammen zu bringen. (vgl. Man Lui und Chan 2006) Zweitens können in weiterer Folge auch die Leistungen der Schüler*innen bei den vier Programmier-Projekten (zwei in Python, zwei mit dem MIT App Inventor) in einen Kontext mit den Leistungen bei der Programmier-Vorerhebung gebracht werden.

Aufgrund dieser Ergebnisse sowie persönlicher Präferenzen wurde folgende Einteilung der Schüler*innen für das PP vorgenommen:

	Schüler*in	**Punkte**
Team 1	S(w)1 und S(w)2	12 / 10
Team 2	S(m)1 und S(m)2	10 / 10
Team 3	S(m)3 und S(m4)	10 / 10
Team 4	S(w)3 und S(m)5	10 / 8
Team 5	S(m)6 und S(w)4	8 / 8
Team 6	S(m)7 und S(m)8	6 / 6
Team 7	S(m)9 und S(m)10	n. / n.

In Klammer wird das Geschlecht angegeben, S(m)9 und S(m)10 haben nicht an der Vorerhebung teilgenommen, weshalb keine Punkte angegeben werden können.

Umsetzung der fachdidaktischen Konzepte

Im nächsten Schritt der empirischen Untersuchung wurde das fachdidaktische Konzept für den MIT App Inventor (Maulwurf-App) in der Arbeitsform Pair Programming von den Schüler*innen bearbeitet. Alle Schüler*innen waren anwesend und konnten innerhalb der vorgegebenen Zeit (2*50 Minuten) eine funktionierende App programmieren. Abgesehen von Rückfragen, die auf eine Konkretisierung einer Aufgabe (etwa der geforderten Berechnung der Position des Maulwurfs) bezogen, haben alle Paare selbstständig an den Projekten gearbeitet, auch der Wechsel der Rollen (*driver* und *navigator*) nach ca. 20 Minuten hat gut funktioniert. In den nächsten zwei Einheit arbeiteten die Schüler*innen in der Arbeitsform Solo Programming am fachdidaktischen Konzept des Pong-Spiels. Auch hier haben es alle Schüler*innen geschafft, zumindest die Grundfunktionen der App zu implementieren. Auffallend war, dass die Schüler*innen deutlich ruhiger waren als im PP, auch Rückfragen gab es kaum.

Die fachdidaktischen Konzepte zur textbasierten Programmierung mit Python wurden umgesetzt, nachdem die Schüler*innen in einer Doppel-Einheit einige erste Erfahrungen mit Syntax und Semantik in Python sammeln konnten. Auch hier wurde zuerst im PP das Kasino-Programm erstellt. Es war auffallend, dass die Schüler*innen deutlich schneller als bei den ersten PP-Einheiten in einen produktiven Arbeitsmodus kamen und mit der Programmierung beginnen konnten. Gleichzeitig war aber auch spürbar, dass es mehr Diskussionsbedarf gibt: In allen Paaren wurde die richtige Umsetzung des Konzeptes mehr oder weniger hitzig debattiert. Am Ende der Einheiten konnten alle Paare ein Programm abgeben, welches die Anforderungen zumindest in der Grundfunktion erfüllte. In den letzten zwei Einheiten arbeiteten die Schüler*innen in der Arbeitsform Solo Programming an der Implementierung der Bibliothek. Analog zu den Erfahrungen mit dem MIT App Inventor war es auch hier spürbar ruhiger, gleichzeitig tauchten viele Verständnisfragen auf. Am Ende der letzten Einheit konnten trotzdem wieder alle Schüler*innen funktionierende Programme abgeben.

Erhebungen mit Fragebogen und Fokusinterviews

Alle teilnehmenden Schüler*innen füllten direkt nach Abschluss des jeweiligen Blocks (visuelle Programmierung bzw. textbasierte Programmierung) den jeweiligen Fragebogen aus Anhang C aus. Um eine Zuordnung der Schüler*innen zu den von ihnen ausgefüllten Fragebögen zu ermöglichen, wurden in den Fragebögen jeweils Vorname und das Alter abgefragt. Diese Zuordnung war notwendig, um die Auswahl der Interviewpartner*innen für die Fokusinterviews möglichst treffsicher zu gestalten.

Die Schüler*innen hatten zehn Minuten Zeit, um den Fragebogen auszufüllen, wobei die meisten Schüler*innen schon nach fünf Minuten abgegeben haben. Basierend auf der Auswertung (siehe nächstes Kapitel) der Fragebögen wurden im letzten Schritt der empirischen Untersuchung Fokusinterviews mit drei Paare aus der PP-Phase geführt. Hierzu wurden die Schüler*innen einzeln gefragt, ob sie einem (anonymen) Fokusinterview mit der Dauer von ca. 10 – 15 Minuten zustimmen. Alle angefragten Schüler*innen haben dem Interview zugestimmt, wodurch die gewünschte Bandbreite an Interviewpartner*innen erreicht werden konnten. Die Interviews wurden außerhalb der Unterrichtszeit jeweils einzeln geführt und mit einem Audio-Aufnahmegerät aufgezeichnet. In den ersten Minuten beschäftigten sich die Schüler*innen praktisch mit den Programmen des MIT App Inventors (Minigolf-Spiel) und dem Python-Programm (Unfaire Notenberechnung). Im nächsten Schritt wurde der Fokus anhand der Fragen aus Anhang C auf die Erfahrungen mit PP und SP gelenkt, wobei die Gesprächsaufforderung dazu dienen sollte, möglichst freie Assoziationen der Interviewten zu ermöglichen. Alle sechs Schüler*innen (2 Schülerinnen, 4 Schüler) sind der Gesprächsaufforderung nachgekommen und haben ausführlich von ihren Erfahrungen und Gedanken zu PP und SP erzählt. Die Interviews bildeten den Abschluss der empirischen Untersuchung.

4.2. Auswertung der Daten

Auswertung der Programme

Die Auswertung der von den Schüler*innen erstellen Programme erfolgte nach einem klar nachvollziehbaren Punkteschema: Für jedes Programm waren maximal 10 Punkte zu erreichen. Punkteabzüge gab es, wenn bestimmte Anforderungen nicht erfüllt waren (z.B. Challenge wurde nicht implementiert) oder wenn eine Funktion fehlerhaft war. Zu jedem abgegebenen Programm bekamen die Schüler*innen neben der erreichten Punkteanzahl auch ein mündliches Feedback mit Erklärungen und Verbesserungsvorschlägen. In der folgenden Tabelle sind die Ergebnisse für alle Schüler*innen, gruppiert in die jeweiligen PP-Teams, zusammengefasst.

		Quiz	**Visuell PP**	**Visuell SP**	**Textbasiert PP**	**Textbasiert SP**
Team 1	S(w)1	100%	100%	100%	100%	70%
	S(w)2	83%	100%	80%	100%	60%
Team 2	S(m)1	83%	90%	80%	90%	70%
	S(m)2	83%	90%	90%	90%	60%
Team 3	S(m)3	83%	90%	90%	100%	60%
	S(m)4	83%	90%	90%	100%	50%
Team 4	S(w)3	83%	90%	60%	100%	60%
	S(w)4	67%	90%	70%	100%	60%
Team 5	S(m)6	67%	100%	70%	90%	50%
	S(m)5	67%	100%	60%	90%	40%
Team 6	S(m)7	67%	60%	40%	70%	50%
	S(m)8	50%	60%	50%	70%	30%
Team 7	S(m)9	n.	70%	50%	60%	50%
	S(m)10	n.	70%	50%	60%	50%
	Durchschnitt		**86%**	**70%**	**87%**	**54%**
	Minimum		**60%**	**40%**	**60%**	**30%**

Auch bei diesen Ergebnissen muss festgehalten werden, dass aufgrund der geringen Stichprobe keine quantitative Analyse der Leistungen möglich ist. Primär dient diese Zusammenfassung als ergänzendes Material zu den Fragebögen, um die Auswahl der Interviewpartner*innen für die Fokusinterviews zu erleichtern.

Gleichzeitig lohnt es sich aber auch, die Ergebnisse für die teilnehmenden Schüler*innen im Detail zu betrachten. Der Durchschnittswert der erreichten Punkte lag bei den PP-Projekten bei 86% (visuell) und 87% (textbasiert), bei den SP-Projekten aber nur bei 70% (visuell) und 54% (textbasiert). Im Durchschnitt erreichten die Schüler*innen im PP also deutlich mehr Punkte, wobei der Unterschied bei der textbasierten Programmierung (mit 33% Unterschied zwischen PP und SP) nochmals viel deutlicher ausfällt als bei der visuellen Programmierung (mit 16% Unterschied zwischen PP und SP). Beim Minimum der erreichten Punkte zeigt sich ein ähnliches Bild: Sowohl bei der textbasierten als auch bei der visuellen Programmierung war das schlechteste Ergebnis im PP 60%, im SP aber deutlich weniger (40% für die visuelle Programmierung und 30% für die textbasierte Programmierung).

Auch wenn aus diesen Ergebnissen noch keine klaren Schlüsse bezüglich der unterschiedlichen Effekte von PP auf den einführenden Programmierunterricht für textbasierte und visuelle Programmiersprachen getroffen werden können, ist bei den teilnehmenden Schüler*innen ein Trend erkennbar: Die Schüler*innen haben im PP deutlich mehr Punkte erreicht als im SP, wobei dieser Effekt bei der textbasierten Programmierung viel deutlicher ausgeprägt war.

Auswertung der Fragebögen

Die ersten zwei Fragen zielen auf die Selbsteinschätzung der Schüler*innen in Bezug auf ihr informatisches Interesse und Talent ab, insofern sind diese beiden Fragen sowohl für die textbasierte als auch für die visuelle Programmierung relevant.

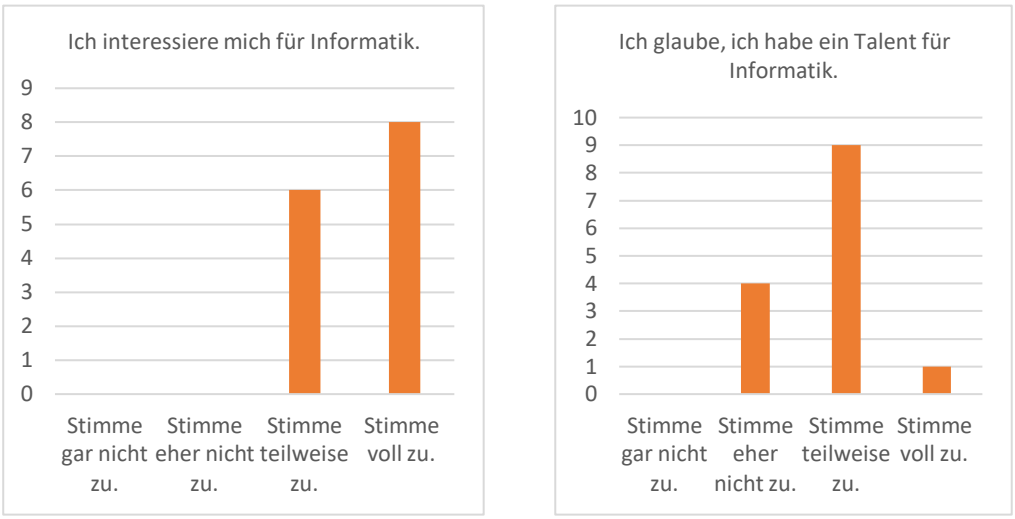


Abbildung 15: Interesse und Talent (allgemein)

Wie erwartet – schließlich handelt es sich bei den Proband*innen um Teilnehmer*innen des Wahlpflichtfaches – ist das Interesse für Informatik sehr hoch. Interessant ist, dass immerhin vier Schüler*innen der Aussage *„Ich glaube, ich habe ein Talent für Informatik“* eher nicht zustimmen. Diese Angaben decken sich mit zahlreichen Studien (siehe z.B. Gomes und Mendes 2007), wonach Informatik und Programmieren von Schüler*innen als kompliziert und schwierig wahrgenommen wird.

Die nächste Frage bezog sich darauf, ob das Programmieren mit Python bzw. mit dem MIT App Inventor aus Sicht der Schüler*innen schwierig war.

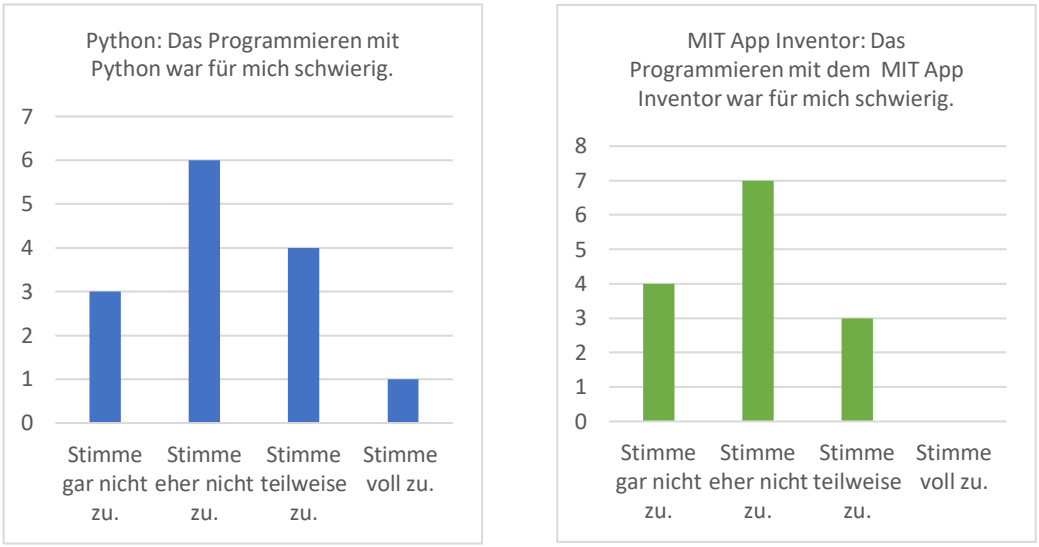


Abbildung 16: Schwierigkeit (Python vs. MIT App Inventor)

Hier zeigt sich, dass sowohl bei der textbasierten als auch bei der visuellen Programmierung die Mehrzahl der Schüler*innen (9 von 14 für Python bzw. 11 von 14 für den MIT App Inventor) gar nicht oder eher nicht zustimmten. Insgesamt zeigt sich aber auch, dass die Schüler*innen die textbasierte Programmierung mit Python schwieriger empfunden haben als die visuelle Programmierung mit dem MIT App Inventor: Bei Python stimmten 5 Schüler*innen teilweise oder voll zu, dass das Programmieren schwierig war, beim MIT App Inventor nur 3 Schüler*innen (davon stimmte niemand voll zu). Dieses Ergebnis entspricht der Erwartung, dass textbasierte Programmierung als schwieriger empfunden wird.

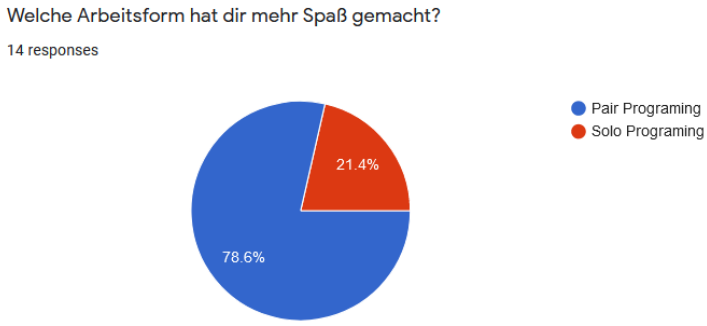
Die nächsten beiden Fragen beziehen sich darauf, ob es den Schüler*innen im PP Arbeitsmodus leichter fiel, den Tutorials bzw. den Challenges zu folgen.

Abbildung 17: Tutorials und Challenges (Python vs. MIT App Inventor)

Sowohl bei den Tutorials als auch bei den Challenges zeigt sich ein deutlicher Unterschied zwischen der textbasierten und der visuellen Programmierung: Während beim MIT App Inventor nur 6 von 14 Schüler*innen teilweise oder voll zustimmten, im PP den Tutorials besser folgen zu können, waren es bei Python 11 von 14 Schüler*innen, also fast doppelt so viele. Bei der Frage zu den Challenges ist dieser Unterschied noch deutlicher: 12 von 14 Schüler*innen stimmen teilweise oder voll zu, dass diese im PP leichter lösbar waren – beim MIT App Inventor machten nur 7 von 14 Schüler*innen diese Angabe.

Bei der Frage, welche Arbeitsform mehr Spaß macht, zeigt sich sowohl bei der textbasierten als auch bei der visuellen Programmierung ein eindeutiges Bild: Die deutliche Mehrheit der Schüler*innen gibt an, dass PP mehr Spaß macht. Die Unterschiede zwischen Python und dem MIT App Inventor sind dabei sehr gering.

Python



MIT App Inventor

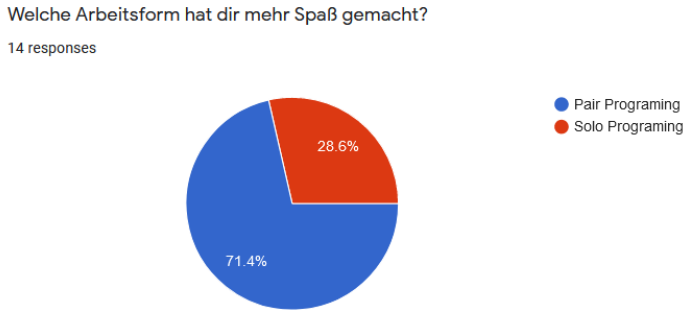
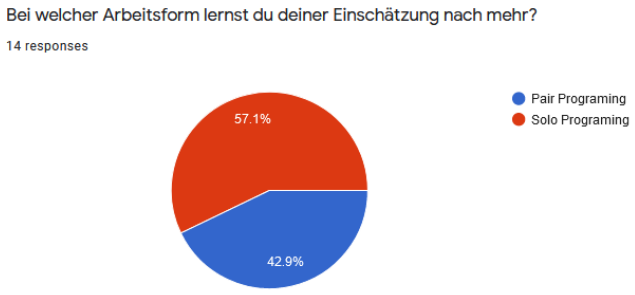


Abbildung 18: Spaß (Python vs. MIT App Inventor)

Deutlich überraschender ist die Einschätzung der Schüler*innen, bei welcher Arbeitsform der Lerneffekt größer ist. Bei Python geben 57% der Schüler*innen (also 8 von 14) an, im SP mehr zu lernen – beim MIT App Inventor erhöht sich der Prozentsatz sogar auf 71% oder 10 von 14 Schüler*innen.

Python



MIT App Inventor

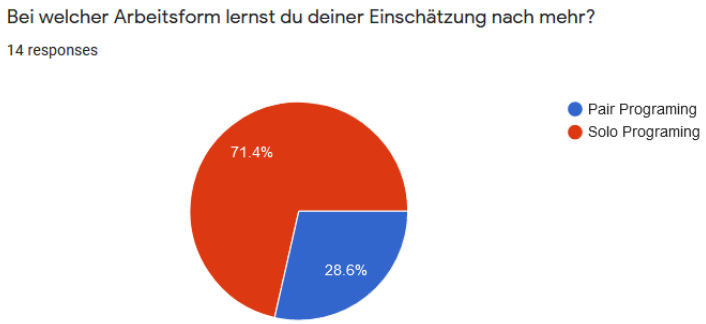


Abbildung 19: Lerneffekt (Python vs. MIT App Inventor)

Diese Selbsteinschätzung vieler Schüler*innen ist deshalb relevant, weil die Effekte von PP auf den Lernerfolg (bei visuellen und textbasierten Programmiersprachen) untersucht werden sollen. Aus diesem Grund werde ich diesen Aspekt in den qualitativen Interviews näher beleuchten.

Die nächste Frage bezog sich auf die Problemlösung je nach Arbeitsform. Hier ist das Ergebnis wieder eindeutig: Sowohl bei Python als auch beim MIT App Inventor gibt ein Großteil der Schüler*innen (9 von 14 bei Python sowie 8 von 14 beim MIT App Inventor) an, Probleme im PP schneller lösen zu können.

Python

MIT App Inventor

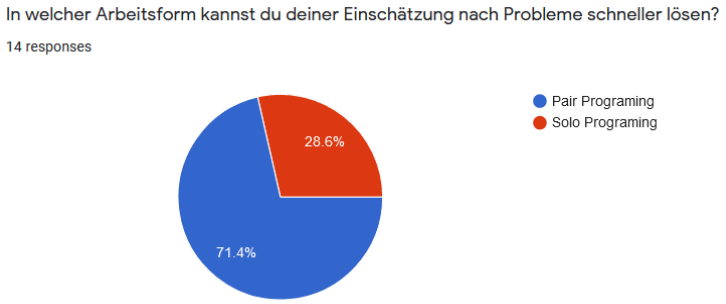
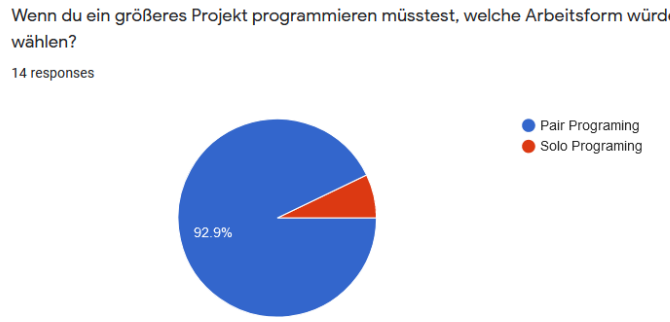


Abbildung 20: Lernerfolg (Python vs. MIT App Inventor)

Die letzte Frage bezog sich auf das Szenario eines größeren Projektes. Hier zeigt sich ein deutlicher Unterschied zwischen Python und dem MIT App Inventor: Während fast alle Schüler*innen bei größeren Python-Projekten PP als Arbeitsform bevorzugen würden (13 von 14), ist der Prozentsatz beim MIT App Inventor deutlich geringer (9 von 14), wenn auch immer noch sehr hoch.

Python



MIT App Inventor

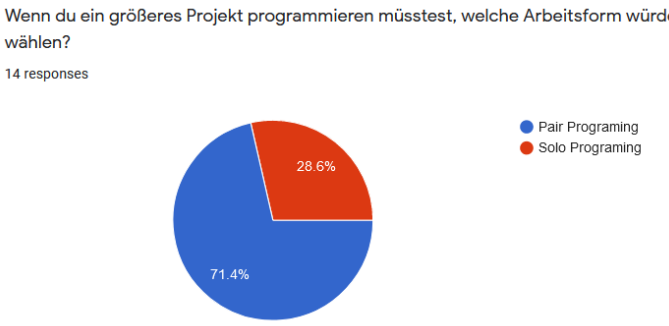


Abbildung 21: Projektprogrammierung (Python vs. MIT App Inventor)

In den offenen Fragen konnten die Schüler*innen ihre Wahrnehmung von den PP-Effekten für die visuelle und textbasierte Programmierung in Textform bringen. Die Antworten auf diese offenen Fragen liefern einerseits erste qualitative Anhaltspunkte für die Beantwortung meiner Forschungsfrage, andererseits sind sie eine weitere Basis für die Auswahl der Interviewpartner*innen für die Fokusinterviews.

Nachdem sich viele Antworten ähnlich sind, werden hier nur einige exemplarische Antworten für unterschiedliche genannte Argumente vorgestellt.

Vorteile von PP (visuelle Programmierung)

Bei der visuellen Programmierung wurde ein wesentlicher Vorteil beim PP darin gesehen, die wahrgenommene Komplexität des MIT App Inventor User-Interfaces gemeinsam besser bewältigen zu können:

> *Fürs erste Mal programmieren fand ich es sehr gut zusammenzuarbeiten, weil man zu zweit weniger überfordert ist, mit einem neuen Programm. Ich und meine Freundin denken sehr unterschiedlich und deshalb hatte ich manche Sachen wie zum Beispiel wie man sich Bilder einfügt schneller verstanden als sie, während sie dafür bei wieder bei anderen Konzepten schneller den Dreh heraus hatte. (Team 1: S(w)2)*

Die Antwort von S(w)2 steht exemplarisch für mehrere ähnliche Antworten. Als weiterer Aspekt wurden auch das verbesserte logische Verständnis sowie die höhere Motivation im PP genannt:

> *Man hat einen zweiten Kopf, der anders denkt und vielleicht hinter manche Logiken schneller kommt als du. Zu zweit ist man einfach schneller beim Erarbeiten neuer Sachen. Man kann sich gegenseitig motivieren. (Team 2: S(m)2)*

Vorteile von PP (textbasierte Programmierung)

Wenig überraschend wurde als Vorteil von PP bei der textbasierten Programmierung von (fast) allen Schüler*innen die Bedeutung von Syntax-Fehlern bei Python genannt, wie S(w)4 beschreibt:

> *Meine Partnerin hat mich auf beispielsweise Tippfehler aufmerksam gemacht und wir konnten beide unser Wissen nutzen und uns gegenseitig ausbessern. Es war außerdem eher lustig als deprimierend, wenn wir beide keine Ahnung mehr hatten wo wir in den Codes sind oder wo der Fehler ist. (Team 4: S(w)4)*

Interessant ist die Aussage, wonach gerade die Fehlersuche bei Python im PP als „*eher lustig als deprimierend*“ beschrieben wird. Visuelle Programmiersprachen gelten gerade deshalb als so beliebt, weil Syntax-Fehler vermieden werden können. (vgl. Wolber 2011: 603)

Dass es gerade bei der textbasierten Programmierung, wo die Behebung von Syntax-Fehlern besonders wichtig ist, im Arbeitsmodus Pair Programming besonders positive Effekte auf die Motivation der Lernenden gibt, wäre ein naheliegender Schluss.

Herausforderungen (textbasierte Programmierung)

Die spezifischen Herausforderungen bei der textbasierten Programmierungen wurden von den meisten Schüler*innen sehr ähnlich nach dem Beispiel von S(m)7 beschrieben:

> *Ich verwechsle die verschiedenen Codes oft miteinander, oder schreibe sie falsch hin und finde dann den Fehler nicht, probiere dann neue Codes aus und ruiniere im Endeffekt das ganze Programm, weil ich mich selbst nicht mehr auskenne. (Team 6: S(m)7)*

Diese Herausforderung wird einerseits aus zahlreichen Antworten der Schüler*innen deutlich, andererseits auch aus den Ergebnissen der von den Schüler*innen erstellten Programmen. Das zum Teil deutlich schlechtere Abschneiden der Schüler*innen bei den fachdidaktischen Konzepten zur textbasierten Programmierung ist ein Hinweis auf den genannten Faktor.

Motivation

Viele Schüler*innen beschreiben positive Effekte auf die Motivation im Arbeitsmodus Pair Programming – und zwar unabhängig von der Art der Programmierung (visuelle oder textbasierte Programmierung):

> *Ich weiß nicht mehr genau wobei mir meine Partnerin geholfen hat, aber ich weiß, dass ich ohne ihre Hilfe auf jeden Fall um einiges länger gebraucht hätte. Außerdem war es viel lustiger zu zweit, als allein zu arbeiten. (Team 1: S(w)1)*

Die Beschreibung von S(w)1 kommt in ähnlicher Form in verschiedenen Aussagen der Schüler*innen vor. Bemerkungen zu eventuell geringerer Motivation im PP kommen nicht vor.

Nachteile von PP (visuelle und textbasierte Programmierung)

Als möglichen Nachteil von PP wurde etwa von S(m)8 angemerkt, dass es bei Unstimmigkeiten im Team oder ungleichem Wissensstand zu Konflikten kommen kann.

> *Einer hat manchmal mehr gemacht als der andere, weil der eine sich manchmal gar nicht ausgekannt hat. Es kann zu Uneinigkeiten kommen, weil jeder es auf seiner Art fertig stellen will. (Team 6: S(m)8)*

In der SLR wurden verschiedene Studien präsentiert, die sich genau mit diesem Konfliktpotential beschäftigen, wobei die Zusammensetzung im Team als wichtiger Faktor für positive Lerneffekte bezeichnet wird. (siehe dazu Maguire, Maguire und Hyland 2014)

Besonders interessant ist, dass mehrere Schüler*innen angeben, dass PP zwar definitiv zu mehr Spaß am Programmieren führt, gleichzeitig aber die „Leistung" geringer wird, wie S(m)4 in dieser Antwort anführt:

> *Es kann jeweils nur einer arbeiten und der andere sitzt daneben und gibt nur Anweisungen. Man lenkt sich gegenseitig ab. Man hat natürlich mehr Spaß zusammen, das heißt aber halt auch dass man wieder langsamer arbeitet. Außerdem sollte man sich gut verstehen und auch gut gelaunt sein, wenn man 2-4 Stunden mit einander arbeitet. (Team 3: S(m)4)*

Dieses Argument wurde von mehreren Schüler*innen als möglicher Nachteil von PP genannt.

Zusammenfassend kann festgehalten werden, dass der Fragebogen einerseits wichtige Daten für die Auswahl der Interviewpartner*innen sowie Themenbereiche für die qualitativen Interviews liefert. Andererseits deuten die Ergebnisse – auch wenn aufgrund der geringen Stichprobe keine quantitative Analyse gemacht werden kann – darauf hin, dass es durchaus Unterschiede bei Einsatz von PP zwischen textbasierten und visuellen Programmiersprachen geben könnte.

Anhand der Auswertung der qualitativen Fokusinterviews im nächsten Kapitel werden diesbezüglich deutlichere Aussagen getroffen.

Auswertung der Fokusinterviews

Die Auswahl der Interviewpartner*innen erfolgte nach einer ersten Auswertung der Fragebögen sowie der erstellten Programme, um ein möglichst breites Spektrum an unterschiedlichen Erfahrungen der Schüler*innen zugänglich zu machen.

Folgenden Teams wurden nach diesen Kriterien ausgewählt, wobei sich alle Schüler*innen mit einem Interview einverstanden erklärten:

- Team 1 mit S(w)1 und S(w)2
- Team 3 mit S(m)3 und S(m)4
- Team 6 mit S(m)7 und S(m)8

Die Interviews wurden einzeln mit den Schüler*innen geführt und mit einem Audio-Aufnahmegerät aufgezeichnet. Die Aufnahmen wurden transkribiert und gemäß den gängigen Regeln codiert. (siehe dazu Kuckartz 2008: 39 f.)

Wie in Kapitel 2.3 beschrieben wurden diese Kategorien deduktiv auf Basis der Erkenntnisse der SLR gebildet, wobei natürlich der Fokus gemäß der Forschungsfrage angepasst wurde. Im nächsten Schritt wurden aus den transkribierten Interviews passende Textstellen den jeweiligen Kategorien zugeordnet, wobei die Relevanz der Textstellen der maßgebende Faktor für eine etwaige Zuordnung war. (vgl. Kuckartz 2008: 39 f.)

Für die kategorienbasierte Auswertung folge ich dem Vorschlag von Kuckartz (2008: 43 f.):

1. Zusammenfassung der Textstellen (je Kategorie)
2. Beschreibung der Ergebnisse (je Kategorie)
3. Interpretative Einordung und Einbettung in einen theoretischen Kontext
4. Zusammenfassende Diskussion und Beantwortung der Forschungsfrage

In diesem Kapitel werden die ersten zwei Punkte dieser Vorgehensweise behandelt, die Interpretation und Beantwortung der Forschungsfrage werde ich im nächsten Kapitel vornehmen.

	Kategorie	Definition	Codierregel
OK1	Herausforderungen	Generelle Schwierigkeiten, die in der Bearbeitung der Projekte aufgekommen sind.	
UK1.1	Herausforderungen MIT App Inventor	Spezielle Herausforderungen bei der visuellen Programmierung	Umfasst Textstellen mit expliziten Aussagen zum MIT App Inventor
UK1.2	Herausforderungen Python	Spezielle Herausforderungen bei der textbasierten Programmierung	Umfasst Textstellen mit expliziten Aussagen zu Python
OK2	Effekte von PP allgemein	Generelle, subjektiv erlebte Effekte von PP	
UK2.1	Vorteile PP allgemein	Spezielle, vorteilhafte Effekte beim PP	
UK2.2	Motivation PP allgemein	Subjektiv erlebte Effekte auf die Motivation bei PP	
UK2.3	Nachteile PP allgemein	Subjektiv erlebte negative Effekte bei PP	
OK3	Unterschiede PP textbasierte und visuelle Programmierung	Wahrgenommene Unterschiede beim PP zwischen visueller und textbasierter Programmierung	Umfasst nur Textstellen, in denen Unterschiede zwischen visueller und textbasierter Programmierung erwähnt werden.
UK3.1	Unterschiede bezüglich der Motivation	Wahrgenommene Unterschiede in Bezug auf die Motivation	
UK3.2	Unterschiede bezüglich der Qualität der Programme	Wahrgenommene Unterschiede in Bezug auf die Qualität der Programmierung	
UK3.3	Unterschiede bezüglich der Lösung von Problemen	Wahrgenommene Unterschiede in Bezug auf die Problemlösung	

Kategorie: Herausforderungen bei der Programmierung

> *Beim App Inventor ist nur manchmal das Problem, dass es so viele Funktionen gibt [...] aber wirklich schwierig ist es nicht, im Vergleich zu Python. (S(m)4, Zeile 9)*

Die Ergebnisse der Fokusinterviews ergeben ein sehr deutliches Bild in Bezug auf die Herausforderungen, die von den Schüler*innen bei der Bearbeitung der fachdidaktischen Konzepte beschrieben wurden. Nur ein einziger Schüler hat auf Herausforderungen beim MIT App Inventor hingewiesen:

> *Ich fand bei App Inventor [...] manchmal war es extrem schwer, die Zusammenhänge zu verstehen, weil man so viel miteinander verbinden muss, die Bilder mit dem Code und so weiter. (S(m)8, Zeile 8)*

Der Schüler S(m)8 hatte, das zeigte auch die Auswertung der erstellten Programme, sowohl bei der textbasierten als auch bei der visuellen Programmierung relativ zu den anderen Schüler*innen große Probleme bei der Bearbeitung der Aufgaben. Im Gegensatz zum MIT App Inventor haben bei der Programmierung mit Python fast alle Schüler*innen auf unterschiedliche Herausforderungen hingewiesen. Besonders oft wurde die Problematik von Syntax-Fehlern sowie die Verschachtelung von Schleifen und Bedingungen in Python genannt:

> *Ich fand Python schon schwieriger, weil man sich auf Tippfehler sehr konzentrieren muss. (S(w)2, Zeile 14)*

Die genannten Herausforderungen wurden von fast allen Interviewten genannt, es zeigt sich also, dass in der Wahrnehmung der Schüler*innen die Programmierung mit Python deutlich mehr Herausforderungen im Vergleich zur Programmierung mit dem MIT App Inventor mit sich bringt.

Kategorie: Effekte von PP (allgemein)

> *Ich glaube schon, dass ich motivierter war [...] Wir haben uns ja auch gegenseitig motiviert, wenn etwas nicht funktioniert hat. (S(m)3, Zeile 17)*

Die Effekte von PP waren in der Auswertung der Fokusinterviews in drei verschiedene Unterkategorien unterteilt: UK2.1 allgemeine Vorteile von PP, UK2.2 Effekte auf die Motivation und UK3.3 allgemeine Nachteile von PP. Allen Unterkategorien konnten zahlreiche Textstellen aus den transkribierten Interviews zugewiesen werden. Zur

Unterkategorie UK2.1 wurden zahlreiche Effekte genannt, die bereits in den Ergebnissen der SLR als bekannte Vorteile von PP genannt wurden:

- Höhere Motivation und mehr Spaß an der Programmierung (z.B. S(m)3, Zeile 10)
- Fehlersuche wird einfacher (z.B. S(w)1, Zeile 22)
- Schnellere Programmierung (z.B. S(m)7, Zeile 13)

Diese Vorteile wurden sowohl im Kontext der textbasierten als auch bei der visuellen Programmierung genannt und wurden somit von den Interviewten bei beiden Arbeitsformen wahrgenommen. Folgendes Zitat beschreibt die Grundeinstellung der Interviewten zum PP:

> *Grundsätzlich ist es zu zweit einfach besser [...] es macht mehr Spaß und man lernt voneinander. (S(w)2, Zeile 16)*

Auch der Unterkategorie UK2.2 wurden viele relevanten Textstellen zugeordnet. Diese Unterkategorie war notwendig, weil die Kategorie „Motivation" als positiver Effekt von PP besonders oft genannt wurde. Stellvertretend für viele ähnliche Beschreibungen der Schüler*innen steht folgendes Zitat von S(w)1:

> *Es macht mehr Spaß, weil man sich einfach mit der anderen Person austauschen kann. (S(w)1, Zeile 11)*

Dieses Empfinden wurde von allen Interviewten in unterschiedlicher Deutlichkeit geäußert, negative Effekte auf die Motivation wurden nicht beschrieben.

Besonders interessant war die Auswertung von relevanten Textstellen zur Unterkategorie UK2.3: Insgesamt wurden sieben Textstellen aus den transkribierten Interviews dieser Kategorie zugeordnet. Dabei stammen die meisten Beschreibungen von Schüler*innen, deren Programme im Verhältnis zum Mittelwert mit vielen Punkten bewertet wurden – d.h. von Schüler*innen, die überwiegend gut mit den fachdidaktischen Konzepten zurechtgekommen sind. Zusammenfassend wurden als Nachteile von PP folgende Punkte genannt:

- PP führt zu „langsamer" Programmierung (z.B. S(w)1, Zeile 8)
- Bei Unstimmigkeiten im Team wird es schwierig (z.B. S(w)2, Zeile 12)
- Ungleiches Lerntempo (z.B. S(m)3, Zeile 27)
- Höhere Konzentration, wenn man allein arbeitet (z.B. S(m)4, Zeile 10)

Folgendes Zitat von S(w)2 finde ich in diesem Kontext besonders relevant:

> *Manchmal hat es mich genervt, wenn ich genau wusste, wie es geht [...] und dann wollte M. etwas anderes [...] aber am Ende hat es dann auch mit ihrer Version geklappt. (S(w)2, Zeile 20)*

Die Ergebnisse dieser Unterkategorie zeigen, dass die interviewten Schüler*innen durchaus auch Nachteile in der Arbeitsform PP sehen, wobei vor allem jene Schüler*innen sich dazu geäußert haben, die mit den fachdidaktischen Konzepten (sowohl im PP als auch im SP) gut zurechtgekommen sind.

Kategorie: Unterschiede von PP (textbasierte und visuelle Programmierung)

> *Fehlersuchen ist viel schwieriger bei Python, wenn man zu zweit ist, ist es schon besser [...] weil einfach die Chance größer ist, dass man etwas findet. (S(m)3, Zeile 20)*
>
> *Bei Python ist es vielleicht besser, gemeinsam zu arbeiten, weil man da mehr Fehler macht. Aber ich glaube ich habe es allein auch ganz gut geschafft (S(w)2, Zeile 24)*

Obwohl sich aus den vorangegangenen Kategorien schon Unterschiede zwischen textbasierter und visueller Programmierung mit der Arbeitsform Pair Programming andeuten, wurden relevante Textstellen zu diesem Thema in einer eigenen Kategorie mit den Unterkategorien UK3.1 Motivation, UK3.2 Qualität der Programme und UK3.3 Problemlösung gesammelt. Die Auswertung der Textstellen zeigt, dass durchaus Unterschiede von den Schüler*innen angesprochen wurden.

Der UK3.1 wurden fünf Textstellen zugeordnet, wobei sich alle Textstellen mit den spezifischen Herausforderungen von Python und den damit einhergehenden Vorteilen von PP als Arbeitsform befassen, wie folgendes Zitat deutlich macht:

> *Bei Python war ich allein nicht so motiviert [...] Ich bin einfach nicht weitergekommen. Beim App Inventor habe ich es allein besser geschafft. (S(m)8, Zeile 25)*

Dieses Zitat ist auch deswegen besonders relevant, weil dieser Schüler – wie auch die Auswertung der erstellten Programme zeigt – Probleme bei der Bearbeitung der fachdidaktischen Konzepte hatte und im PP deutlich bessere Leistungen erbrachte als im SP. Einen anderen Aspekt zeigt folgendes Zitat der Schülerin S(w)2:

> *Ich war bei Python allein schon auch motiviert, da war es ganz cool alles selbst zu machen. Beim App Inventor habe ich nicht so gerne allein gearbeitet, weil es da auch noch mehr Spaß macht gemeinsam rumzuspielen. (S(w)2, Zeile 28)*

Die Schülerin S(w)2 hat fast durchwegs die volle Punkteanzahl erreicht und beschreibt einen wichtigen Aspekt, den auch andere Schüler*innen in einer ähnlichen Form geäußert haben. Das Gefühl, ein Programm ganz allein zu erstellen kann auch zusätzlich motivierend sein und S(w)2 beschreibt, dass dieser Aspekt für sie bei der Programmierung mit Python besonders bedeutend war.

Der zweiten Unterkategorien (Qualität der Programme) wurden vier Textstellen zugeordnet. Zwei von drei haben eine ähnliche Aussage wie das folgende, von S(m)8 stammende Zitat:

> *Insgesamt war mein Python-Programm ziemlicher Schrott, das hat einfach nicht gut funktioniert. Das Programm, das wir gemeinsam gemacht haben, war schon besser [...] Beim App Inventor war der Unterschied nicht so krass. (S(m)8, Zeile 27)*

Hier beschreibt Schüler S(m)8 ganz konkret, dass für ihn der wahrgenommene Effekt von PP als Arbeitsform bei der textbasierten Programmierung viel deutlicher war als bei der visuellen Programmierung. Das dritte Zitat von S(w)1 geht in eine ganz andere Richtung:

> *Für mich war es kein so großer Unterschied. Es ist ja fast dasselbe, nur ein anderes Layout. Man muss halt selbst alles eingeben und hat nicht diese Bausteine wie beim App Inventor. (S(w)1, Zeile 18)*

Diese Schülerin hat persönlich keine großen Unterschiede in den zwei unterschiedlichen PP-Projekten erlebt, wobei es wichtig ist festzuhalten, dass S(w)1 fast durchgängig alle zu erreichenden Punkte bei allen Projekten auch tatsächlich erreicht hat.

Der dritten Unterkategorien (Problemlösung) wurden sechs Textstellen aus den transkribierten Interviews zugeordnet, wobei alle Textstellen im Wesentlichen durch folgendes Zitat von S(m)8 repräsentiert werden:

> *Gerade bei Python hat mir mein Kollege viel geholfen, weil er ist ein bisschen [...] nicht so chaotisch wie ich. Beim App Inventor war das nicht so schlimm, aber bei Python muss alles ordentlich sein, sonst geht gar nichts. (S(m)8, Zeile 22)*

Der Schüler S(m)8, der mit den fachdidaktischen Konzepten im Vergleich zu den anderen Schüler*innen Probleme hatte, beschreibt in diesem Zitat den spezifischen Vorteil von PP als Arbeitsform sehr deutlich war.

Das folgende Zitat von S(m)3 hat eine ähnliche Aussage:

Fehlersuchen ist viel schwieriger bei Python, wenn man zu zweit ist, ist es schon besser [...] weil einfach die Chance größer ist, dass man etwas findet. (S(m)3, Zeile 20)

Bei Python waren auch für S(m)3 die wahrgenommenen Vorteile der Arbeitsform PP deutlich größer als bei der Arbeit mit dem MIT App Inventor. In unterschiedlichen Formen findet sich diese Aussage in fast allen Textstellen dieser Unterkategorie, wobei auffallend ist, dass sich vor allem Schüler*innen dazu geäußert haben, deren Programme mit einer mittleren bis niedrigen Punkteanzahl bewertet wurden.

Nachdem in diesem Kapitel die in der empirischen Untersuchung gesammelten Daten ausgewertet wurden, werde ich im nächsten Kapitel diese Auswertung interpretieren und in einen Kontext mit den in der SLR behandelten Studien setze. Auf dieser Basis werden die Ergebnisse im Hinblick auf die Beantwortung der Forschungsfrage vorgestellt und diskutiert.

5. ERGEBNISSE UND INTERPRETATION

5.1. Beantwortung der Forschungsfragen

Ziel dieser Arbeit war es, die unterschiedlichen Effekte von Pair Programming als Arbeitsform für den einführenden Programmierunterricht von textbasierten und visuellen Programmiersprachen zu erforschen. Die zentrale Forschungsfrage wurde in Kapitel 1 folgend formuliert:

> ZF: *Wie unterscheiden sich die Effekte von Pair Programming als Arbeitsform im einführenden Unterricht einer textbasierten und einer visuellen Programmiersprache?*

Um diese Forschungsfrage beantworten zu können, wurde zunächst eine SLR (Systematische Literaturrecherche) zur relevanten Literatur durchgeführt, deren Ergebnis wichtige Erkenntnisse zum Forschungsstand lieferte, die in Kapitel 2 zusammengefasst sind. Im nächsten Schritt wurde das Design der qualitativen Forschungsmethoden entworfen (Kapitel 3) sowie das empirische Forschungskonzept vorgestellt (Kapitel 4). In Kapitel 5 wurden zunächst die Durchführung der empirischen Untersuchung beschrieben. In Kapitel 5.2 wurden die gesammelten Daten schließlich für drei Bereiche ausgewertet:

1. Auswertung der erstellten Programme
2. Auswertung der Fragebögen
3. Auswertung der Fokusinterviews

Die Ergebnisse der Auswertung lieferten gemeinsam mit den theoretischen Erkenntnissen aus Kapitel 2 die Basis für die Beantwortung der Teil-Forschungsfragen TF1 bis TF3 sowie der zentralen Forschungsfrage ZF.

> TF1: *Inwiefern beeinflussen die spezifischen Herausforderungen von textbasierten und visuellen Programmiersprachen den erfolgreichen Einsatz von Pair Programming?*

Anhand der Auswertung der von den Schüler*innen erstellten Programme konnte gezeigt werden, dass alle Teams im Durchschnitt mehr Punkte im Pair Programming Arbeitsmodus erreichten, wobei der Effekt bei der textbasierten Programmierung viel stärker ausgeprägt war. Hinsichtlich der Zielerreichung kann also festgehalten werden, dass PP als Arbeitsform

jedenfalls effektiv war, wobei sich dieser positive Faktor bei der textbasierten Programmierung nochmals verstärkt.

Gleichzeitig ergab die Auswertung der Fragebögen, dass einige Schüler*innen durchaus kritisch sind, was die Effizienz von PP angeht:

> *Es kann jeweils nur einer arbeiten und der andere sitzt daneben und gibt nur Anweisungen. Man lenkt sich gegenseitig ab. Man hat natürlich mehr Spaß zusammen, das heißt aber halt auch dass man wieder langsamer arbeitet. Außerdem sollte man sich gut verstehen und auch gut gelaunt sein, wenn man 2-4 Stunden miteinander arbeitet. (Team 3: S(m)4)*

S(m)4 spricht in diesem Zitat eine Befürchtung im Zusammenhang mit der Effizienz von Pair Programming an, die in unterschiedlicher Form im Verlauf der empirischen Untersuchung sowohl zur textbasierten als auch zur visuellen Programmierung immer wieder geäußert wurde – so gaben immerhin 57% der Schüler*innen (also 8 von 14) an, bei Python im SP mehr zu lernen – beim MIT App Inventor war der Prozentsatz sogar 71% oder 10 von 14 Schüler*innen.

Dazu muss festgehalten werden, dass die Ergebnisse der SLR in diesem Bereich ambivalent sind: Einerseits wird die Effizienz von PP in einigen Studien durchaus kritisch gesehen (siehe zum Beispiel Faja 2014), andererseits ist es gerade bei diesem Faktor notwendig, etwas genauer hinzusehen – schließlich ist die Messung der „Effizienz" von Lernprozessen deutlich komplexer als die Messung der Zeit, die Lernende für die Bewältigung definierter Aufgaben brauchen.

Die Auswertung der Fokusinterviews zeigte einen weiteren Aspekt in Bezug auf die Effektivität von Pair Programming: Die wahrgenommenen Unterschiede zwischen der textbasierten und der visuellen Programmierung unterschieden sich teilweise recht deutlich, wie folgende Zitate zeigen:

> *Für mich war es kein so großer Unterschied. Es ist ja fast dasselbe, nur ein anderes Layout. Man muss halt selbst alles eingeben und hat nicht diese Bausteine wie beim App Inventor. (S(w)1, Zeile 18)*

> *Bei Python war ich allein nicht so motiviert [...] Ich bin einfach nicht weitergekommen. Beim App Inventor habe ich es allein besser geschafft. (S(m)8, Zeile 25)*

Für die Interpretation dieser Aussagen ist es wichtig, den Kontext zu beachten: Während S(w)1 in fast allen getesteten Bereichen die volle Punkteanzahl für die erstellten Programme erhielt (sowie die volle Anzahl bei der Programmier-Vorerhebung), schnitt S(m)8 in fast allen

Bereichen schlechter ab als die anderen Teilnehmer*innen. Obwohl beide Interviewten grundsätzlich angeben, dass PP als Arbeitsform positive Effekte hat, wird das Ausmaß dieser Effekte doch unterschiedlich angegeben: S(m)8 beschreibt große Unterschiede zwischen textbasierter und visueller Programmierung im SP, während S(w)1 keine großen Unterschiede wahrnimmt. Für S(m)8 war die subjektiv wahrgenommene Effektivität von PP dementsprechend viel höher, weil die Herausforderungen der textbasierten Programmierung für ihn sehr groß waren. S(w)1 hatte dagegen weder mit der textbasierten noch mit der visuellen Programmierung größere Probleme und hat die unterschiedlichen Effekte von PP weniger deutlich wahrgenommen.

Diese Erkenntnis scheint für die Beantwortung von TF1 besonders relevant: Pair Programming führt sowohl bei der textbasierten als auch bei der visuellen Programmierung zu positiven Effekten auf die Effektivität, wobei diese Effekte bei der textbasierten Programmierung in meiner empirischen Untersuchung stärker ausgeprägt waren. Gleichzeitig zeigte die Auswertung der Programme und jene der Fokusinterviews, dass gerade jene Lernenden besonders von der Arbeitsform Pair Programming profitieren könnten, für die das Erlernen einer Programmiersprache besonders herausfordernd ist.

> TF2: ***Wie unterscheiden sich die Erfahrungen von Schüler*innen im Hinblick auf die im Vortest erhobenen Leistungsniveaus?***

Die Auswertung der empirischen Untersuchung zeigt im Zusammenhang mit TF2 wesentliche Unterschiede im Hinblick auf die Motivation der Lernenden. Grundsätzlich zeigen die Ergebnisse, dass Pair Programming sowohl in der visuellen als auch in der textbasierten Programmierung zu positiven Effekten auf die Motivation und das Selbstvertrauen der Lernenden führt.

In der Auswertung der Fragebögen zeigte sich, dass vor allem die Interaktion und das gegenseitige Feedback als motivierend empfunden wurden, wie folgendes Zitat exemplarisch zeigt:

> *Ich weiß nicht mehr genau wobei mir meine Partnerin geholfen hat, aber ich weiß, dass ich ohne ihre Hilfe auf jeden Fall um einiges länger gebraucht hätte. Außerdem war es viel lustiger zu zweit, als allein zu arbeiten. (Team 1: S(w)1)*

Der Schülerin S(w)1 ist vor allem in Erinnerung geblieben, dass sie im Pair Programming Arbeitsmodus mehr Spaß an der Programmierung hatte. Da diese Schüler*in – wie die

Auswertung der Programme und die Ergebnisse der Fokusinterviews zeigen – insgesamt sehr gut mit den fachdidaktischen Konzepten zurechtgekommen ist, zeigt diese Aussage das Potential von PP gerade für leistungsstarke Schüler*innen. Gleichzeitig zeigt die Auswertung der Fokusinterviews, dass für die leistungsstärkeren Schüler*innen die Unterschiede zwischen der textbasierten und der visuellen Programmierung in Bezug auf die Motivation und das Selbstvertrauen der Lernenden eher gering sind und dass manche Schüler*innen es auch motivierend finden, Programme ganz allein zu erstellen. Folgende Aussage von S(w)2 ist ein Beispiel für diesen Zusammenhang:

> *Ich war bei Python allein schon auch motiviert, da war es ganz cool alles selbst zu machen. Beim App Inventor habe ich nicht so gerne allein gearbeitet, weil es da auch noch mehr Spaß macht gemeinsam rumzuspielen. (S(w)2, Zeile 28)*

Die Auswertung der Fokusinterviews zeigt, dass diese Einschätzung nicht für alle Schüler*innen gilt. Insbesondere bei Schüler*innen, die im Vortest im Vergleich zu den anderen Proband*innen wenig Punkte erzielten zeigte sich, dass sie deutliche Unterschiede in Bezug auf die Motivation und das Selbstvertrauen zwischen textbasierter und visueller Programmierung im PP Arbeitsmodus erlebten, wie folgendes Zitat zeigt:

> *Da war es schon gut, zu zweit zu programmieren. Allein war ich da oft verloren und wollte nicht mehr weitermachen. (S(m)8, Zeile 13)*

Der Schüler S(m)8 bezieht sich in dieser Aussage auf die fachdidaktischen Konzepte zur textbasierten Programmierung mit Python. Anders als die Schülerin S(w)2 im vorherigen Zitat spricht der Schüler S(m)8 ein negatives Gefühl („sich verloren fühlen“) an, welches die Motivation negativ beeinflusste. Im Pair Programming Arbeitsmodus wurde dieses Gefühl abgeschwächt, wobei die Interpretation naheliegt, dass der Austausch mit seinem Teamkollegen zu Problemen und Lösungsansätzen sich positiv auf die Motivation auswirkte. Ähnlich wie bei der Frage nach der Effektivität und Effizienz zeigt sich also auch in Bezug auf die Motivation und das Selbstvertrauen der Lernenden ein positiver Effekt von Pair Programming, wobei dieser Effekt gerade bei der textbasierten Programmierung bei leistungsschwächeren Schüler*innen deutlicher ausgeprägt sein dürfte. Gleichzeitig kann aber bei besonders leistungsstarken Schüler*innen beobachtet werden, dass auch die Bearbeitung von Aufgaben im Single Programming Arbeitsmodus sich positiv auswirken

kann, weil für Schüler*innen das völlig eigenständige Erstellen von Programmen auch ein zusätzlich motivierender Faktor sein kann.

> TF3: *Inwiefern unterscheiden sich die Erfahrungen im Pair Programming bei der Lösung von Problemen zwischen textbasierten und visuellen Programmiersprachen?*

Die Entwicklung der Kompetenz Probleme, die bei der Programmierung von Algorithmen entstehen, lösen zu können, ist ein zentrales Element für den einführenden Programmierunterricht. Anhand der Auswertung der Fragebögen zeigte sich in diesem Zusammenhang ein deutliches Bild: Während bei der Arbeit mit dem MIT App Inventor nur 6 von 14 Schüler*innen angaben, den Tutorials im Pair Programming Arbeitsmodus besser folgen zu können, erhöhte sich die Anzahl bei der textbasierten Programmierung mit Python deutlich auf 12 von 14 Schüler*innen.

Dieses Ergebnis lässt darauf schließen, dass sich die Schüler*innen im Pair Programming Arbeitsmodus insbesondere bei der Bearbeitung des fachdidaktischen Konzeptes zur textbasierten Programmierung gegenseitig helfen konnten. Diese Vermutung wird auch durch die Auswertung der erstellten Programme unterstützt: Hier zeigte sich, wie bereits erwähnt, gerade bei der textbasierten Programmierung ein großer Unterschied zwischen den Arbeitsmodi Single Programming und Pair Programming.

In den Fokusinterviews wurde der Frage nachgegangen, wie Schüler*innen das Lösen von Problemen im Pair Programming erlebt haben. Die Auswertung der Kategorien zeigte, dass das Pair Programming gerade in der textbasierten Programmierung für viele Schüler*innen besonders hilfreich war. Die Gründe für diese Wahrnehmung scheinen vielfältig zu sein: Schülerin S(w)4 stellt im folgenden Zitat etwa einen Zusammenhang zwischen Problemlösung und Motivation her:

> *Meine Partnerin hat mich auf beispielsweise Tippfehler aufmerksam gemacht und wir konnten beide unser Wissen nutzen und uns gegenseitig ausbessern. Es war außerdem eher lustig als deprimierend, wenn wir beide keine Ahnung mehr hatten wo wir in den Codes sind oder wo der Fehler ist. (Team 4: S(w)4)*

Die Schülerin beschreibt hier einen Prozess, der vielfach als demotivierend empfunden wird: Die Behebung von Syntax-Fehlern bei textbasierten Programmiersprachen. Diese Aussage ist ein Indiz dafür, dass es neben dem augenscheinlichen Effekt, dass *„[…] vier Augen immer mehr*

[sehen] als zwei […]“ (S(m)3, Zeile 14) noch einen weiteren positiven Effekt von Pair Programming im Kontext der Problemlösung gibt. Dass die Suche nach Lösungen im Pair Programming mehr Spaß macht, könnte zu einer höheren Resilienz und einer geringeren Frustration bei den Schüler*innen führen. Diese Faktoren könnten wiederum die Bereitschaft der Schüler*innen, sich diesen Problemen für längere Zeit konzentriert zu widmen, positiv beeinflussen.

Ein anderer Effekt scheint bei leistungsschwächeren Schüler*innen relevant zu sein. Visuelle Programmiersprachen verhindern durch ihre Struktur, dass Syntax-Fehler zu Problemen führen, was gerade bei leistungsschwächeren Schüler*innen ein wichtiger Erfolgsfaktor für den einführenden Programmierunterricht zu sein schein. (vgl. Wolber 2011: 603) Bei der textbasierten Programmierung fällt diese Struktur weg, was – das zeigte sowohl die Auswertung der erstellten Programme als auch die Auswertung der Fragebögen – bei einigen Schüler*innen zu großen Herausforderungen bei der Problemlösung führt. Folgendes Zitat von S(m)8 zeigt in diesem Zusammenhang das besondere Potential von Pair Programming:

> *Gerade bei Python hat mir mein Kollege viel geholfen, weil er ist ein bisschen [...] nicht so chaotisch wie ich. Beim App Inventor war das nicht so schlimm, aber bei Python muss alles ordentlich sein, sonst geht gar nichts. (S(m)8, Zeile 22)*

Die Zusammenarbeit hat diesem Schüler geholfen, bei der textbasierten Programmierung eine bessere Strukturierung des Codes zu erreichen, was die Lösung von Problemen erleichtert. Diese Erkenntnis kann wiederum als Indiz für besonders positive Effekte von Pair Programming bei der textbasierten Programmierung mit leistungsschwächeren Schüler*innen gesehen werden.

Aus den in diesem Kapitel beschriebenen Ergebnissen im Rahmen der Beantwortung der Teil-Forschungsfragen TF1 bis TF3 lässt sich nun die zentrale Forschungsfrage ZF beantworten. Die Ergebnisse dieser Arbeit legen nahe, dass es entscheidende Unterschiede in Bezug auf die Effekte von Pair Programming als Arbeitsform im einführenden Unterricht einer textbasierten und einer visuellen Programmiersprache gibt. Die Unterschiede betreffen verschiedene Faktoren, die in vorangegangenen empirischen Arbeiten (siehe Kapitel 2) als Gelingensbedingungen für den erfolgreichen Einsatz von Pair Programming definiert wurden.

Aus der Beantwortung der Teil-Forschungsfragen lassen sich zumindest drei zentrale Unterschiede mit weitreichenden Implikationen beschreiben:

1. Effektivität und Effizienz

Die Auswertung der erstellten Programme zeigt in Kombination mit der Auswertung der Fragebögen sowie der Fokusinterviews, dass die Schüler*innen insgesamt im Pair Programming effektiv und auch effizient gearbeitet haben. Gleichzeitig konnten deutliche Unterschiede zwischen der textbasierten und der visuellen Programmierung festgestellt werden: Während leistungsstarke Schüler*innen eher geringe Unterschiede zwischen textbasierter und visueller Programmierung im Pair Programming Arbeitsmodus wahrnahmen, zeigte sich bei leistungsschwächeren Schüler*innen viel deutlichere Effekte. Gerade Schüler*innen, für die alle fachdidaktischen Konzepte sehr herausfordernd waren, profitierten bei der textbasierten Programmierung sehr viel stärker vom Pair Programming als bei der visuellen Programmierung. Diese Erkenntnis ist für den einführenden Programmierunterricht besonders relevant: Schüler*innen, für die das Erlernen einer Programmiersprache grundsätzlich herausfordernd ist, profitieren demnach besonders von der Arbeitsform Pair Programming. Die Ergebnisse dieser Arbeit zeigen für die Teilnehmer*innen der empirischen Studien, dass dieser Effekt bei textbasierten Programmiersprachen noch deutlicher ausgeprägt ist als bei visuellen Programmiersprachen.

2. Motivation und Selbstvertrauen

Es ist nicht verwunderlich, dass in der empirischen Untersuchung positive Effekte von Pair Programming auf die Motivation und das Selbstvertrauen der Lernenden ermittelt werden konnten. Die Ergebnisse der SLR zeigten für unterschiedliche Zielgruppen und Kontexte ähnliche Effekte. (vgl. Salleh, Mendes und Grundy 2011: 11) In Bezug auf Unterschiede zwischen textbasierter und visueller Programmierung zeigten sich aber in den Fokusinterviews zwei konträre Erfahrungen der Lernenden: Einerseits scheint der positive Effekt von Pair Programming auf die Motivation und das Selbstvertrauen besonders für leistungsschwache Schüler*innen bei der textbasierten Programmierung stärker ausgeprägt zu sein als bei der visuellen Programmierung. Andererseits scheint gerade für leistungsstarke Schüler*innen ein gegenteiligen Effekt auftreten zu können, weil die Arbeit mit einer komplexeren Umgebung und das Tippen von Code als positive (individuelle) Herausforderung gesehen wird.

3. Problemlösung

Es ist augenscheinlich, dass die Bedeutung der Problemlösungskompetenz für den einführenden Programmierunterricht steigt, sobald Schüler*innen die an sie gestellten Aufgaben als besonders herausfordernd empfinden. Die Auswertung der Fokusinterviews zeigte, dass Pair Programming für die Schüler*innen vor allem bei der textbasierten Programmierung hilfreich für die Lösung von Problemen war. Die Gründe für diesen positiven Effekt von Pair Programming sind vielfältig: Völlig klar ist, dass „*[…] vier Augen immer mehr [sehen] als zwei*“, wie S(m)3 im Fokusinterview anmerkte. Darüber hinaus wurde die gemeinsame Fehlersuche als „lustig“ beschrieben, was auf eine höhere Motivation und in weiterer Folge auf eine höhere Frusttoleranz schließen lässt. Schließlich zeigte sich vor allem bei leistungsschwächeren Schüler*innen, dass die Zusammenarbeit im Pair Programming auch dabei helfen kann, den eigenen Code bei der textbasierten Programmierung zu strukturieren und zu ordnen, was die Lösung von Problemen natürlich einfacher macht. Nachdem visuelle Programmiersprachen die Struktur sehr viel stärker vorgeben, scheint dieser positive Effekt bei textbasierten Programmiersprachen sehr viel stärker ausgeprägt zu sein.

5.2. Überprüfung der Hypothesen

Die in Kapitel 1.2 aufgestellten Hypothesen zu den Effekten von Pair Programming im einführenden Programmierunterricht von visuellen und textbasierten Programmiersprachen lassen sich anhand der bisher gesammelten Erkenntnisse überprüfen.

> H1: *Die Effekte von Pair Programming sind im einführenden Programmierunterricht mit textbasierten Programmiersprachen insgesamt stärker ausgeprägt als mit visuellen Programmiersprachen.*

Diese Hypothese kann anhand der Ergebnisse aus Kapitel 5.1 verifiziert werden. Die Auswertung der von den Schüler*innen erstellten Programme zeigte, dass im PP im Durchschnitt mehr Punkte erreicht wurden als im SP, wobei dieser Effekt bei der textbasierten Programmierung viel deutlicher ausgeprägt war. Die Auswertung der Fragebögen sowie der Fokusinterviews zeigte für alle Themenbereiche stärkere Effekte bei der textbasierten Programmierung.

> H2: *Leistungsstarke Schüler*innen profitieren beim Pair Programming mit textbasierten Programmiersprachen stärker als bei visuellen Programmiersprachen.*

Diese Hypothese kann nicht verifiziert werden. Obwohl die positiven Effekte von Pair Programming in der empirischen Untersuchung auch für besonders leistungsstarke Schüler*innen festgestellt werden konnten, waren diese Effekte bei der textbasierten Programmierung nicht stärker ausgeprägt. Im Gegenteil gibt es Indizien dafür, dass leistungsstarke Schüler*innen die besonderen Herausforderungen der textbasierten Programmierung schätzen und deshalb in bestimmten Kontexten auch vom Solo Programming profitieren könnten.

> H3: *Für leistungsschwache Schüler*innen ist Pair Programming vor allem bei der textbasierten Programmierung hilfreich, um Probleme zu lösen.*

Die Ergebnisse aus Kapitel 5.1 zeigen klar, dass die positiven Effekte von Pair Programming – die bei allen Schüler*inne festgestellt werden konnten – insbesondere bei jenen Schüler*innen besonders stark sind, für die das Erlernen einer Programmiersprache sehr

herausfordernd ist. Die spezifischen Herausforderungen der textbasierten Programmierung führen dazu, dass die positiven Effekte von Pair Programming hier noch einmal verstärkt beobachtet werden können.

> H4: *Die Effekte auf Effektivität und Effizienz von Pair Programming sind bei leistungsstarken und leistungsschwachen Schüler*innen bei der textbasierten Programmierung ähnlich ausgeprägt.*

Diese Hypothese kann aus der vorliegenden Arbeit weder verifiziert noch falsifiziert werden. Einerseits zeigen die Ergebnisse der empirischen Untersuchung, dass leistungsschwache Schüler*innen bei der textbasierten Programmierung stärker profitieren als leistungsstarke Schüler*innen. Andererseits ist es denkbar, dass mit steigender Komplexität der Aufgaben die positiven Effekte von Pair Programming auch für leistungsstarke Schüler*innen deutlicher wahrnehmbar werden.

6. DISKUSSION

Die Ergebnisse dieser Arbeit zeigen, dass Pair Programming als Arbeitsform für den einführenden Programmierunterricht sowohl bei der textbasierten als auch bei der visuellen Programmierung zu positiven Effekten führt. Diese Ergebnisse stimmen mit den Erkenntnissen aus der SLR (siehe Kapitel 2.1) überein, in der 41 wissenschaftliche Studien zu diesem Themenbereich analysiert wurden. Besonders deutlich war in meiner empirischen Untersuchung der positive Effekt von Pair Programming auf die Motivation der Lernenden, der auch in zahlreichen Studien nachgewiesen wurde:

„*Almost all studies' findings reported that students' satisfaction was higher when using PP compared with working individually*". (Salleh, Mendes und Grundy 2011: 11)

Das spezifische Interesse meiner Arbeit lag in der Erforschung etwaiger Unterschiede dieser positiven Effekte zwischen textbasierter und visueller Programmierung. In Kapitel 5 wurde anhand der Auswertung der empirischen Daten gezeigt, dass die positiven Effekte von Pair Programming bei der textbasierten Programmierung stärker ausgeprägt sind als bei der visuellen Programmierung. Die Einschränkungen dieser Erkenntnis sowie Folgerungen und weitere Forschungen werde ich in Kapitel 6.1 und 6.2 diskutieren.

6.1. Einschränkungen und weitere Forschungen

In Kapitel 3 wurde das Design für die empirische Untersuchung vorgestellt. Diese Untersuchung lieferte die empirische Datenbasis für die Beantwortung der Forschungsfragen, die in Kapitel 1.2 vorgestellt wurde. Die Umsetzung von vier fachdidaktischen Konzepte zum einführenden Programmierunterricht mit einer textbasierten Programmiersprache (Python) sowie einer visuellen Programmiersprache (MIT App Inventor) in den zwei Arbeitsmodi (PP und SP) lieferte wertvolles Datenmaterial, beinhaltete aber auch einige Einschränkungen. Wie

in Kapitel 4 beschrieben wurde, lässt die Auswertung der erstellten Programme aufgrund der geringen Anzahl der Proband*innen keine quantitativen Aussagen zu. Weitere Forschungen, insbesondere auch mit quantitativen Forschungsinteressen, zu Unterschieden bei textbasierten und visuellen Programmiersprachen in Bezug auf den Arbeitsmodus Pair Programming wären wünschenswert und könnten wertvolle Erkenntnisse für die informatikdidaktische Praxis liefern.

Die Ergebnisse der empirischen Untersuchung zeigen weiters, dass Pair Programming gerade bei der textbasierten Programmierung zu höherer Motivation und besseren Ergebnissen führt. In diesem Kontext wären weitere Forschungen in Bezug auf die Komplexität der Aufgabenstellungen interessant: Denkbar wäre beispielsweise, dass die von den Lernenden wahrgenommene Komplexität mit der Einschätzung des „Erfolges" von Pair Programming durch die Lernenden korreliert. Gerade bei den leistungsstarken Schüler*innen in meiner Untersuchung zeigte sich, dass diese die Effekte des Pair Programmings bei den als „leicht" empfundenen Aufgabenstellungen des MIT App Inventors als eher gering eingeschätzt haben.

Nachdem die Proband*innen für meine empirische Untersuchung alle am Wahlpflichtfach Informatik teilnahmen, wäre auch eine weitere Forschung mit Proband*innen aus dem Pflichtgegenstand Informatik interessant. Dabei wäre vor allem die Frage spannend, ob sich die Auswirkungen von Pair Programming auf die Motivation der Lernenden ähnlich äußern wie in meiner Untersuchung, oder ob es Unterschiede gibt.

Eine weitere Einschränkung dieser Arbeit liegt darin, dass mögliche Gender-Aspekte nicht erforscht werden können. Die Ergebnisse der SLR zeigten, dass es zwar viele Studien zu Pair Programming und Gender gibt, diese jedoch sehr unterschiedliche und teils widersprüchliche Ergebnisse lieferten. (vgl. Zhong, Wang und Cheng 2016, 424 f.)

Es ist ein naheliegender Schluss, dass die positiven Effekte von Pair Programming – gerade im einführenden Programmierunterricht bei der textbasierten Programmierung – auch dazu führen könnten, Barrieren im Bereich Gender und

Coding abzubauen. Dieser Schluss würde an die Erkenntnisse von Liebenberg, Mentz und Breed (2012) anknüpfen: „*They [Mädchen, Anm.] especially enjoyed the socialization and communication brought about by pair programming. The assistance, support, motivation, focus and encouragement they received from partners when stuck or while fixing errors made the programming experience more enjoyable for them.*" (Liebenberg, Mentz und Breed 2012: 219) Diese Erkenntnisse anhand einer empirischen Untersuchung mit einem Fokus auf Unterschiede zwischen textbasierten und visuellen Programmiersprachen zu erweitern, wäre sowohl für die Fachdidaktik der Informatik als auch für Praktiker*innen in der schulischen und außerschulischen Informatikbildung jedenfalls lohnend.

Schließlich sehe ich eine wesentliche Einschränkung dieser Arbeit darin, dass – trotz des Versuchs, durch die „Challenges" unterschiedliche Leistungsniveaus anzusprechen – die Aufgaben in den fachdidaktischen Konzepten nicht ausreichend divers waren, um für alle Schüler*innen optimale Herausforderungen bieten zu können. Weitere Studien könnten Unterschiede nicht nur zwischen textbasierten und visuellen Programmiersprachen, sondern speziell zwischen unterschiedlichen Aufgabenformaten und Komplexitätslevels für diverse Zielgruppen untersuchen, um weitere Erkenntnisse zu den Effekten von Pair Programming im einführenden Programmierunterricht zu erlangen.

6.2. Folgerungen und Auswirkungen

Die Ergebnisse dieser Arbeit unterstützen die These, dass der Einsatz von Pair Programming im einführenden Programmierunterricht zu zahlreichen positiven Effekten führen kann. Der Vergleich zwischen den unterschiedlichen Effekten bei einer textbasierten und einer visuellen Programmiersprache legt den Schluss nahe, dass das Potential von Pair Programming gerade für jene Schüler*innen besonders groß ist, für die das Erlernen einer textbasierten Programmiersprache sehr herausfordernd ist.

Für die Praxis des Programmierunterrichts folgt aus den Ergebnissen, die in Kapitel 5 vorgestellt wurden, dass es einige wichtige Unterschiede zwischen textbasierten und visuellen Programmiersprachen gibt, die beim Einsatz von Pair Programming beachtet werden müssen. Insgesamt gilt, und diese Erkenntnis schließt an die Ergebnisse der SLR an, dass zahleiche „Gelingensfaktoren" beachtet werden müssen, um die Potentiale von Pair Programming möglichst gut nutzen zu können. Grundsätzlich unterstützen die Ergebnisse dieser Arbeit die These, dass diese Gelingensfaktoren für den erfolgreichen Einsatz von Pair Programming insbesondere im Bereich des Sozialen zu finden sind. Die Zusammensetzung der Paare ist hier an vorderster Stelle zu nennen: Die Bedeutung der persönlichen Beziehung ist dabei entscheiden, gleichzeitig sollten die Lernenden auch gemäß ihren Vorerfahrungen und ihrem Leistungspotential zugeteilt werden. Die Auswertung der Fokusinterviews zeigte, dass gerade bei leistungsstarken Schüler*innen darauf geachtet werden muss, dass die Herausforderungen der Aufgaben tatsächlich dem Leistungsvermögen der Lernenden entsprechen. Nur so können die positiven Effekte von Pair Programming – insbesondere im Hinblick auf die Lösung von Problemen und die Steigerung der Motivation – voll entfaltet werden. Bei Schüler*innen, für die das Erlernen einer Programmiersprache besonders herausfordernd ist, lässt sich aus den Ergebnissen der empirischen Forschung ein anderer Schluss treffen: Hier gilt, dass der Einsatz von Pair Programming besonders erfolgsversprechend ist, wobei eine wertschätzende und offene Kommunikation – insbesondere in Bezug auf mögliche Fehler und Verständnisprobleme – der Schlüssel zum Erfolg zu sein scheint. In Kombination mit Aufgaben, die den Schüler*innen genügend Hilfen für motivierende Lernerlebnisse bieten, ist Pair Programming gerade für diese Zielgruppe besonders gut für den einführenden Programmierunterricht geeignet.

LITERATURVERZEICHNIS

Bruner, Jerome S. (1967): On knowing: Essays for the left hand. Cambridge, Mass: Harvard University Press.

Chou, Pai H. (2002): Algorithm Education in Python, http://www.ece.uci.edu/~chou/pdf/chou-python02.pdf [Zugriff: 15.03.2020]

Cockburn, Alistair und Williams, Laurie (2000): The costs and benefits of pair programming. Extreme programming examined, pp. 223 – 247.

Fink, Arlene (2005): Conducting Research Literature Reviews: From the Internet to Paper (2. Ausgabe), Thousand Oaks, California: Sage Publications.

Geisler, Uwe (2010): Digitale Zaubereien im Informatikunterricht. Komplexe Informatik Themen als Science Show für die Sekundarstufe I, Brandhofer Gerhard u.a. (Hg.), 25 Jahre Schulinformatik in Österreich, Zukunft mit Herkunft, Wien: Österreichische Computer Gesellschaft, pp. 198 – 201.

Gläser, Jochen und Laudel, Grit (2009): Experteninterviews und qualitative Inhaltsanalyse als Instrumente rekonstruierender Untersuchungen, Wiesbaden: VS-Verlag.

Gomes, Anabela und Mendes, Antonio (2007): An environment to improve programming education, CompSysTech '07: Proceedings of the 2007 international conference on Computer systems and technologies, June 2007, pp. 1 – 6.

Gray, Jeff; Abelson, Hal; Wolber, David et al (2012): Teaching CS principles with App-Inventor. In: Proceedings of the 50th Annual Southeast Regional Conference, New York, NY, USA, 29th March 2012, pp. 405 – 406.

Hanks, Brian; Fitzgerald, Sue; McCauley, Renée et al (2011): Pair programming in education: a literature review, Computer Science Education, Vol. 21, Nr. 2, pp. 135 – 173.

Helfferich, Cornelia (2019): Leitfaden- und Experteninterviews, Baur, Nina und Blasius, Jörg (Hg.), Handbuch Methoden der empirischen Sozialforschung, Springer VS, Wiesbaden, pp. 669 – 686.

Hopf, Christel (2000): Qualitative Interviews – ein Überblick, Flick, Uwe; v. Kardorff, Ernst; Steinke, Ines (Hg.): Qualitative Sozialforschung, Reinbek: Rowohlt, pp. 349 – 359.

Joolingen, Wouter van (1999): Cognitive tools for discovery learning, International Journal of Artificial Intelligence in Education, Vol. 10, Nr. 1, pp. 385 – 397.

Kalogiannakis, Michail; Zaranis, Nicholas; Papadakis, Stamatios et al (2016): Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study, Int. J. Technology Enhanced Learning, Vol. 8, Nr. 3, pp. 217 – 237.

Kerres, Michael (2001): Multimediale und telemediale Lernumgebungen. Konzeption und Entwicklung, München: Oldenburg.

Kitchenham, Barbara und Charters, Stuart (2007): Guidelines for performing Systematic Literature Reviews in Software Engineering, Technical Report No. EBSE-2007-, Keele, UK: Keele University. http://www.dur.ac.uk/ebse/guidelines.php [Zugriff: 15.04.2020]

Klafki, Wolfgang (2007): Neue Studien zur Bildungstheorie und Didaktik, Zeitgemäße Allgemeinbildung und kritisch-konstruktive Didaktik, Weinheim und Basel: Beltz Verlag.

Kloss, Jörg H. (2011): Android-Apps Programmierung für Einsteiger. Mobile Anwendungen entwickeln mit AppInventor, München: Markt+Technik Verlag.

Krishnendu, Roy (2012): App-Inventor for android: report from a summer camp. In: Proceedings of the 43rd ACM technical symposium on Computer Science Education, New York, NY, USA, pp. 283-288.

Kuckartz, Udo (2008): Qualitative Evaluation: Der Einstieg in die Praxis, Hamburg: VS-Verlag.

Liebenberg, Janet; Ment, Elsa; Breed, Betty (2012) Pair programming and secondary school girls' enjoyment of programming and the subject Information Technology (IT), Computer Science Education, Vol. 22, Nr. 3, 219 – 236.

Maguire, Phil; Maguire; Rebecca; Hyland, Philip et al (2014): Enhancing collaborative learning using pair programming: Who benefits?, All Ireland Journal of Teaching and Learning in Higher Education, Vol. 6, Nr. 2, pp. 1411 – 14125.

Mayring, Phillip (2002): Einführung in die qualitative Sozialforschung, Eine Anleitung zum qualitativen Denken, Weinheim und Basel: Beltz.

Mayring, Philipp (2010): Qualitative Inhaltsanalyse, Grundlagen und Techniken, Weinheim und Basel: Beltz.

Nam, Dongsoo; Kim, Yungsik; Lee, Taewook (2010): The Effects of Scaffolding-Based Courseware for The Scratch Programming Learning on Student Problem Solving Skill, Proceedings of the 18th International Conference on Computers in Education ICCE2010, pp. 723 – 727.

Neber, Heinz (1981): Entdeckendes Lernen, Weinheim: Beltz.

Nepomuceno, Vilmar (2019): On the need to Update Systematic Literature Reviews, Information and Software Technology, Vol. 109, Nr. 4, pp. 40 – 42.

OECD (2005): Definition und Auswahl von Schlüsselkompetenzen, Zusammenfassung 2005. http://www.oecd.org/pisa/35693281.pdf [Zugriff: 3.08.2020]

Okoli, Chitu und Schabram, Kira (2010): A Guide to Conducting a Systematic Literature Review of Information Systems Research, Sprouts: Working Papers on Information Systems, Vol. 10, Nr. 26, http://sprouts.aisnet.org/10-26 [Zugriff: 12.04.2020]

Papert, Seymour und Harel, Idit (1991): Constructionism: research reports and essays 1985 – 1990, Norwood, NJ: Ablex Publications.

Perdikuri, Katerina (2014): Students' Experiences from the use of MIT App-Inventor in classroom, Proceedings of the 18th Panhellenic Conference on informatics, 2nd October 2014, Athens, Greece, pp. 1 – 6.

Przyborski, Aglaja und Wohlrab-Sahr, Monika (2008): Qualitative Sozialforschung, München: Oldenbourg.

Salleh, Norsaremah; Mendes, Emilia; Grundy, John (2011): Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review, IEEE Trans Software Engineering, Vol 37, Nr. 1, 509 – 525.

Schiller, Jeffrey; Turbak, Franklyn; Abelson, Hal et al (2014): Live Programming of Mobile Apps in App-Inventor. In: Proceedings of the 2nd Workshop on programming for mobile & touch, 20th October 2014, Portland, UR, USA, pp. 1 – 8.

Schubert, Sigrid und Schwill, Andreas (2011): Didaktik der Informatik. 2. Auflage, Heidelberg: Spektrum Akademischer Verlag.

Schwill Andreas, (1993), Fundamentale Ideen der Informatik. http://www.informatikdidaktik.de/Forschung/Schriften/ZDM.pdf [Zugriff: 10.03.2017]

Stajano, Frank (2002): Python in Education: Raising a Generation of Native Speakers, Proceedings of 8th International Python Conference, https://www.cl.cam.ac.uk/~fms27/papers/python-native-speakers.html [Zugriff: 14.03.2020]

Williams, Laurie; Wiebe, Eric; Yang, Kai et al (2002): In Support of Pair Programming in the Introductory Computer Science Course, Computer Science Education, Vol. 12, Nr. 3, pp. 197 – 212.

Wolber, David (2011): App-Inventor and real-world motivation, Proceedings of SIGCSE Technical Symposium on Computer Science Education, Dallas, TX, USA, pp. 601 – 606.

Zarb, Mark und Hughes, Janet (2015): Breaking the communication barrier: guidelines to aid communication within pair programming, Computer Science Education, Vol. 25, Nr. 2, pp. 120 – 151.

Zhong, Baichang; Wang, Qiyun; Chen, Jie (2016): The impact of social factors on pair programming in a primary school, Computers in Human Behavior, Vol. 64., Nr. 1, pp. 423 – 431.

Literatur als Bestandteil der SLR

Effektivität und Effizienz

S1: Isong, Bassey (2014): A Methodology for Teaching Computer Programming: first year students' perspective, I.J. Modern Education and Computer Science, Vol. 9, Nr. 4, pp. 15 – 21.

S2: Müller, Matthias und Padberg, Frank (2004): An Empirical Study about the Feelgood Factor in Pair Programming, Proc. 10th Int. Symp. On Software Metrics, IEEE Computing Soc., pp. 151 – 158.

S3: Iskrenovic-Momcilovic, Olivera (2019): Pair programming with Scratch, Education and Information Technologies, Vol. 24, pp. 2943 – 2952.

S4: Isong, Bassey; Moemi, Thuso; Dladlu, Nosipho et al (2016): Empirical Confirmation of Pair Programming Effectiveness in the Teaching of Computer Programming, 2016 International Conference on Computational Science and Computational Intelligence, pp. 276 – 282.

S5: Williams, Laurie; Kessler, Robert Cunningham, Ward et al (2000) Strengthening the Case for Pair Programming, IEEE Software, Vol. 17, 19 – 25.

S6: Faja, Silvana (2014): Evaluating Effectiveness of Pair Programming as a Teaching Tool in Programming Courses, Information Systems Education Journal (ISEDJ), Vol. 12, Nr. 6, pp. 36 – 45.

S7: Lewis, Colleen M. (2011): Is pair programming more effective than other forms of collaboration for young students?, Computer Science Education, Vol. 21, Nr. 2, pp. 105 – 134.

S8: He, Xinran und Chen, Yuwei (2014): Analyzing the Efficiency of Pair Programming in Education, Bachelor of Science, University of Gothenburg.

S9: Estácio, Bernardo; Valentim, Natasha; Rivero, Luis et al (2015): Evaluating the Use of Pair Programming and Coding Dojo in Teaching Mockups Development: An Empirical Study, 48th Hawaii International Conference on System Sciences, 5th - 8th January 2015.

S10: Williams, Laurie; Wiebe, Eric; Yang, Kai et al (2002): In Support of Pair Programming in the Introductory Computer Science Course, Computer Science Education, Vol. 12, Nr. 3, pp. 197 – 212.

S11: McChesney, Ian (2016): Three Years of Student Pair Programming: Action Research Insights and Outcomes, SIGCSE 16', 84 – 89.

S12: Toll, Theodore; Lee, Roger; Ahlswede, Thomas (2007): Evaluating the Usefulness of Pair Programming in a Classroom Setting, 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS), pp. 302 – 308.

S13: Nawrocki, Jerzy R., Jasinski, Michal, Olek, Lukas et al (2005): Pair Programming vs. Side-by-side Programming, 12th European Conf. on Software Process Improvement, EuroSPI 2005, pp. 28 – 38.

Motivation

S14: Bishop-Clark, Cathy; Courte, Jill; Howard, Elizabeth V. (2006): Programming in Pairs with Alice to Improve Confidence, Enjoyment, and Achievement, Journal of Educational Computing Research, Vol. 34, Nr. 2, pp. 213 – 228.

S15: Aarne, Onni; Peltola, Petrus; Leinonen, Juho et al (2018): A Study of Pair Programming Enjoyment and Attendance using Study Motivation and Strategy Metrics, SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education, February 2018, pp. 759 – 764.

S16: Cockburn, Alistair und Williams, Laurie (2000): The costs and benefits of pair programming, Extreme programming examined, pp. 223 – 247.

S17: Nagappan, Nachiappan; Williams, Laurie; Ferzli, Miriam et al (2003): Improving the CS1 experience with pair programming, ACM Sigcse Bulletin, Vol. 35, pp. 359 – 362.

S18: Liebenberg, Janet; Mentz, Elsa; Breed, Betty (2012): Pair programming and secondary school girls' enjoyment of programming and the subject Information Technology (IT), Computer Science Education, Vol. 22, Nr. 3, pp. 219 – 236.

S19: Braught, Grant; Wahls, Tim; Eby, L. Marlin (2011): The Case for Pair Programming in the Computer Science Classroom, ACM Transactions on Computing Education, Vol. 11, Nr. 1, Article 2.

Lernen und Lehren

S20: Gold-Veerkamp, Carolin; Klopp, Marco und Abke, Jörg (2019): Pair Programming as a Didactical Approach in Higher Education and its Evaluation, IEEE Global Engineering Education Conference (EDUCON), pp. 1055 – 1062.

S21: Kavitha, R. und Irfan, Mohammed Salman (2013): Knowledge sharing through pair programming in learning environments: An empirical study, Education and Information Technologies, Vol. 20, pp. 319 – 333.

S22: Swamidurai, Rajendran und Umphress, David (2014): The Impact of Static and Dynamic Pairs on Pair Programming. Proceedings - 8th International Conference on Software Security and Reliability - Companion, SERE-C 2014, pp. 57 – 63.

S23: Mentz, Elsa; Van der Walt, Johannes; Goosen, Leila (2008): The effect of incorporating cooperative learning principles in pair programming for student teachers, Computer Science Education, Vol. 18, Nr. 4, pp. 247 – 260.

S24: Reddy, Patil Deepti; Mishra, Shitanshu; Ramakrishnan, Ganesh et al (2015): Thinking, Pairing, and Sharing to Improve Learning and Engagement in a Data Structures and Algorithms (DSA) Class, 2015 International Conference on Learning and Teaching in Computing and Engineering, pp. 144 – 151.

S25: Lukosch, Stephan (2009): Understanding Tools and Practices for Distributed Pair Programming, Journal of Universal Computer Science, Vol. 15, Nr. 16, pp. 3101 – 3125.

S26: McKinsey, Jonathan (2015): Remote Pair Programming in a Visual Programming Language, Technical Report, University of California at Berkeley https://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-139.pdf [Zugriff: 23.4.2020]

S27: Fatourou, Eleni; Zygouris, Nikos; Loukopoulos, Thanasis et al (2020): More Than Structured Programming in Primary School Syllabus, International Conference on Interactive Collaborative Learning (ICL) 2019, pp. 212 – 221.

S28: Fetaji, Majlinda; Fetaji, Bekim; Ebibi, Mirlinda (2019): Analyses of possibilities of Flipped Classroom in Teaching Computer Science Courses, MIPRO 2019, pp. 747 – 752.

S29: Neha, Katira; Williams, Laurie; Osborne, Jason (2005): Towards increasing the compatibility of student pair programmers, ICSE '05, pp. 625 – 626.

Soziale Faktoren

S30: Maguire, Phil; Maguire; Rebecca; Hyland, Philip et al (2014): Enhancing collaborative learning using pair programming: Who benefits?, All Ireland Journal of Teaching and Learning in Higher Education, Vol. 6, Nr. 2, pp. 1411 – 14125.

S31: Zarb, Mark und Hughes, Janet (2015): Breaking the communication barrier: guidelines to aid communication within pair programming, Computer Science Education, Vol. 2, Nr. 2, pp. 120 – 151.

S32: Çal, Habibe; Can, Gülfidan (2020): The influence of pair programming on secondary school students' confidence and achievement in computer programming. Trakya Journal of Education, Vol. 10, Nr. 1, pp. 221 – 237.

S33: Plonka, Laura; Segal, Judith; Sharp, Helen; van der Linden, Janet (2012): Investigating Equity of Participation in Pair Programming, Agile India 2012, pp. 21 – 29.

S34: Man Lui, Kim; Chan, Keith C.C. (2006): Pair programming productivity: Novice–novice vs. expert–expert, Int. J. Human-Computer Studies, Vol. 64, pp. 915 – 925.

S35: Campe, Shannon; Denner, Jill; Green, Emily; Torres, David (2019): Pair programming in middle school: variations in interactions and behaviors, Computer Science Education, pp. 1 – 25.

S36: Tsan, Jennifer; Vandenberg, Jessica; Zakaria, Zarifa et al (2020): A Comparison of Two Pair Programming Configurations for Upper Elementary Students, The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, pp. 346 – 352.

S37: DeClue, Timothy (2003): Pair programming and pair trading: Effects on learning and motivation in a CS2 course, Journal of Computing Sciences in Colleges – JCSC, Vol.18, Nr. 5, pp. 49 – 56.

S38: Zhong, Baichang; Wang, Qiyun; Chen, Jie (2016): The impact of social factors on pair programming in a primary school, Computers in Human Behavior, Vol. 64, pp. 423 – 431.

S39: Rodríguez, Fernando; Price, Kimberly; Boyer, Kristy (2017): Exploring the Pair Programming Process: Characteristics of Effective Collaboration, ACM SIGCSE Technical Symposium, pp. 507 – 512.

S40: Salleh, Norsaremah; Mendes, Emilia; Grundy, John; Burch, Giles (2010): An empirical study of the effects of personality in pair programming using the five-factor model, International Conference on Software Engineering, pp. 577 – 586.

S41: Faja, Silvana (2011): Pair Programming as a Team Based Learning Activity, Issues in Information Systems, Vol. 12, Nr.2, pp. 207 – 216.

ABBILDUNGSVERZEICHNIS

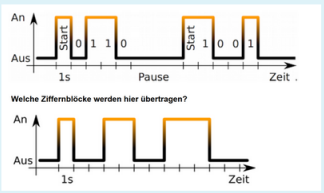
ANHANG A – ALGORITHMISCHES VERSTÄNDNIS

Aufgabe 1

Alice und Bob möchten nachts mit ihren Taschenlampen Nachrichten übertragen.
Sie senden sich Blöcke von vier Ziffern. Die Ziffern sind 0 oder 1.

Zum Start eines Ziffernblocks schalten sie die Taschenlampe für eine Sekunde ein.
Danach kommen die vier Ziffern im Sekundentakt.
Taschenlampe an bedeutet 1, Taschenlampe aus bedeutet 0.
Bis zum nächsten Block kommt dann eine Pause von mindestens einer Sekunde,
mit Taschenlampe aus.

Das Beispiel zeigt die Übertragung der Ziffernblöcke **0110** und **1001**:



Welche Ziffernblöcke werden hier übertragen?

Wählen Sie eine Antwort:

- a. Die Ziffernblöcke 0011 und 1100
- b. Die Ziffernblöcke 0011 und 1110
- c. Die Ziffernblöcke 1100 und 0011
- d. Nur der Ziffernblock 0101

Quelle: https://www.ocg.at/de/biber-der-informatik [Zugriff: 12.03.2020]

Aufgabe 2

Bei einem Schachturnier treffen Teams aufeinander.
Jedes Team hat sechs Spieler; drei davon spielen mit weißen Figuren,
die drei anderen spielen mit schwarzen Figuren.

Über die Begegnung der beiden besten Teams ist Folgendes zu erfahren:

- Die Spieler A, B, C, D, E und F spielen mit weißen Figuren,
  die Spieler G, H, I, J, K und L spielen mit schwarzen Figuren.
- Spieler A spielt gegen Spieler H, K spielt gegen E, C gegen I und F gegen G.
- Folgende Spielerpaare sind jeweils aus dem gleichen Team:
  B und C, I und J, H und B, C und G.
- Die Spieler L und G sind aus verschiedenen Teams.

A	B	C	D	E	F	G	H	I	J	K	L

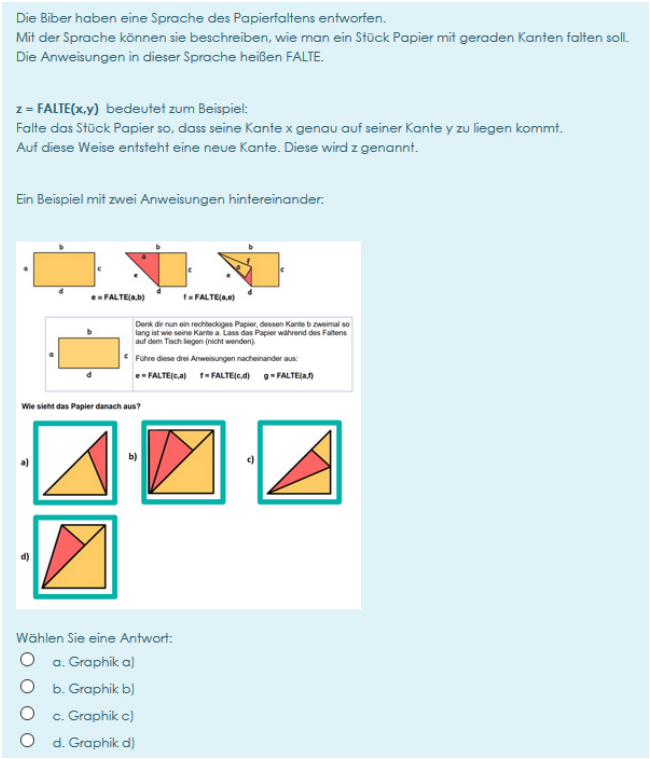
In der Grafik sollen die Teamkameraden von Spieler A rot
und die Spieler des gegnerischen Teams grün gefärbt werden.

Wählen Sie eine Antwort:

- a. Graphik a)
- b. Graphik b)
- c. Graphik c)
- d. Graphik d)

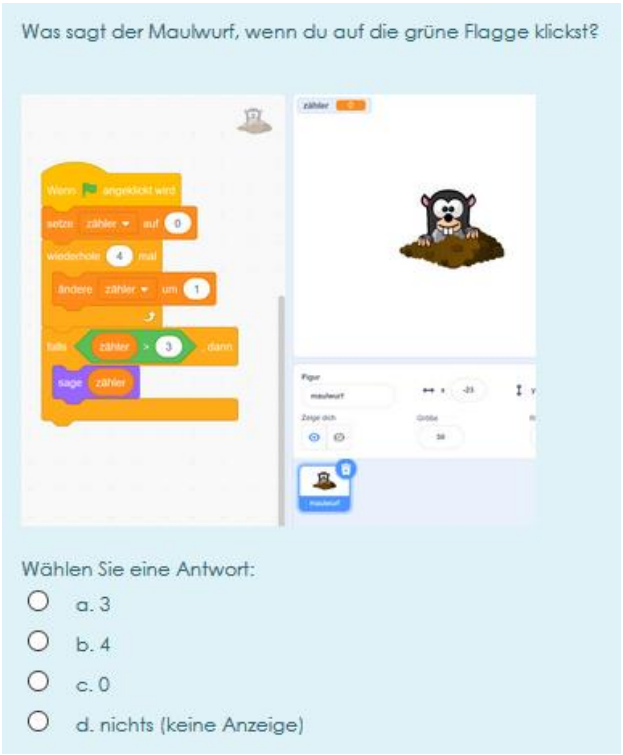
Quelle: https://www.ocg.at/de/biber-der-informatik [Zugriff: 12.03.2020]

Aufgabe 3



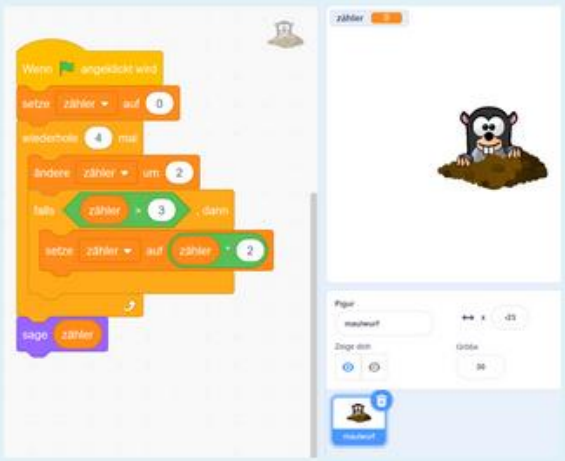
Quelle: https://www.ocg.at/de/biber-der-informatik [Zugriff: 12.03.2020]

Aufgabe 4



Aufgabe 5

Was sagt der Maulwurf, wenn du auf die grüne Flagge klickst?

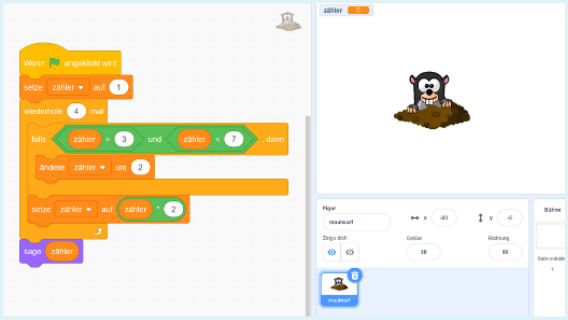


Wählen Sie eine Antwort:

- a. 22
- b. 42
- c. 38
- d. 44

Aufgabe 6

Was sagt der Maulwurf, wenn du auf die grüne Flagge klickst?



Wählen Sie eine Antwort:

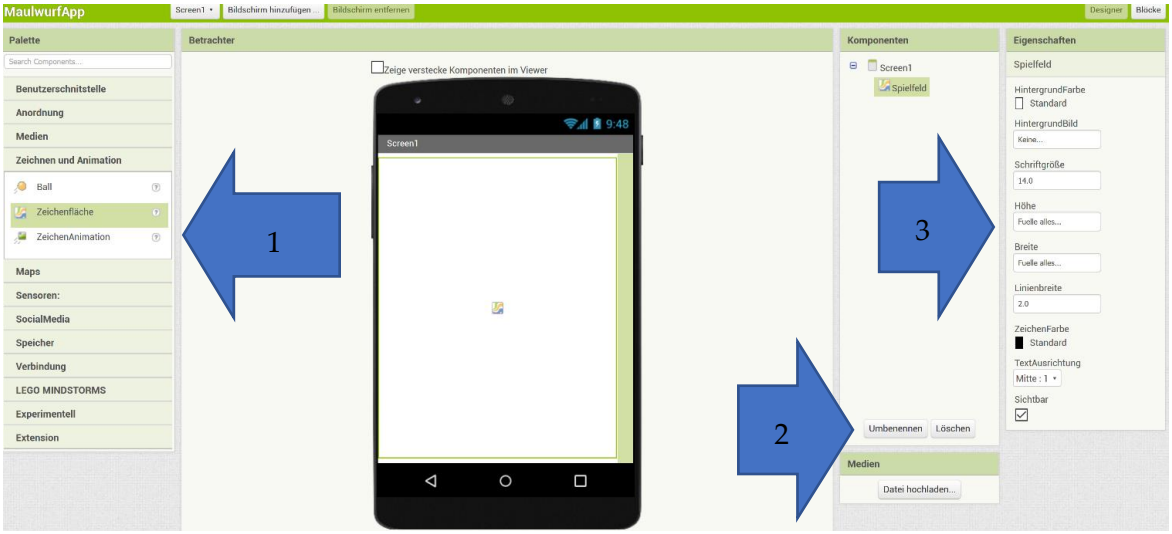
- a. 18
- b. 20
- c. 22
- d. 24

ANHANG B – FACHDIDAKTISCHE KONZEPTE

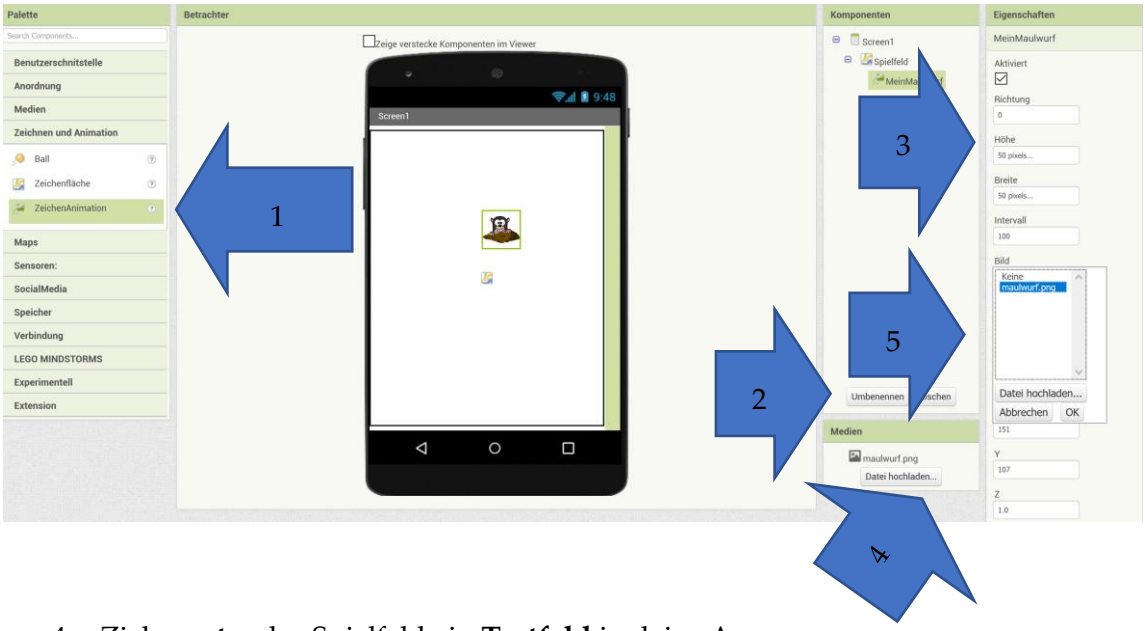
MIT App Inventor: Pair Programming (Maulwurf-Spiel)

Design

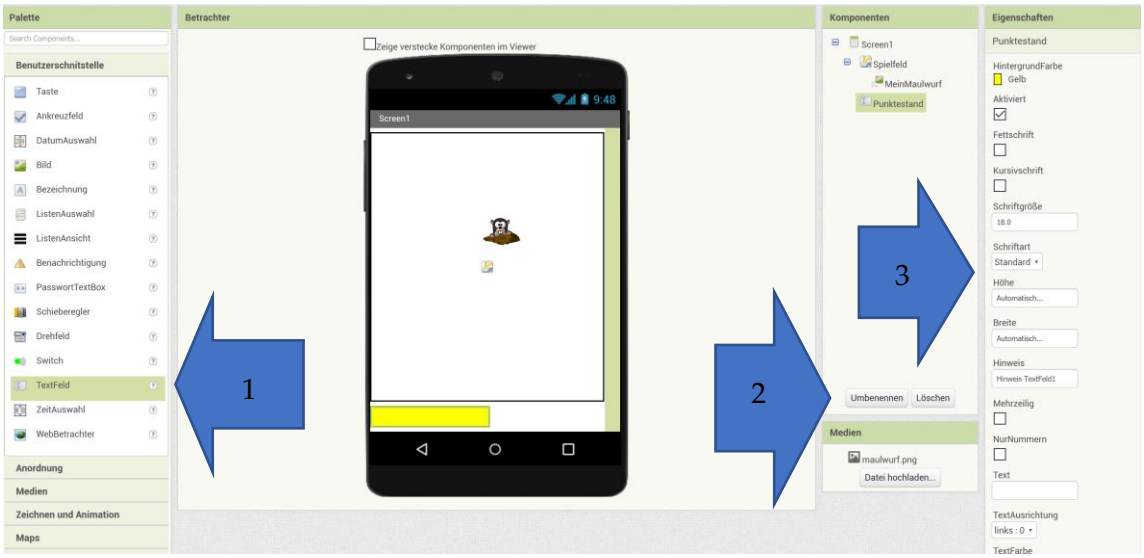
1. Lege ein neues Projekt an. Projektname: **MaulwurfApp.**
2. Ziehe eine neue **Zeichenfläche** in deine App.
    a. Benenne die Zeichenfläche in **„Spielfeld“** um.
    b. Ändere die Höhe und Breite auf **„fuelle pixels“**.



3. Ziehe eine neue **ZeichenAnimation (Sprite)** in deine App.
    a. Benenne die ZeichenAnimation in **„MeinMaulwurf“** um.
    b. Ändere die Höhe und Breite auf je **50 pixel.**
    c. Lade unter „Medien“ das Bild **maulwurf.png** hoch und weise deiner Zeichenanimation in den Eigenschaften dieses Bild zu.

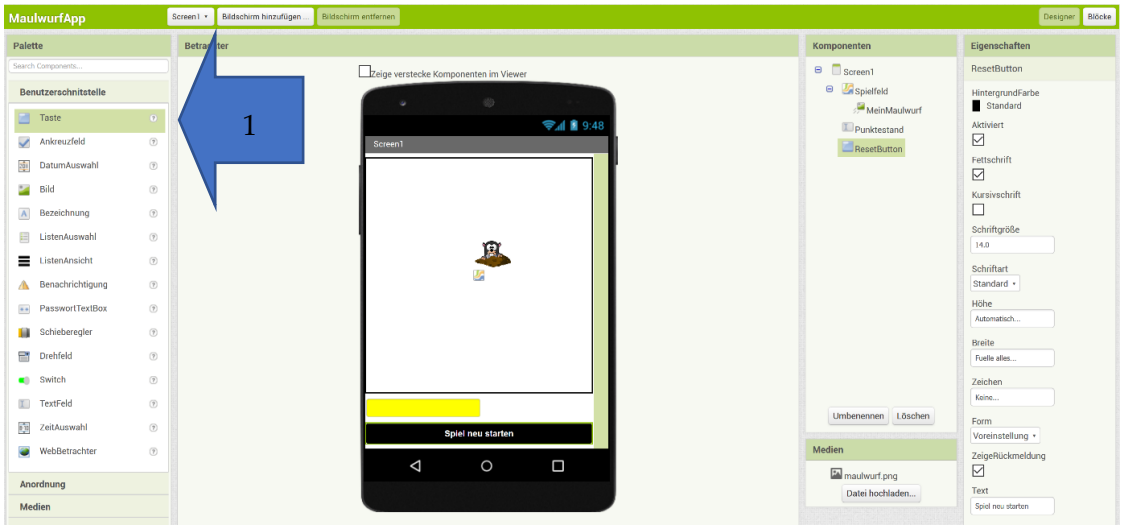


4. Ziehe **unter** das Spielfeld ein **Textfeld** in deine App.
    a. Benenne es in „Punktestand“ um.
    b. Ändere die **Hintergrundfarbe** beliebig und die **Schriftgröße auf 18**.



5. Ziehe eine **Taste (Button)** ganz unten in deine App.

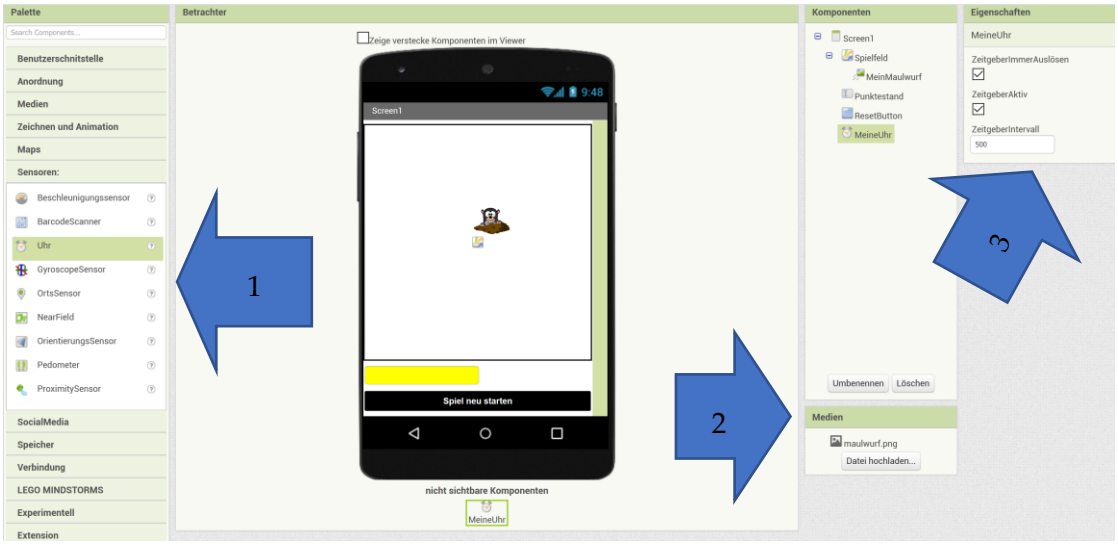
    a. Ändere die Eigenschaften so, dass das Ergebnis gleich aussieht wie im Tutorial.



6. Für die Bewegung des Maulwurfs brauchen wir noch eine Uhr. Ziehe die Komponente **„Uhr“** (Clock) in deine App, sie erscheint als **nicht sichtbare Komponente**.

    a. Ändere den Namen auf „Meine Uhr“.

    b. Ändere das **Zeitintervall** auf 0,5 Sekunden (500 ms).



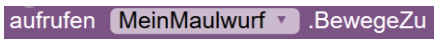
Super, du hast das Design für die App fertig! Jetzt wird es Zeit für die Programmierung …

Programmierung

1. Sorge dafür, dass sich dein Maulwurf zufällig bewegt!

a. Nutze den 

Block. Dieser wird (durch die Uhr) momentan alle 0,5 Sekunden ausgelöst.

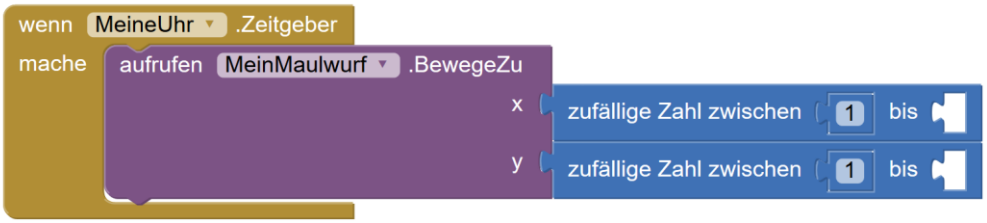
b. Rufe darin die Prozedur 

auf. Hier kannst du den Maulwurf entlang der x- und y- Koordinate zu einem bestimmten Punkt bewegen.

c. Wähle als **x- und y- Koordinate** einen zufälligen Punkt innerhalb des Spielfeldes: Fülle die Vorlage unten so aus, dass zufällige Zahlen zwischen

   i. „1“ und der Spielfeldbreite (für x) und

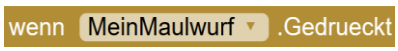
   ii. „1“ und der Spielfeldhöhe (für y) gewählt werden



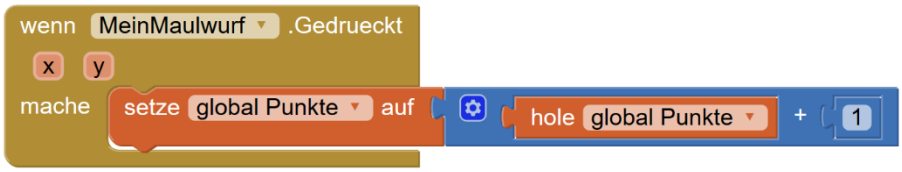
Verbessere das Programm so, dass der Maulwurf nicht zu weit am Rand auftaucht!

2. Erhöhe den **Punktestand**, wenn du den Maulwurf triffst!

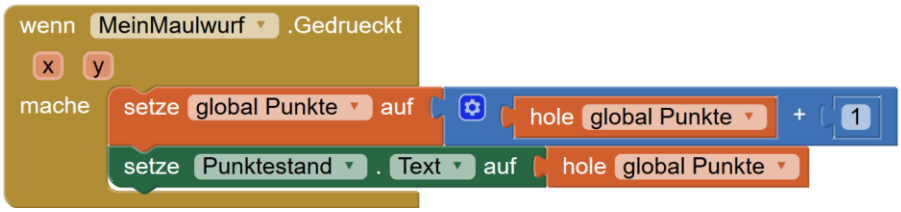
a. Erzeuge eine Variable mit dem Namen Punkte und setze sie auf „0“.

b. Verwende den 

Block, um die Variable immer zu erhöhen, wenn der Maulwurf berührt wird:



c. Setze nach jedem neuen Punkt den Text deines Textfeldes **„Punktestand“** auf den Wert deiner Variable:

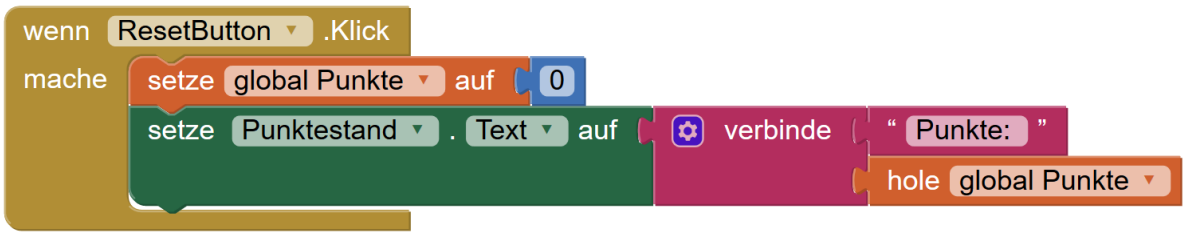


Verbessere das Programm so, dass vor der Punktezahl der Text „Punkte:“ erhalten bleibt!

3. Setze die Punkte beim Drücken des **„Reset-Buttons“** auf „0“!

a. Verwende den 

Block, um die Punkte auf „0“ zu setzen.



Super, dein Maulwurfspiel sollte funktionieren!

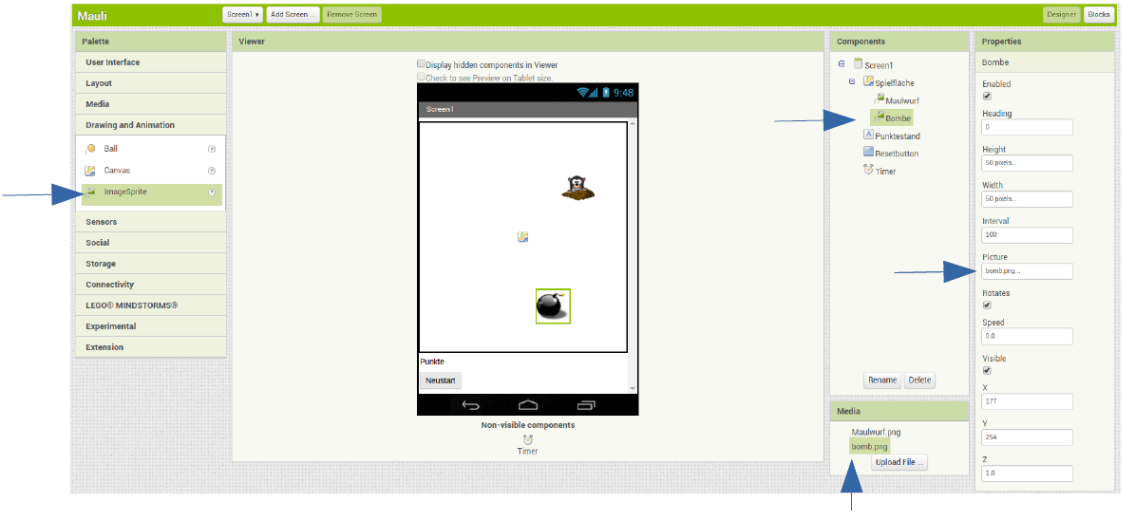
Jetzt wird es Zeit für einige Challenges…

Challenge 1

Programmiere eine Bombe!
Wenn du die Bombe klickst, werden die Punkte auf **“0”** gesetzt.

Du brauchst:

- Ein Bild der Bombe (Google – Nutzungsrechte beachten!)
- Einen **“ImageSprite”** unter “Drawing and Animation”
- Das Bild der Bombe musst du dem ImageSprite zuweisen (siehe Abbildung)

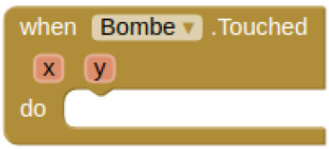


Ändere dein Programm so, dass sich die Bombe (so wie der Maulwurf) jede Sekunde bewegt.

Tipp: Verwende deine Prozedur “bewegung”!

Dann programmiere die Bombe so, dass die Punkte auf 0 gesetzt werden, wenn du die Bombe berührst.

Tipp: In diesen Block schreibst du deine Befehle!



Challenge 2

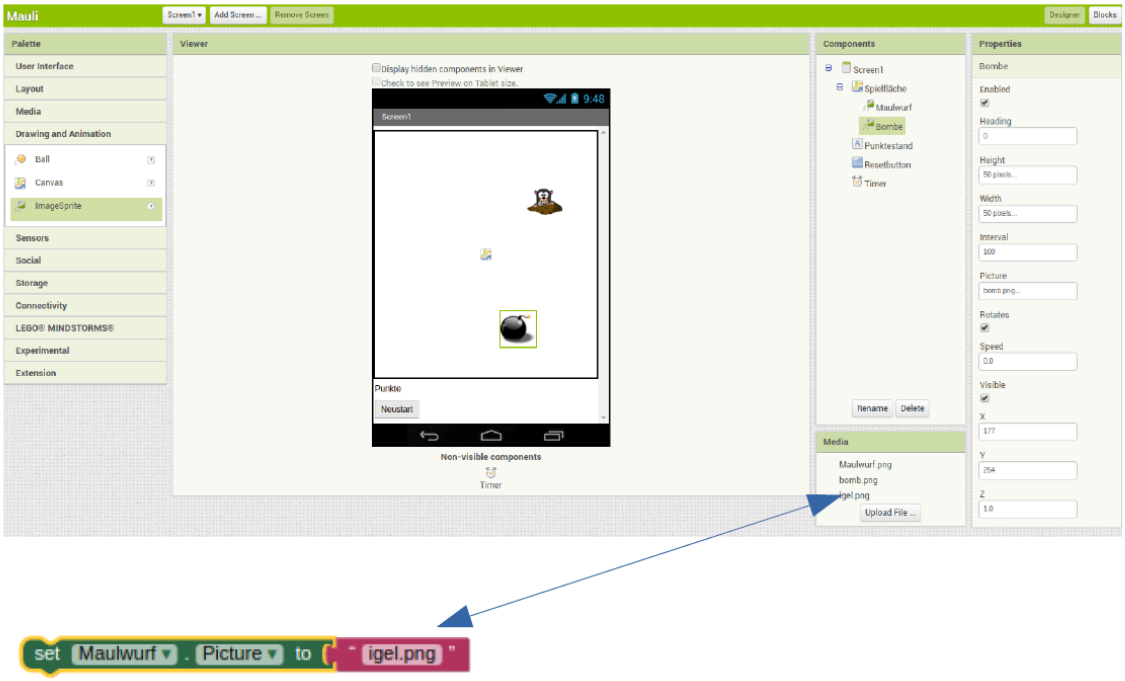
Ändere die Spielfigur (ImageSprite)

Sobald du mehr als 2 Punkte hast, soll sich die Spielfigur ändern. Statt des Maulwurfs taucht also z.B. ein Igel auf, den du fangen musst.

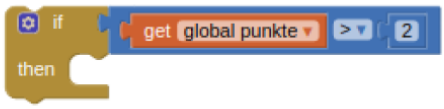
Du brauchst:

- Ein Bild einer neuen Spielfigur (Google – Nutzungsrechte beachten!)

Tipp: Mit dem Block **"set Maulwurf.Picture to"** kannst du das Bild ändern. Der Name danach muss genau der Name deines Bildes, inklusive Dateiendung, sein.



Tipp: Um nachzuschauen, ob du bereits mehr als 2 Punkte hast, kannst du eine **"if-Schleife"** verwenden. Wenn die Bedingung neben dem "if" (Englisch für "wenn") zutrifft, dann wird der Teil neben dem "then" ausgeführt.



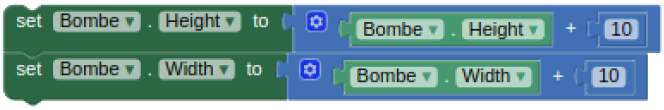
Was soll passieren, wenn die Bedingung **"get global punkte"** ist größer als **"2"** zutrifft?

Challenge 3

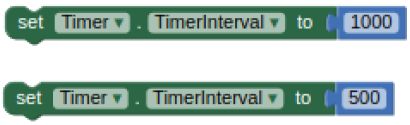
Überlege dir neue Spielfunktionen und programmiere sie!

Zwei Vorschläge:

- Die Bombe wird mit jedem Punkt größer
  Tipp: So kannst du die Höhe und Breite der Bombe um 10 Pixel vergrößern!

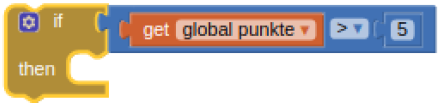


- Bombe und Maulwurf sollen sich nach 5 Punkten schneller bewegen
  Tipp: So kannst du die Zeit deiner Stoppuhr ändern!



Im blauen Kästchen stehen die Millisekunden, nach denen sich der Maulwurf und die Bombe bewegen – also 1000ms (1 Sekunde) oben und 500ms (0,5 Sekunden unten.

Tipp: Wie du abfragst, ob schon mehr als 5 Punkte erreicht hast, weißt du schon!



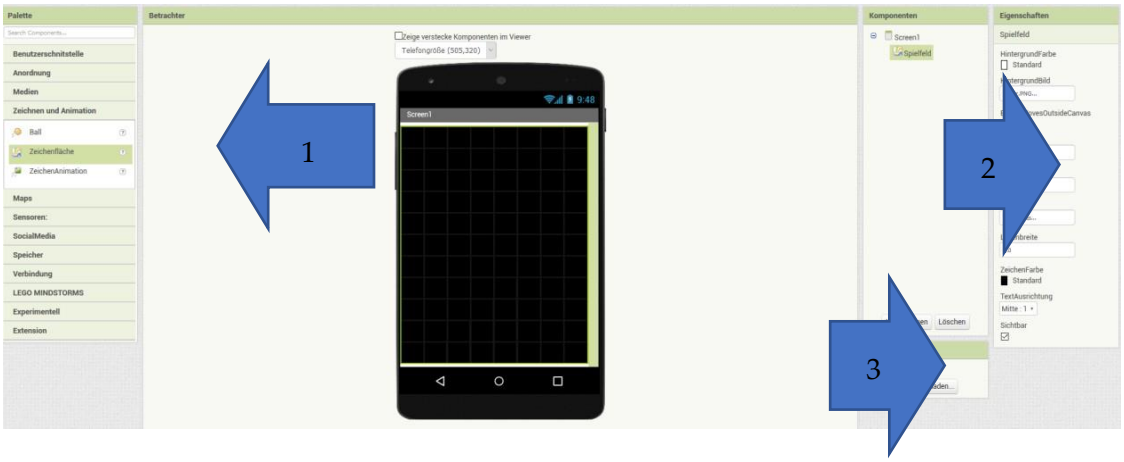
Weitere Challenges:

- ✓ **Bei 15 Punkten taucht ein Banner mit „Gewonnen!“ auf und es erklingt ein Lied.**
- ✓ **Alle 15 Sekunden wird die Bewegungsgeschwindigkeit des Maulwurfs erhöht.**
- ✓ **Die Anzahl der Bomben verdoppelt sich alle 15 Sekunden.**
- ✓ **Per Sprachausgabe („Text-To-Speech“) wird der Spieler / die Spielerin alle 15 Sekunden motiviert.**

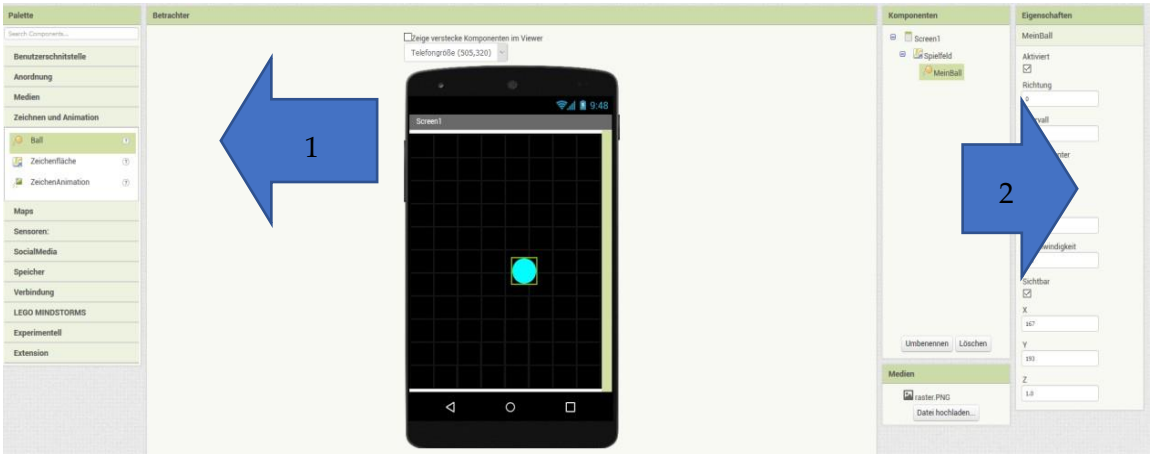
MIT App Inventor: Solo Programming (Pong Spiel)

Design

1. Lege ein neues Projekt an. Projektname: **PongSpiel.**
2. Ziehe eine neue **Zeichenfläche** in deine App.
    a. Benenne die Zeichenfläche in **„Spielfeld“** um
    b. Ändere die Höhe und Breite auf „**fuelle alles**“
    c. Lade das Bild „raster.png“ hoch und weise es dem Spielfeld zu



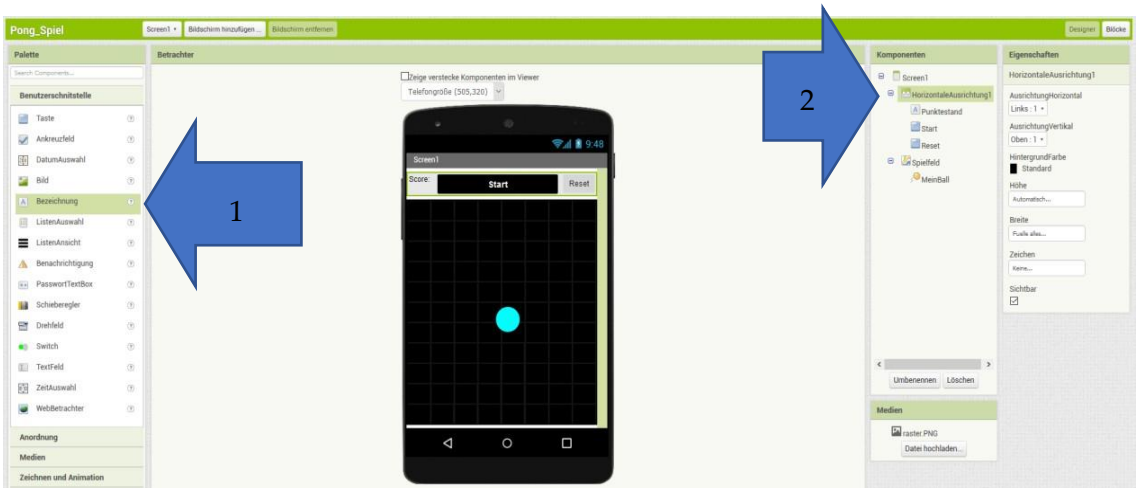
3. Ziehe einen neuen **Ball** in deine App.
    a. Benenne den Ball in **„MeinBall“** um
    b. Ändere **den Radius auf 20** und wähle eine **Farbe**



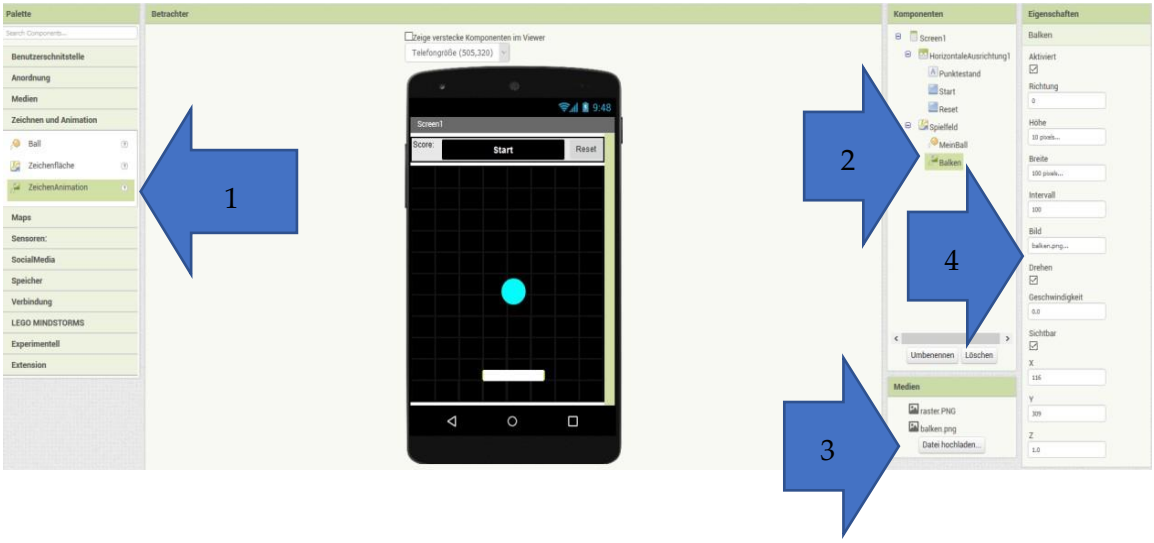
4. Ziehe **über** das Spielfeld unter **Anordnung** ein **horizontales Ausrichtungsfeld** in deine App. Damit können wir mehrere Elemente nebeneinander platzieren.

    Ziehe dann folgende Elemente in deine App und definiere sie wie im Screenshot:

    a. Ein **Bezeichnungsfeld** „Punktestand“ mit dem Text „Score:“
    b. Ein **Start-Button** mit dem Text „Start“
    c. Ein **Reset-Button** mit dem Text „Reset“



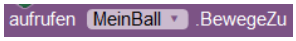
5. Ziehe eine neue Zeichenanimation **(Sprite)** in dein Spielfeld.
    a. Benenne die Zeichenfigur in „**Balken**“ um
    b. Lade das Bild „**balken.png**“ in deine App und weise es der Zeichenanimation zu
    c. Ändere die Eigenschaften so, dass Höhe und Breite passen



Sorge dafür, dass sich dein Ball bewegt!

a. Nutze den 

Block. Dieser wird ausgelöst, wenn du den Start-Button klickst.

b. Rufe darin die Prozedur 

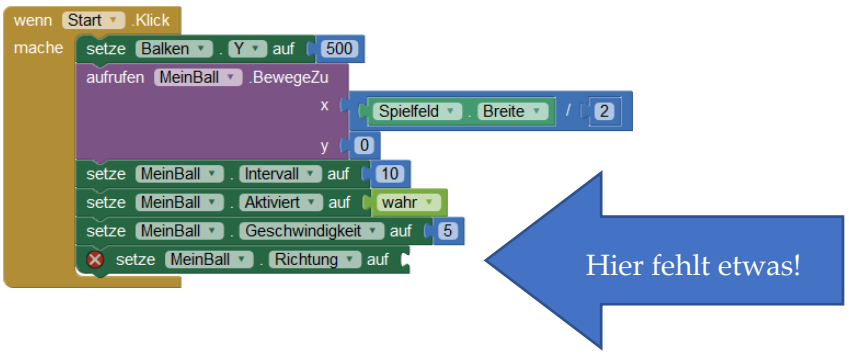
auf. Hier kannst du die Startposition des Balles definieren. Der Ball soll ganz oben in der Mitte starten.

c. Setze die Werte des Balles.

d. Probiere andere Werte für Intervall und Geschwindigkeit aus

e. Die Richtung des Balles wird durch die Winkelzahl (0 Grad bis 360 Grad) angegeben

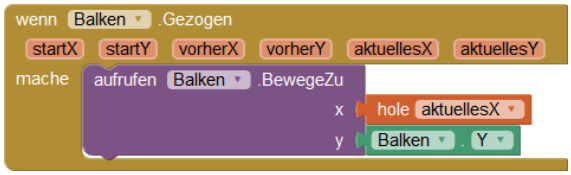
f. Setze die Richtung auf einen **Zufallswert**, so dass der Start-Winkel des Balles immer in zufälligem Winkel **nach unten** zeigt



Sorge dafür, dass du den Balken bewegen kannst!

a. Verwende den folgenden Block:

- Die Y-Position bleibt immer gleich
- Die X-Position verändert sich je nachdem, wo du den Balken hinziehst.



Lass den Ball abprallen!

a. Verwende den folgenden Block:

- Ändere die Richtung so, dass der Ball realistisch abprallt

Definiere die Spielfeld-Ränder

a. Die Ränder werden mit Zahlen angesprochen:

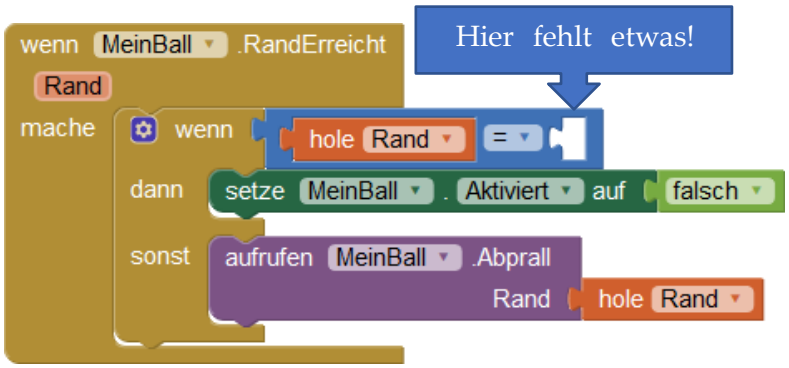
Oben = 1, Unten = -1, Rechts = 3, Links = -3

b. Das Spiel wird verloren, wenn der untere Rand berührt wird

c. Ansonsten soll der Ball abprallen

d. Verwende den folgenden Block:

- Setze die passende Zahl ein!

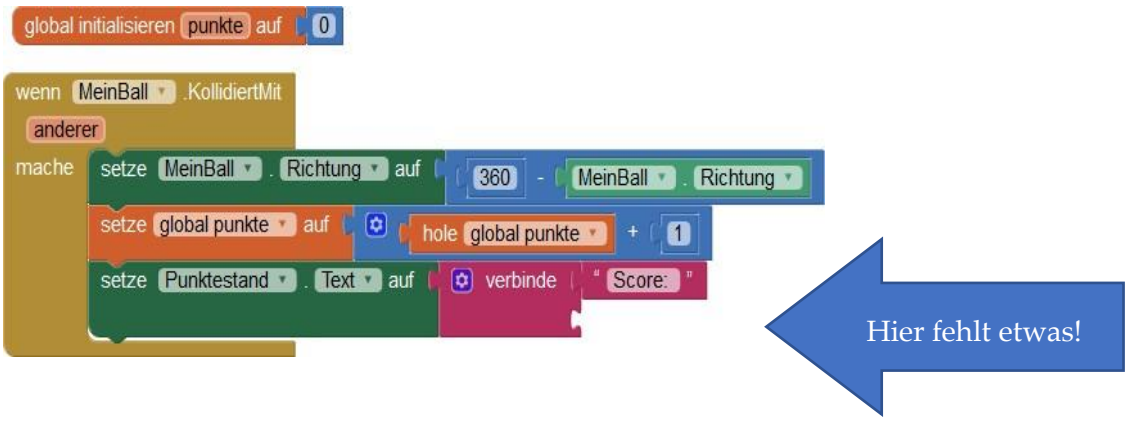


Zähle die Punkte im „Score“

a. Definiere eine Variable „**punkte**“ und setze ihren Wert auf 0

b. Erhöhe die Punkte um 1, wenn der Balken berührt wird

c. Vergiss nicht, die Punkte jeweils im Textfeld „**Punktestand**“ anzuzeigen



Achtung: Setze die Punkte auf 0, wenn der untere Rand berührt wird!

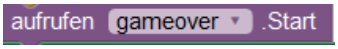
Setze dein Spiel mit „Reset“ zurück

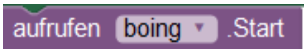
a. Setze alle Werte so zurück, dass das Spiel neu begonnen werden kann

Füge Sounds hinzu

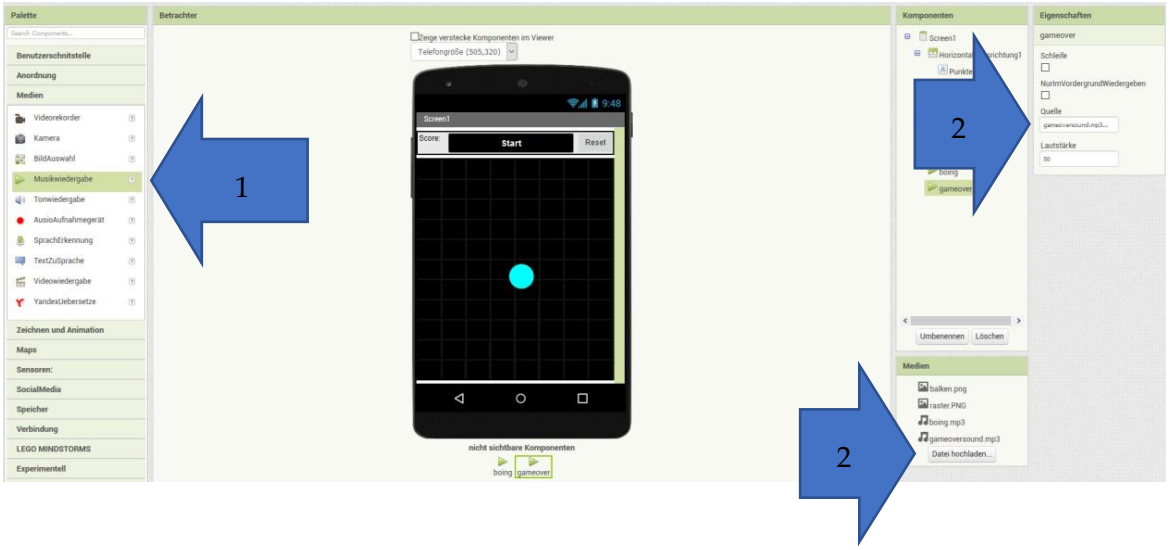
a. Ziehe im „Designer“ zwei Musikwiedergaben in deine App und benenne sie in „boing“ und „gameover“ um

b. Lade unter „Medien“ die Sounds **boing.mp3** und **gameover.mp3** hoch und weise sie deinen Musikwiedergaben zu

c. Verwende die Blöcke 

sowie 

an die passende Stelle, um die Sounds abzuspielen



Super, dein Pong-Spiel sollte funktionieren!

Jetzt wird es Zeit für einige Challenges…

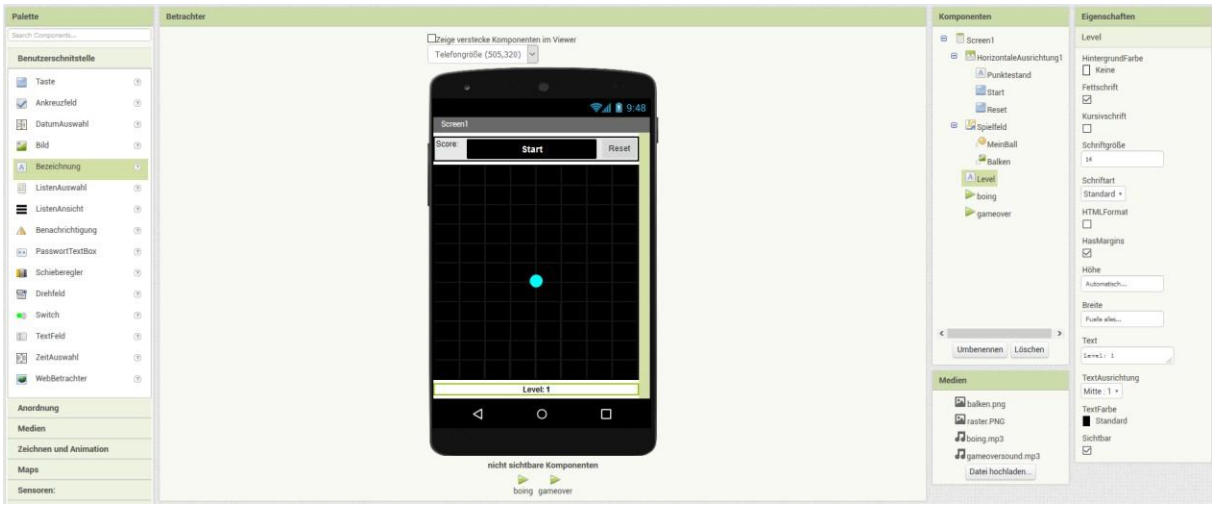
Challenge 1

Programmiere Levels!

Es sollen 5 Levels mit jeweils steigender Schwierigkeit (Geschwindigkeit des Balles) programmiert werden. Die Levels werden in einem Textfeld ganz unten angezeigt.

Du brauchst:

- Ein Textfeld („Bezeichnung“)



Ändere dein Programm so, dass die Levels angezeigt und die Geschwindigkeit jeweils erhöht wird.

- Level 1: 0 bis 4 Punkte
- Level 2: 5 bis 9 Punkte
- Level 3: 10 bis 14 Punkte
- Level 4: 15 bis 19 Punkte
- Level 5: Ab 20 Punkte

Erhöhe nun in jedem Level die **Geschwindigkeit**!

Tipp: Verwende den Block 

Challenge 2

Ändere den Hintergrund

Der Hintergrund soll sich alle 5 Punkte ändern!

- Level 1: weiß
- Level 2: grün
- Level 3: rot
- Level 4: blau
- Level 5: pink

Achte darauf, dass bei „Reset“ und „Start“ die Hintergrundfarbe wieder auf weiß gesetzt wird!

Tipp: Verwende den Block:

Challenge 3

Programmiere ein Hindernis!

Das Hindernis soll alle 3 Sekunden zufällig die Position wechseln. Wenn der Ball auf dein Hindernis trifft, gelten für 1 Sekunde folgende Regeln:

- Der Ball ändert ständig zufällig die Richtung
- Die Geschwindigkeit wird auf 20 gesetzt
- Der Ball wird immer größer

Nach 1 Sekunde soll alles wieder „normal“ laufen.

Du brauchst:

- Ein Bild von einem Hindernis (Achtung: Urheberrecht)
- Einen „Image Sprite“(Zeichenanimation)
- Eine Uhr (setze das Intervall auf 100 Millisekunden)

Ändere dein Programm so, dass der Ball auch vom Hindernis abprallt.

Tipp: Verwende folgenden Block:

Verwende dann die Uhr als Zeitgeber, um für 2 Sekunden die „Regeln“ zu ändern. Du kannst so anfangen:

Weitere Challenges

- ✓ Bei 15 Punkten taucht ein Banner mit „Gewonnen!“ auf und es erklingt ein Sound.
- ✓ Per Sprachausgabe („Text-To-Speech“) wird der Spieler / die Spielerin alle 15 Sekunden motiviert.

Python: Solo Programming (Bibliothek)

Grundprogramm

1. **Programmiere eine while-Schleife mit Abbruchbedingung!**
    a. Programmiere die **while-Schleife**, in der gilt:

    So lange start == True:

    i. Wenn der User „x“ eingibt (If-Schleife), wird start auf *False* gesetzt: start = False

    ii. Dadurch wird das Programm beendet

    b. Das Ergebnis sollte so ausschauen:

```
[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 2
2
[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 1
1
[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! x
x
Programm beendet
>>>
```

2. **Füge ein Buch hinzu, wenn der User „1“ eingibt! (Teil 1)**
    a. Programmiere innerhalb der passenden if-Schleife
    b. Erhöhe book_nummer um 1: book_nummer = book_nummer + 1
    c. Erzeuge 4 Variablen **titel**, **isbn**, und **preis** und verwende **input()**, um sie auf den User-Input zu setzen

    Beispiel: titel = input(„Bitte Titel eingeben“)

    d. Erzeuge die Variable **status** und setze sie auf „verfügbar“:

    status = „verfügbar“

    e. Gib alle Variablen aus:

    print(book_nummer, titel, isbn, preis, status)

    f. Das Ergebnis sollte so ausschauen:

```
[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 1
Bitte Titel eingebenTestbuch
Bitte ISBN eingeben123
Bitte Preis eingeben12
1 Testbuch 123 12 verfügbar
```

3. **Füge ein Buch hinzu, wenn der User „1“ eingibt! (Teil 2)**
    a. Programmiere innerhalb der passenden if-Schleife
    b. Erzeuge eine Liste newbook ganz am **Ende der Schleife**
    c. Setze als Elemente der Liste alle erzeugten Variable in der **richtigen Reihenfolge**

        newbook=[book_nummer, titel, isbn, preis, status]

    d. Füge newbook zur Liste books hinzu: books.append(newbook)
    e. Das Ergebnis sollte so ausschauen:

```
[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 1
Bitte Titel eingebenTest
Bitte ISBN eingeben123
Bitte Preis eingeben12


[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 3
[0, 'Herr d. Ringe', 123456, 33.23, 'entliehen']
[1, 'Test', '123', '12', 'verfügbar']
```

4. **Implementiere einen Passwort-Schutz!**
    a. Erzeuge eine Variable **passwort** VOR der while-Schleife
    b. Verwende **input()**, damit der User das Passwort eingeben kann
    c. Solange das Passwort nicht gleich „**mama**“ ist, kommt der User nicht zum „Hauptprogramm“
    d. Das Ergebnis sollte so ausschauen:

```
Willkommen zur Bibliothek! Bitte Passwort hinzufügen!papa
Falsches Passwort - neuer Versuch!oma
Falsches Passwort - neuer Versuch!mama


[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch!
```

Challenges

Tipp: Erstelle ein **„Backup“** deiner funktionierenden .py Datei, bevor du die Challenges beginnst!

Challenge 1

Implementiere eine ISBN-Suchfunktion für den Fall, dass der User „2“ eingibt!

a. Erzeuge eine Variable **suche_isbn** und setzte diese auf **input()**, damit der User die gesuchte ISBN-Nummer eingeben kann

b. Erzeuge eine for-Schleife von 0 bis zur Länge der Liste „books“: for i in range (0, len(books))

c. Verwende eine if-Bedingung, um zu überprüfen, ob suche_isbn **Teil der Liste** an der Stelle „i“ ist:

   If suche_isbn in books[i]:

d. Gib in diesem Fall das Element „i“ der Liste „books“ aus: print(books[i])

e. Das Ergebnis sollte so ausschauen:

```
[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 3
[0, 'Herr d. Ringe', 123456, 33.23, 'entliehen']
[1, 'Test', '123', '12', 'verfügbar']
[2, 'Superbuch', '12345', '32', 'verfügbar']


[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [x] für Abbruch! 2
ISBN eingeben12345
[2, 'Superbuch', '12345', '32', 'verfügbar']
```

Challenge 2

Brich das Programm ab, wenn der User 4 Mal das falsche Passwort eingibt!

a. Erzeuge eine Variable **anzahl** ganz am Beginn des Programms und setze sie auf 0

b. Arbeite nun in der while-Schleife: while(passwort != „mama“)

c. Erhöhe anzahl um 1, wenn der User das **falsche Passwort** eingibt und überprüfe in einer if-Schleife, ob **anzahl > 2** ist

   i. Wenn ja, setzte start auf False, um das Programm zu beenden: start = False

   ii. verwende den **break**-Befehl, um die while-Schleife zu verlassen

d. Das Ergebnis sollte so ausschauen:

```
Passwortpapa
Passwort  1  Mal falsch eingeben
Passwort noch einmal eingebenoma
Passwort  2  Mal falsch eingeben
Passwort noch einmal eingebenonkelhubert
Passwort  3  Mal falsch eingeben
Passwort noch einmal eingebenonkelfranz
Passwort zu oft falsch eingegeben!
Programm beendet
>>>
```

Challenge 3

Erstelle einen zufälligen Buch-Tipp, wenn der User „4“ eingibt!

a. Programmiere in der richtigen if-Schleife!

b. Importiere die Bibliothek „**random**“ am Beginn des Programms: import random

c. Erzeuge eine Zufallszahl zwischen **0** und dem **letzten Element der Liste books:**

   zz = random.randint(0,len(books) -1)

d. Erzeuge eine Variable „**buch**“ und setzte sie auf: buch = books[zz]

e. Gib das 1. Element (den **Titel**) von „**buch**“ aus

f. Das Ergebnis sollte so ausschauen:

```
[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [4] für einen Buchtipp und [x] für Abbruch! 3
[0, 'Herr d. Ringe', 123456, 33.23, 'entliehen']
[1, 'Carlos', '12354', '23', 'verfügbar']
[2, 'Beautiful', '43634', '21', 'verfügbar']


[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [4] für einen Buchtipp und [x] für Abbruch! 4
Du solltest unbedingt das Buch ===  Carlos  === lesen!


[1] Buch hinzufügen, [2] für ISBN Suche, [3] für Anzeige aller Bücher, [4] für einen Buchtipp und [x] für Abbruch! 4
Du solltest unbedingt das Buch ===  Beautiful  === lesen!
```

Python: Pair Programming (Kasino)

Grundprogramm

1. **Programmiere eine Endlos-Schleife!**
    a. In dieser Schleife soll der User zur **Eingabe einer Würfelzahl** aufgefordert werden
    b. Diese **User-Zahl** soll ausgegeben werden
    c. Das Ergebnis sollte so ausschauen:

```
Wuerfelzahl eingeben!2
Die User-Zahl ist 2
Wuerfelzahl eingeben!3
Die User-Zahl ist 3
Wuerfelzahl eingeben!4
Die User-Zahl ist 4
Wuerfelzahl eingeben!5
Die User-Zahl ist 5
```

2. **Erzeuge die Zufallszahl!**
    a. Importiere die *random*-Bibliothek am Anfang des Programmes: import random
    b. Speichere eine **Zufalls-Würfelzahl** in einer Variable
    c. Gib die Zufalls-Würfelzahl nach der User-Zahl aus
    d. Das Ergebnis sollte so ausschauen:

```
Wuerfelzahl eingeben!1
Die User-Zahl ist 1
Die Zufalls-Würfelzahl ist 5
Wuerfelzahl eingeben!2
Die User-Zahl ist 2
Die Zufalls-Würfelzahl ist 5
Wuerfelzahl eingeben!3
Die User-Zahl ist 3
Die Zufalls-Würfelzahl ist 6
```

3. **Überprüfe, ob der User gewinnt!**
    a. Lösche die Ausgabe der User-Zahl
    b. Erzeuge eine **If-Schleife**, in der überprüft wird, ob User-Zahl und Zufalls-Würfelzahl gleich sind
        i. Wenn ja: Gib „**GEWONNEN**“ aus
        ii. Wenn nein: Gib „**VERLOREN**“ aus
    c. Das Ergebnis sollte so ausschauen:

```
Wuerfelzahl eingeben!4
Die Zufalls-Würfelzahl ist 5
VERLOREN
Wuerfelzahl eingeben!4
Die Zufalls-Würfelzahl ist 4
GEWONNEN
Wuerfelzahl eingeben!4
Die Zufalls-Würfelzahl ist 3
VERLOREN
```

4. **Erzeuge den Kontostand!**
    a. Erzeuge **VOR** der Endlos-Schleife eine Variable „**konto**“ und setze sie auf 10
    b. Gib den **Kontostand** in jeder Würfelrunde aus
    c. Das Ergebnis sollte so ausschauen:

```
Dein Kontostand beträgt: 10
Wuerfelzahl eingeben!3
Die Zufalls-Würfelzahl ist 3
GEWONNEN
Dein Kontostand beträgt: 10
Wuerfelzahl eingeben!3
Die Zufalls-Würfelzahl ist 4
VERLOREN
```

5. **Berechne den Kontostand!**

a. Erhöhe konto um 2, wenn der Spieler gewinnt:

   konto = konto + 2

b. Verringere konto um 1, wenn der Spieler verliert:

   konto = konto - 1

c. Das Ergebnis sollte so ausschauen:

```
Dein Kontostand beträgt: 10
Wuerfelzahl eingeben!3
Die Zufalls-Würfelzahl ist 3
GEWONNEN
Dein Kontostand beträgt: 12
Wuerfelzahl eingeben!3
Die Zufalls-Würfelzahl ist 1
VERLOREN
```

6. **Begrüße den Spieler!**

   a. Gib „**Willkommen**“ am Anfang jeder Spielrunde aus

   b. Tipp: Verwende den Befehl **\n** für einen Zeilenumbruch:

      print("\n ===== WILLKOMMEN =========")

   c. Das Ergebnis sollte so ausschauen:

```
 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  3
Wuerfelzahl eingeben!3
Würfelzahl:  3
Du hast gewonnen, JUHU!

 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  5
Wuerfelzahl eingeben!3
Würfelzahl:  6
Leider verloren
```

7. **Beende das Spiel, wenn kein Geld mehr übrig ist!**

a. Verwende eine **verschachtelte If-Schleife** innerhalb der Endlosschleife

   i. Wenn **konto > 0** ist, soll das Spiel weiterlaufen

   ii. Ansonsten soll „**Du bist leider pleite**“ ausgegeben und das Spiel beendet werden

   Tipp: Mit dem Befehl **break** kannst du aus der Endlosschleife springen und das Spiel beenden.

b. Das **fertige Spiel** sollte so ausschauen:

```
 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  2
Wuerfelzahl eingeben!3
Würfelzahl:  5
Leider verloren

 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  1
Wuerfelzahl eingeben!3
Würfelzahl:  2
Leider verloren

 ===== WILLKOMMEN =========
Du bist leider pleite :(
>>>
```

Challenges

Tipp: Erstelle ein **„Backup“** deiner funktionierenden .py Datei, bevor du die Challenges beginnst!

Challenge 1

Implementiere eine **Wartezeit** zwischen Würfeln und Ergebnisanzeige!

a. Nach der Eingabe der Würfelzahl wird **… es wird gewürfelt …** für 1 Sekunde angezeigt

b. Erst dann wird das Ergebnis gezeigt

   Tipp: Verwende die Bibliothek time: **import time**

   time.sleep(1) lässt dein Programm für 1 Sekunde warten, bevor die nächste Codezeile ausgeführt wird

c. Das Ergebnis sollte so ausschauen:

```
 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  5
Wuerfelzahl eingeben!4
.... es wird gewürfelt ...
Würfelzahl:  1
Leider verloren

 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  4
Wuerfelzahl eingeben!3
.... es wird gewürfelt ...
```

Challenge 2

Programmiere einen **Zähler** für gewonnene/verlorene Spiele!

a. Erzeuge zwei Variablen VOR der Endlosschleife und setze sie auf 0: **gewonnen** und **verloren**

b. Erhöhe den Wert der Variable

   i. **gewonnen**, wenn der User gewinnt

   ii. **verloren**, wenn der User verliert

c. Gib die Werte beider Variablen nach dem Kontostand aus.

d. Das Ergebnis sollte so ausschauen:

```
 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  3
Du hast 0  Mal gewonnen und  2  Mal verloren
Wuerfelzahl eingeben!3
.... es wird gewürfelt ...
Würfelzahl:  3
Du hast gewonnen, JUHU!

 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  5
Du hast 1  Mal gewonnen und  2  Mal verloren
Wuerfelzahl eingeben!3
.... es wird gewürfelt ...
Würfelzahl:  4
Leider verloren
```

Challenge 3

Programmiere einen **Wahrscheinlichkeits-Simulator**!

a. Berechne die Wahrscheinlichkeit, dass zwei Würfel bei einem Wurf die **gleiche Zahl** anzeigen

b. Wenn der User als Input die Zahl „9“ eingibt, passiert folgendes:

   i. In einer for-Schleife werden 1000 Mal je **zwei Zufalls-Würfelzahlen** erzeugt

   ii. Die Würfelzahlen werden verglichen (If-Schleife)

      1. Die „**win**“ (Würfelzahlen sind gleich) und „**lose**“ (Würfelzahlen sind ungleich) Spiele werden gezählt (Variablen erhöhen)

c. Nach der for-Schleife wird die **Gewinn-Quote** mit folgender Formel berechnet:

   quote = win/(win+lose)

d. Gib die **win und lose** sowie die **Quote** aus

e. Das Ergebnis sollte so ausschauen:

```
 ===== WILLKOMMEN =========
Dein Kontostand beträgt:  4
Du hast 0  Mal gewonnen und  1  Mal verloren
Wuerfelzahl eingeben!9
Gewonnen:  163 Verloren: 837 Quote:  0.163
```

Von 1000 Spielen wurden **163** gewonnen und **837** verloren, das entspricht einer Gewinn-Wahrscheinlichkeit von **0.163** oder **16,3%**.

Das entspricht ziemlich genau der **Würfel-Wahrscheinlichkeit** von einem Sechstel oder 16.666%.

ANHANG C – FRAGEBOGEN UND FOKUSINTERVIEW

Fragebogen Visuelle Programmierung

Visuelle Programmierung

Liebe Schüler/innen,

in meiner Masterarbeit untersuche ich, ob das Programmieren im Team besser klappt als das Programmieren alleine.

Beim App-Inventor habt ihr zwei "Projekte" programmiert:

1. Maulwurf-Spiel (zu zweit programmiert - Pair Programming)
2. Pong-Spiel (alleine programmiert - Solo Programming)

In diesem Fragebogen geht es darum, wie es euch mit diesen zwei Arbeitsformen gegangen ist.

* Required

1. Bitte gib dein Alter und deinen Vornamen an. *

2. Was sind aus deiner Erfahrung die Herausforderungen beim Programmieren mit dem App Inventor? *

3. Welche Vorteile hat es aus deiner Erfahrung, Programme im App-Inventor zu zweit (Pair Programming) zu beabeiten? *

4. Welche Nachteile hat es aus deiner Erfahrung, Programme im App-Inventor zu zweit (Pair Programming) zu beabeiten? *

5. Fallen dir Beispiele ein, wo ihr euch beim Maulwurf-Spiel (Pair Programming) gegenseitig geholfen habt? Wenn ja, welche? *

6. Kreuze an, wie sehr du folgenden Aussagen zustimmst. *

Mark only one oval per row.

	Stimme gar nicht zu.	Stimme eher nicht zu.	Stimme teilweise zu.	Stimme voll zu.
Ich interessiere mich für Informatik.	◯	◯	◯	◯
Ich glaube, ich habe ein Talent für Informatik.	◯	◯	◯	◯
Das Programmieren mit dem App Inventor war für mich schwierig.	◯	◯	◯	◯
Den Tutorials konnte ich allein gleich gut folgen wie im Pair Programing.	◯	◯	◯	◯
Im Pair Programing fiel es mir leichter, den Tutorials zu folgen.	◯	◯	◯	◯
Im Pair Programing fiel es mir leichter, die „Challenges" zu lösen.	◯	◯	◯	◯

7. Welche Arbeitsform hat dir mehr Spaß gemacht? *

Mark only one oval.

- ◯ Pair Programing
- ◯ Solo Programing

8. Bei welcher Arbeitsform lernst du deiner Einschätzung nach mehr? *

Mark only one oval.

- ◯ Pair Programing
- ◯ Solo Programing

9. In welcher Arbeitsform kannst du deiner Einschätzung nach Probleme schneller lösen? *

Mark only one oval.

- ◯ Pair Programing
- ◯ Solo Programing

10. Wenn du ein größeres Projekt programmieren müsstest, welche Arbeitsform würdest du wählen? *

Mark only one oval.

- ◯ Pair Programing
- ◯ Solo Programing

11. Wenn du zusätzliche Anmerkungen hast, ist hier der Raum dafür: *

Fragebogen Textbasierte Programmierung

Textbasierte Programmierung

Liebe Schüler/innen,

in meiner Masterarbeit untersuche ich, ob das Programmieren im Team besser klappt als das Programmieren alleine.

Ihr habt ihr zwei "Projekte" in Python programmiert:

1. Würfel-Spiel (zu zweit programmiert - Pair Programming)
2. Bibliothek (alleine programmiert - Solo Programming)

In diesem Fragebogen geht es darum, wie es euch mit diesen zwei Arbeitsformen gegangen ist.

* Required

1. Bitte gib dein Alter und deinen Vornamen an. *

2. Was sind aus deiner Erfahrung die Herausforderungen beim Programmieren mit Python? *

3. Welche Vorteile hat es aus deiner Erfahrung, Programme in Python zu zweit (Pair Programming) zu beabeiten? *

4. Welche Nachteile hat es aus deiner Erfahrung, Programme in Python zu zweit (Pair Programming) zu beabeiten? *

5. Fallen dir Beispiele ein, wo ihr euch beim Würfel-Spiel (Pair Programming) gegenseitig geholfen habt? Wenn ja, welche? *

6. Kreuze an, wie sehr du folgenden Aussagen zustimmst. *

Mark only one oval per row.

	Stimme gar nicht zu.	Stimme eher nicht zu.	Stimme teilweise zu.	Stimme voll zu.
Ich interessiere mich für Informatik.	◯	◯	◯	◯
Ich glaube, ich habe ein Talent für Informatik.	◯	◯	◯	◯
Das Programmieren mit Python war für mich schwierig.	◯	◯	◯	◯
Den Tutorials konnte ich allein gleich gut folgen wie im Pair Programing.	◯	◯	◯	◯
Im Pair Programing fiel es mir leichter, den Tutorials zu folgen.	◯	◯	◯	◯
Im Pair Programing fiel es mir leichter, die „Challenges" zu lösen.	◯	◯	◯	◯

7. Welche Arbeitsform hat dir mehr Spaß gemacht? *

Mark only one oval.

- ◯ Pair Programing
- ◯ Solo Programing

8. Bei welcher Arbeitsform lernst du deiner Einschätzung nach mehr? *

Mark only one oval.

- ◯ Pair Programing
- ◯ Solo Programing

9. In welcher Arbeitsform kannst du deiner Einschätzung nach Probleme schneller lösen? *

Mark only one oval.

- ◯ Pair Programing
- ◯ Solo Programing

10. Wenn du ein größeres Projekt programmieren müsstest, welche Arbeitsform würdest du wählen? *

Mark only one oval.

- ◯ Pair Programing
- ◯ Solo Programing

11. Wenn du zusätzliche Anmerkungen hast, ist hier der Raum dafür: *

Leitfaden Visuelle Programmierung

Du wirst jetzt ein Spiel ausprobieren, das mit dem MIT App Inventor programmiert wurde und ich werde dir einige Fragen dazu stellen.

1. Stell dir vor, du musst dieses Spiel nachprogrammieren. Welche Herausforderungen könntest du bei der Programmierung erleben?
2. Womit würdest du beginnen?
3. Welche Vorteile könnte es haben, dieses Spiel zu zweit im PP zu programmieren?
4. Glaubst du, deine Motivation für die Programmierung wäre im PP höher? Warum?
5. Glaubst du, du könntest das Spiel besser im PP oder im SP programmieren? Warum?
6. Möchtest du sonst noch etwas sagen?

Leitfaden Textbasierte Programmierung

Du wirst jetzt ein Programm ausprobieren, das ich in Python programmiert habe und ich werde dir einige Fragen dazu stellen.

1. Stell dir vor, du musst dieses Spiel nachprogrammieren. Welche Herausforderungen könntest du bei der Programmierung erleben?
2. Womit würdest du beginnen?
3. Welche Vorteile könnte es haben, dieses Spiel zu zweit im PP zu programmieren?
4. Glaubst du, deine Motivation für die Programmierung wäre im PP höher? Warum?
5. Glaubst du, du könntest das Spiel besser im PP oder im SP programmieren? Warum?
6. Möchtest du sonst noch etwas sagen?