Universität Würzburg
Fakultät für Mathematik und Informatik
Institut für Mathematik
Prof. Dr. Martin Hennecke

Seminar Didaktik der Informatik

Motivierende Themen:
Spieleprogrammierung mit Scratch und BYOB

Wintersemester 2011/12

Michael Heinelt
3. Semester
eMail: michael.heinelt@stud-mail.uni-wuerzburg.de

Inhaltsverzeichnis

I Abbildungsverzeichnis 3

1 Scratch und BYOB als Motivation im Informatikunterricht 4

2 Didaktische Aspekte 5

2.1 Anwendung von *Scratch* und *BYOB* 5
2.2 Visuelle Programmierung 6
2.3 Verbreitung in der Online Community 8
2.4 Umsetzung von objektorientierter Programmierung 9

3 Scratch und BYOB im bayerischen Curriculum 12

3.1 7. Klasse 12
3.2 8. Klasse 15
3.3 10. Klasse 15
3.4 11. Klasse 18

4 Interdisziplinäres Beispiel 22

4.1 Projektidee 22
4.2 Sachanalyse 23
4.3 Implementierung 24

5 Fazit und Ausblick Lego NXT 30

6 Literaturverzeichnis 32

Abbildungsverzeichnis

1 Oberfläche von *Scratch* Quelle: vgl. Stoll et al., 2008, S. 86 5
2 Befehlsgruppen 6
3 Beispielcode im Programmbereich 7
4 Vereinfachtes Klassen- und Objektdiagramm von *Sprite* 10
5 Erstellen eines Blocks 1 11
6 Blockeditor 11
7 Kommunikation zwischen Objekten Quelle: (vgl. Harvey, Mönig, 2011, S. 24 onl.) 12
8 Beispiel *RobotKarol* 13
9 Beispiel einer Turtle-Grafik 14
10 Beispiel von Rekursion Quelle: vgl: Harvey, Mönig, 2011 ,onl. 18
11 Beispielcode von Rekursion Quelle: vgl: Harvey, Mönig, 2011 ,onl. 19
12 Binary Tree - Erstellung des Knotens Quelle: (Harvey, Mönig, 2011, S. 9, onl.) 20
13 Binary Tree - Implementierung eines Suchbaums 20
14 Binary Tree - Finden des kleinsten Elements 21
15 Spielidee 22
16 Der Einheitskreis 23
17 Skript von *Player* 24
18 Skript von *Ball* 25
19 Der Block *initialisiere* 26
20 Der Block *Bewegung* 26
21 Der Block *prallt vom Rand ab* 26
22 Der Block *prallt vom Player ab* 27
23 Der Block*Ball berührt Boden* 27
24 Skript von *Brick* 28
25 Skript von *Bühne* 29
26 Der Block *loescheKlone* 30

1 Scratch und BYOB als Motivation im Informatikunterricht

Viele Jugendliche wollten schon gerne einmal ein Spiel programmieren. Sie haben große Vorstellungen und Ideen bezüglich des Spiels, jedoch wenig, wenn nicht gar keine Programmiererfahrung. Sobald sie dann versuchen, ein Spiel mittels einer Programmiersprache umzusetzen, resignieren bald viele, da sie merken, wie schwer es ist, einen Ansatzpunkt zu finden und wie viel theoretisches Vorwissen man dazu braucht. Sie kennen zu viele Konzepte nicht und sind mit der abstrakten Denkweise schnell überfordert.

Ein Schüler muss sich mit dem Lehrstoff befassen, ob er nun ein persönliches Interesse daran hat oder nicht. So kann das Unterrichtsthema bei fehlendem Interesse schnell demotivierend wirken. Auf den Informatikunterricht trifft dies gleichermaßen zu. Daher ist es nicht verwunderlich , dass es schon viele Versuche gab, bei Jugendlichen das Interesse an der Programmierung zu wecken und den Einstieg in das Thema zu erleichtern.

„Seit Seymour Papert schon in den Sechzigerjahren LOGO als Programmiersprache für Kinder entwickelte, reißen die Bemühungen nicht ab, Kindern einen leichten Einstieg in die Welt der Programmierung zu ermöglichen.“ (Romeike, Wollenweber, 2009, S. 4). Seitdem wurden Programme entwickelt wie *RobotKarol* [1] oder *JavaKarol*, welche auch im Unterricht an bayerischen Gymnasien verwendet werden. Jedoch kommen mit der Zeit auch noch andere neue Programme als Hilfestellung bei der Programmierung hinzu, welche Beachtung finden. „Selten hat jedoch ein Werkzeug eine so schnelle und weltweite Verbreitung gefunden wie *Scratch*, das im Mai 2007 vom MIT Media Lab herausgegeben wurde“ (ebd.). So wurde „in jahrelangen Untersuchungen erforscht, welche Konzepte von Programmieranfängern intuitiv aufgenommen werden können.“ (ebd.)

Zusätzlich gibt es eine inoffizielle Weiterentwicklung namens *Build Your Own Blocks*, welche im nachfolgenen Text mit *BYOB* abgekürzt wird. Diese Weiterentwicklung ist graphisch sehr ähnlich zu *Scratch* aufgebaut, bietet aber mehr Funktionalitäten. Jedes *Scratch*-Programm lässt sich ohne Schwierigkeiten mit *BYOB* öffnen. Mit einer solchen Starthilfe sollen Jugendliche einen Einstieg in die Programmierung finden, welcher leicht verständlich und vor allem spielerisch möglich ist. Allerdings finden *Scratch* und

[1] Weiterführende Informationen zu *RobotKarol* siehe Abschnitt 3.1 S. 13 oder *http://www.schule.bayern.de/karol/* Verifizierungsdatum: 07.01.2012

BYOB in Bayern kaum eine Verbreitung. Hier haben sich *RobotKarol* und *JavaKarol* im Unterricht durchgesetzt. Daher soll in dieser Arbeit erörtert werden, welche didaktischen Aspekte *Scratch* und *BYOB* bieten, ob sie mit dem bayerischen Lehrplan vereinbar und damit im Unterricht einsetzbar sind und es soll aufgezeigt werden, wie viel Aufwand nötig ist, um ansprechende Spiele realisieren zu können.

2 Didaktische Aspekte

2.1 Anwendung von *Scratch* und *BYOB*

„Der Umgang mit *Scratch* wird [...] recht schnell erfasst - auf Intuitivität wird bei der Entwicklung besonders geachtet" (Romeike, 2010, S. 44). Ohne großen Aufwand sollen also Erfolge erzielt werden, so dass die Schülerinnen und Schüler schnell motiviert werden und ihr Interesse an der Informatik geweckt wird. Im Folgendem wird geprüft, ob die Entwickler diesem Anspruch gerecht geworden sind.

Dazu ist es nötig, sich die Oberfläche von *Scratch*, welche in Abb. 1 gezeigt ist, näher anzusehen. Im Objektbereich werden alle Objekte, die im Programm vorkommen und die Bühne verwaltet. Über *neues Objekt malen* lässt sich eine Oberfläche zum Malen öffnen, welche über Werkzeuge verfügt, wie sie aus einfachen Malprogrammen bekannt sind. Über *neues Objekt aus Datei laden* lassen sich weitere Objekte erstellen, welche der Benutzer aus einer großen Vorlagensammlung auswählen kann.

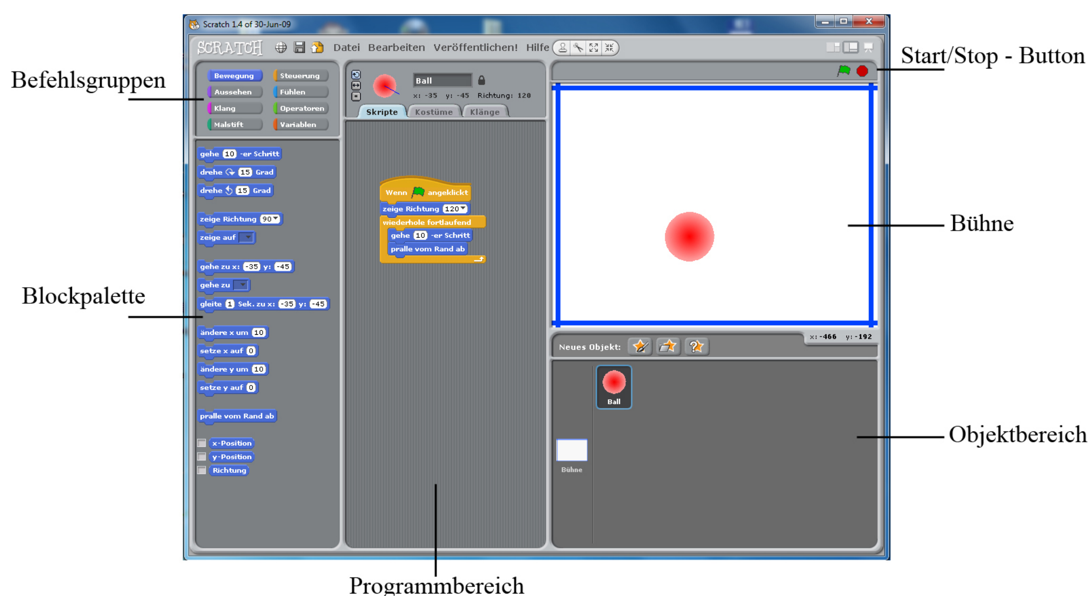


Abbildung 1: Oberfläche von *Scratch*
Quelle: vgl. Stoll et al., 2008, S. 86

Sobald der Benutzer ein Objekt angeklickt hat, werden seine schon programmierten Methoden in den Programmbereich geladen. Unter dem Reiter *Kostüme* kann das Aussehen, wie Form, Farbe oder Größe, mittels des Malprogramms bearbeitet werden und weitere Darstellungen des Objekts in Form von mehreren Kostümen angelegt werden. So kann das Objekt während der Programmausführung auf Befehl sein Aussehen verändern, indem es das Kostüm wechselt. Dadurch lässt sich z.B. leicht eine Animation erstellen. Der Benutzer kann bestimmte *Befehlsgruppen* wie *Bewegung* oder *Steuerung* auswählen und zieht die gewünschten Blöcke in den Programmbereich, welche einen sogenannten Stapel bilden. Sobald man auf den Startbutton (grüne Flagge) klickt, wird das Programm auf der Bühne gestartet und läuft solange, bis es entweder durch das Skript oder durch Klicken auf den roten Stopbutton beendet wird. Die Bühne verfügt über ein kartesisches Koordinatensystem, bei welchem der Koordinatenursprung genau im Mittelpunkt der Bühne liegt. Die Abszissenachse reicht von -240 bis +240 und die Ordinatenachse von -180 bis 180.

So bestätigt Baumann (2009, S. 55) das Anliegen der Entwickler, dass das Programm so intuitiv wie möglich bedienbar ist:

> *Fast ohne Anleitung verstanden die Schüler sofort die Einteilung der Benutzungsoberfläche in Programmierwerkzeuge (Kacheln), Programmbereich, Bühne und Objektbereich. Sie konnten ohne weiteres Objekte erstellen (malen und importieren), ihr Aussehen variieren, sie zu gegebenen Punkten gleiten lassen, Klänge beifügen usw.*

2.2 Visuelle Programmierung

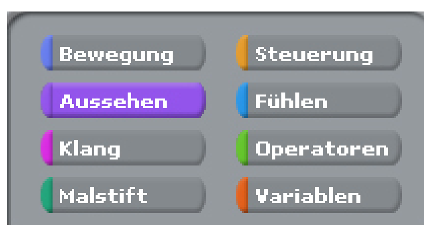


Abbildung 2: Befehlsgruppe

Wie in Abbildung 2 zu sehen ist, gibt es 8 verschiedene, inhaltlich getrennte Befehlsgruppen, aus denen man auswählen kann. Jede Befehlsgruppe hat ihre eigene Farbe und die dazugehörigen Blöcke sind in der gleichen Farbe gefärbt. Programmiert wird nun, indem man einen Block anklickt und mit gedrückter Maustaste in den Programmbereich zieht. Jeder Block hat eine bestimmte Form wie bei einem Puzzle und kann nur an oder in andere Blöcke gesetzt werden, wenn die Form es erlaubt. Insgesamt gibt es 3 Arten von Blöcken in *Scratch*. Zu einem sind das *stapelbare Blöcke*. Diese sind in Abb. 3 S.7 an den Zapfen an der Unterseite bzw. an der Ausbuchtung der Oberseite zu erkennen. Es können beliebig viele solcher Blöcke aneinander gesetzt werden. Weiterhin gibt es sogenannte *Hüte*. Diese sind dafür verwantwortlich, dass einzelne Stapel

im Programmbereich ausgeführt werden. Ein solcher *Hut* ist in Abb. 3 *Wenn grüne Fahne angeklickt*. Zuletzt gibt es die *Reporter*, welche entweder runde oder eckige Enden haben und nur in andere Blöcke passen. *Reporter* mit runden Enden liefern z.B. einen Variablenwert wie *x-Position* zurück, wohingehen *Reporter* mit eckigen Enden einen boolean Wert, wie bei Abb. 3 *wird Rand berührt* zurückgeben (vgl. Scratch Reference Guide, 2012, S. 10).

Abbildung 3: Beispielcode im Programmbereich

Um einen Anfang für das Programm zu setzen, wird aus der Befehlsgruppe *Steuerung* ein *Hut* mit der grünen Flagge in den Programmbereich gezogen. Anschließend wird eine Endlos-Schleife, die genau an die Form des Hutes passt, angehängt. Von der Schleife wird der restliche Code umschlossen. Wie man sehr schön sehen kann, passt in das Steuerungselement *falls* nur bestimmte Elemente, die eine rautenähnliche Form haben. Durch diese Art zu programmieren, wird ein wesentlicher Punkt erreicht: Die Schülerinnen und Schüler können sich gut auf die Semantik des Programms konzentrieren, da die Syntax durch die Art der visuellen Darstellung auf das Wesentlichste reduziert werden konnte. Nun könnte man bemängeln, dass die Syntax einfach übergangen wird. Jedoch meint Romeike (2010, S. 45) dazu, dass „mit *Scratch* zwar keine Syntaxfehler entstehen können, die Syntax einer Programmiersprache erschließt sich aber nicht durch das Beheben fehlender Kommata, sondern durch das Erfassen von Strukturen, die in *Scratch* extra hervorgehoben sind!“

Weiterhin ist auch der Unterricht an sich mit einer visuellen Programmiersprache wie *Scratch* sehr viel angenehmer zu gestalten, da die Lehrerin oder Lehrer direkt vor der Klasse am Whiteboard programmieren kann, anstatt immer zwischen Computer und Klasse wechseln zu müssen.

2.3 Verbreitung in der Online Community

„imagine, program, share“ (Website Scratch) - Dieser Ausspruch findet sich direkt unter dem Logo der Website *http://scratch.mit.edu/*, der offiziellen Website von *Scratch* und verdeutlicht die Denkweise der Entwickler und der Online-Community. So entstand „the YouTube of interactive media“ (Resnick et al., 2009 onl.). Junge Entwickler, welche in der Regel zwischen 8 und 16 Jahren alt sind, können durch einen einzigen Klick in der *Scratch*-Umgebung ihr Projekt auf der Website hochladen und bewerten lassen (vgl. Monroy-Hernández, Hill, 2010 onl.). Und dies ist auch der zusätzliche Reiz, in *Scratch* Programme zu schreiben:

> *The Scratch Online Community makes programming more engaging by turning it into a social activity. Hobbit, a 14-year-old member of the community explains: 'When I think about it, recognition for my work is what really drew me into Scratch. Other things played a part, but the feeling that my work would be seen is what really motivated me.' (Monroy-Hernández, Resnick, 2008 onl.)*

Hochgeladene Projekte stehen nun allen anderen Usern zur Verfügung und können heruntergeladen und gegebenfalls weiterentwickelt werden. Dieser Vorgang wird als *Remixing* bezeichnet, welches jedoch stark umstritten war. Viele User beschwerten sich, dass ihre Projekte ohne Nennung ihrer Namen als ursprüngliche Entwickler nur leicht verändert und von anderen Usern als ihr selbst geschriebenes Programm dargestellt wurden. Allerdings muss man bemerken, dass es durchaus positive Rückmeldung gab, wenn der ursprüngliche Programmierer bei einer Weiterentwicklung genannt wurde (vgl. ebd.). Daher wurden Gegenmaßnahmen ergriffen:

> *Administrators implemented a mechanism that automatically gave attribution by displaying a link to antecedent projects on every remix along with the user name of any antecedent project's creator. (Monroy-Hernández, Hill, 2010 onl.)*

Nun lässt sich durch eine grafische Visualisierung auf der Website die Fortentwicklung eines Projekts durch verschiedene Entwickler bis zu dem ursprünglichen Programmierer nachvollziehen. Obwohl das Remixing zu diesem Zeitpunkt schon mehr akzeptiert war, führten die Administratoren nach ungefähr einem Jahr eine zweite Neuerung ein:

> *The intervention consisted in the creation of a new section of the front page of the website that lists the three projects remixed most often recently. It is important to note that for the members of the community, having ones project included on the front page is very highly regarded. (ebd.)*

So wurde das *Remixing* weitgehend akzeptiert und auch als positiv bewertet. Weiterhin gibt es auf der *Scratch* Website ein großes Forum, in welchem sich User bei Fragen und Problemen gegenseitig helfen. Dieses ist jedoch auf Englisch, was für die meisten Schüler in dem betreffenden Alter durchaus ein Hinterungsgrund sein kann, in diesem Forum nach Hilfe zu suchen. In höheren Jahrgangsstufen kann dies natürlich auch einen fächerübergreifenden Anreiz bieten. Allerdings gibt es auch ein deutsches Forum, welches jedoch wesentlich kleiner ist, aber den Anforderungen vor allem am Anfang beim Einstieg in die *Scratch* Programmierung genügen sollte.

2.4 Umsetzung von objektorientierter Programmierung

Scratch und *BYOB* fallen in die Reihe der objektorientierten Sprachen. Da *BYOB* deutlich mehr objektorientierte Konzepte beinhaltet als *Scratch*, soll hier die Objektorientierung anhand von *BYOB* erläutert werden. *BYOB* besitzt eine prototypenbasierte Objektorientierung. Jedoch schreibt der bayerische Lehrplan eine klassenbasierende objektorientierte Programmierung vor. Daher soll nun kurz der Ansatz einer klassenbasierten Objektorientierung aufgezeigt und anschließend auf Elemente von *BYOB* eingegangen werden, welche klassenbasiert sind, da dies für die späteren Abschnitte von Relevanz sein wird.

Ein Objekt besitzt immer Attribute und kann mit der Umwelt mittels seiner Methoden kommunizieren. Nun sind viele Objekte, von ihren Eigenschaften und Methoden gleich und unterscheiden sich nur in den Werten ihrer Attribute. Wenn z.B. ein *Porsche* als Objekt modelliert werden soll, werden bei einer klassenbasierten Objektorientierung zuerst in der dazugehörigen Klasse *Auto* alle Eigenschaften in Form von Attributen definiert. Diese könnten *Farbe*, *PS*, *Kennzeichen* usw. sein. An dieser Stelle wird den Attributen noch keinen konkreten Wert zugewiesen. Zusätzlich sollen alle Objekte der Klasse *Auto* bestimmte Fähigkeiten haben wie *beschleunige()* oder *biege links ab()*, welche durch Methoden realisiert werden. Erst jetzt können Objekte wie *Porsche* oder *VW* mittels eines Konstruktors mit konkreten Werten instanziert werden.

Im Grunde bietet *BYOB* 2 Klassen an. Diese wären zu einem die Klasse *Stage* und zum anderen die Klasse *Sprite*. Aus der Klasse *Stage* wird direkt ein einziges Objekt instanziert, nämlich die *Bühne* und aus der Klasse *Sprite*, welche in Abb. 4 S. 10 dargestellt ist, können beliebig viele Objekte erzeugt werden, welche im Objektbereich zu finden sind.

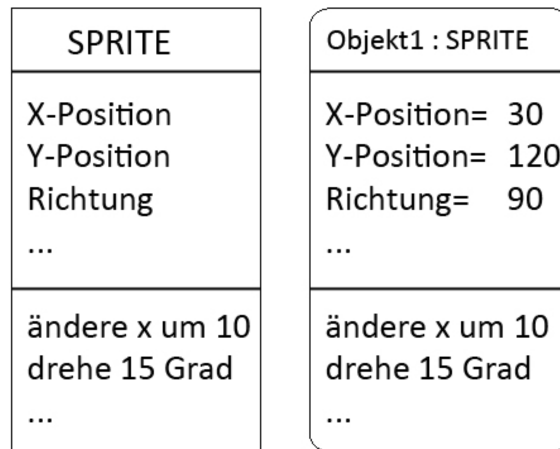


Abbildung 4: Vereinfachtes Klassen- und Objektdiagramm von *Sprite*

Diese Objekte haben nun Attribute[2] wie *X-Position*, *Y-Position* oder *Richtung*. Im Programmbereich können durch aneinandersetzen von Blöcken beliebig viele Stapel erstellt werden. Ein Block selbst bildet im Grunde auch schon eine eigenständige Methode.

Allerdings hat *BYOB* eine prototypenbasierte Objektorientierung und keine klassenbasierte. Um also mehrere gleiche Objekte zu erhalten, kann man sich diese nicht wie bei einer klassenbasierten Objektorientierung mehrmals instanzieren, sondern muss einen anderen Weg wählen.

Am Anfang wird ein konkretes Objekt erstellt, welches bestimmte Methoden und Variablen hat. Falls von diesen Objekt mehrere Kopien erwünscht sind, nimmt man dieses Objekt als Prototype und klont daraus weitere Objekte (vgl. Harvey, Mönig, 2011, S. 25 , onl.). In Bezug auf das obige Beispiel wird das Objekt *Porsche* konkret erstellt und von diesem dann das Objekt *VW* geklont und durch Änderung der Attributwerte dementsprechend angepasst. Der Prototype ist dann der *parent* der geklonten Objekten bzw. diese seine *children*. Sobald beim Prototype eine Änderung einer Eigenschaft erfolgt, wird dies automatisch auf die *children* übertragen. Wird beim *child* etwas geändert, ist dies nur für das *child* gültig. In der Praxis war es nicht möglich, eine Übertragung der Änderung vom Prototyp auf die *children* festzustellen, obwohl dies laut Referenzhandbuch von *BYOB* möglich ist.

> *Prototyping is also a better fit with the Scratch design principle that everything in a project should be concrete and visible on the stage; in class/instance OOP the programming process begins with an abstract, invisible entity, the class, that must be designed before any concrete objects can be made (ebd.).*

[2]Attribute, welche sich alle Objekte grundsätzlich teilen, nennt man in *BYOB Systemattribute*

Mit der Vererbung verhält es sich ähnlich. Man nehme an, dass es Objekte gibt, die sehr ähnlich sind zu anderen Objekten, aber noch zusätzliche Attribute und Methoden haben. Bei einer klassenbasierten Objektorientierung könnte bei dem obigen Beispiel der Fall sein, dass ein Rennauto zwar alle Attribute und Methoden der Klasse *Auto* beinhaltet, aber noch viele weitere besitzt. Dann würde die Klasse *Rennauto* alle Attribute und Methoden von der Klasse *Auto* erben und diese teilweise überschreiben oder erweitern. Bei *BYOB* kann dies bestenfalls umgesetzt werden, dass *children* teilweise verändert werden und dann selbst wiederum als Prototyp für weitere Objekte benutzt werden.

Weiterhin ist es auch in *BYOB* möglich, eigene Attribute, sogenannte Variablen zu erstellen, mit der Wahl, ob diese für alle bestehenden Objekte oder nur für das eigene gültig sind. Dementsprechend kann man sich auch selbst Blöcke zusammenbauen. Dazu klickt man in der Befehlsgruppe auf *Variablen* und wählt *neuer Block*. Hier kann ausgewählt werden, welcher *Befehlsgruppe* der zu erstellende Block angehören soll. Weiterhin kann die Art des Blocks festgelegt werden. In Abbildung 5 ist es in diesem Falle ein *Befehls*-Block. Diesem können nun beliebig viele Parameter übergeben werden. Anschließend gelangt man, wie in Abbildung 6 zu sehen ist, in den Blockeditor und kann sich seinen eigenen Block bauen. Sinnvollerweise wurden auch Skriptvariablen implementiert, so dass diese nur für diesen einen Block gültig sind.

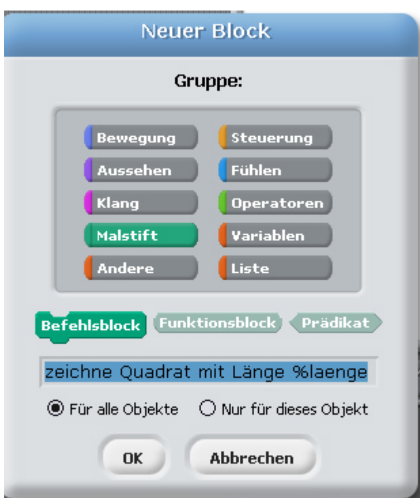


Abbildung 5: Erstellen eines neuen Blocks

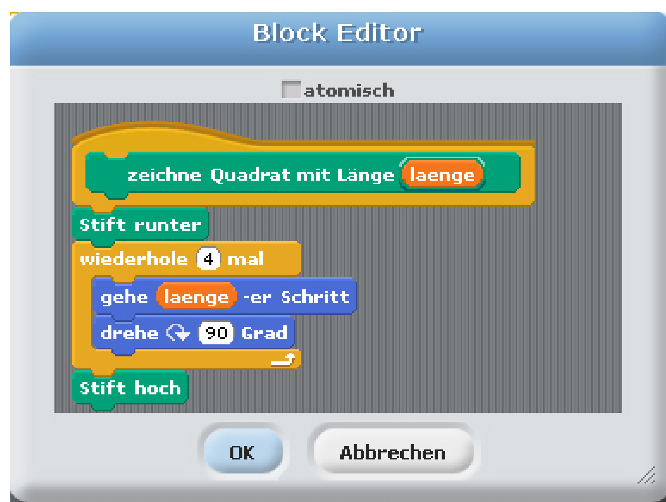


Abbildung 6: Blockeditor

Bei späteren Darstellung vom Bauen von Blöcken wird aus Platzgründen der Rahmen des Editors weggelassen.

Objekte können miteinander über sogenannte Nachrichten kommunizieren, wodurch eine Datenkapselung erreicht wird, welches eines der großen Prinzipien der objektorientierten Programmierung ist. Dabei kann ein Objekt den Block *sende [Nachricht] an alle*[3] ausführen. Der Hut *Wenn ich [Nachricht] empfange* kann dann auf diese Nachricht reagieren und den dazugehörigen Stapel ausführen. Es gibt noch eine 2. Möglichkeit, Objekte miteinander kommunizieren zu lassen, wie in Abbildung 7 zu sehen ist.

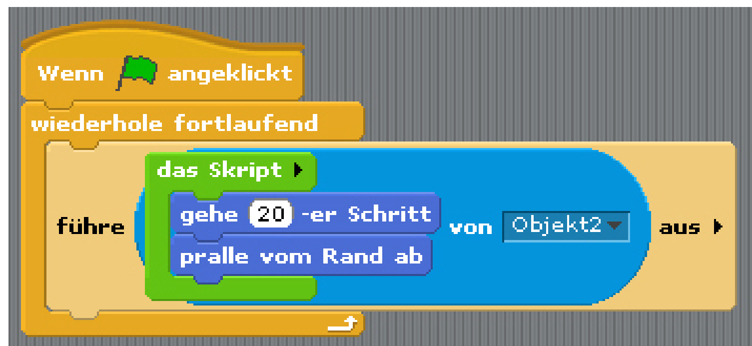


Abbildung 7: Kommunikation zwischen Objekten
Quelle: (vgl. Harvey, Mönig, 2011, S. 24 onl.)

Objekt1 steuert in diesem Stapel *Objekt2*, indem es auf die Methoden von *Objekt2* zugreifen und dadurch seine Attributwerte ändern kann.

Scratch hingegen besitzt in Hinsicht auf Objektorientierung nur den Gedanken, dass ein Objekt aus Attributen und Methoden besteht und über die Möglichkeit, mittels Nachrichten zu kommunizieren.

3 Scratch und BYOB im bayerischen Curriculum

In den folgenden Abschnitten soll nun die Einsatzmöglichkeit von *Scratch* und *BYOB* im bayerischen Curriculum erörtert werden und eventuelle Grenzen ausgelotet werden.

3.1 7. Klasse

In der 7. Klasse besuchen die Schülerinen und Schüler an bayerischen Gymnasien den Unterricht Natur und Technik. In diesem Unterricht kommt auch der Schwerpunkt Informatik vor, welcher aus Vernetzte Informationsstrukturen - Internet, Austausch von Information – E-Mail und Beschreibung von Abläufen durch Algorithmen besteht (vgl. Lehrplan Bayern Jgst 7, S. 35f). Hier bietet sich nur für Beschreibung von Abläufen

[3]Innerhalb der eckigen Klammern steht die zu übertragene Nachricht

durch Algorithmen an, *Scratch* in den Unterricht mit einzubinden. Natürllich könnte *BYOB* auch verwendet werden, aber da die Schüler noch relativ wenig Erfahrung in diesem Fach besitzen, wird *Scratch* für diese Jahrgangsstufe auf jeden Fall ausreichen. Die Schüler sollen „lernen, dass sich ganz allgemein mit Algorithmen Abläufe präzise und verständlich beschreiben lassen und üben an konkreten Sachverhalten [...] Vorgänge aus einfachen Bausteinen aufzubauen" (ebd.). Diese Bausteine sind dann beispielsweise Anweisung, Sequenz, Bedingte Anweisung, Wiederholung (ebd.). Nach dem Erlernen der einzelnen Bausteine ist anschließend das Programmieren eines einfachen Informatiksystems unter Verwendung dieser Bausteine erforderlich (vgl. ebd.).

Es ist gängige Praxis, dieses Thema mit Hilfe des Programms *RobotKarol* einzuführen. In *RobotKarol* befindet sich ein Roboter in einer begrenzten Welt. Der Roboter kann nun kachelweiße in dieser Welt mit Hilfe einer *Subsprache* umherlaufen, Markierungen oder Ziegeln setzen und diese wieder beseitigen. Variablen oder deren Übergabe an Methoden sieht diese Sprache nicht vor. *RobotKarol* besitzt 2 Klassen. Zum einem die Klasse *Roboter*, welche z.B. die Koordinaten und die Richtung als Attribute besitzt und mehrere Methoden (als Beispiel siehe Abb. 8). Zum anderen gibt es die Klasse *Welt*, welche mit *Höhe*, *Breite* und *Länge* festgelegt ist. Zusätzlich hat sie noch Möglichkeiten, den Standort von Markierungen und Ziegeln zu speichern.

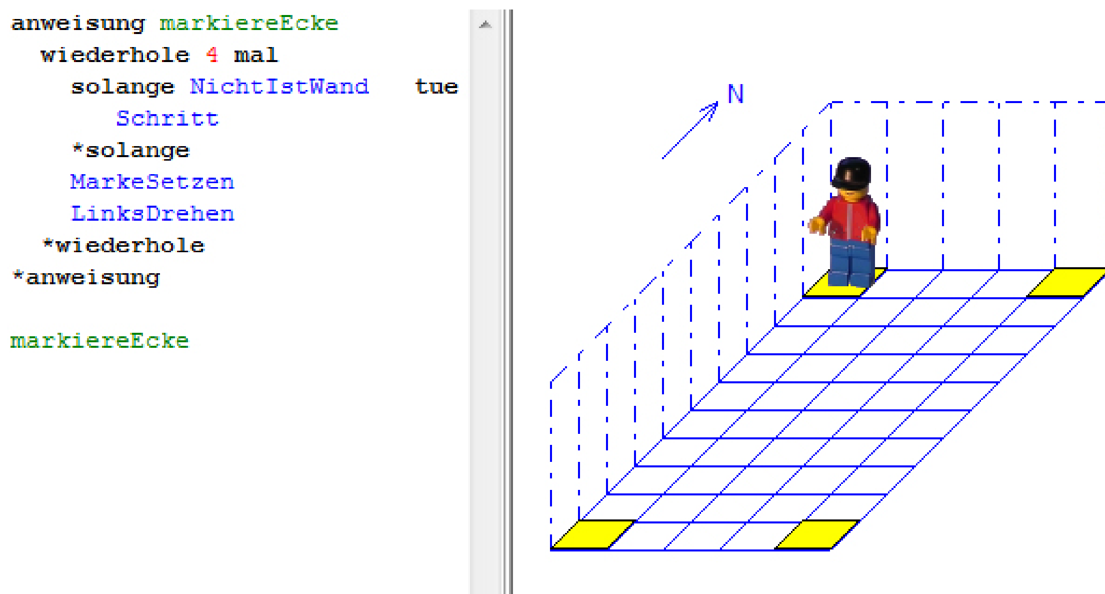


Abbildung 8: Beispiel *RobotKarol*

Wie man sieht, setzt RobotKarol nach der Ausführung des Quellcodes hier in die Ecken seiner Welt eine Markierung. Schwierigkeiten können hier bei der Syntax auftreten, da das Programm nicht richtig kompilieren kann, wenn ein Schüler z.B. das * vor dem *solange* nicht aufschreibt.

In *Scratch* ist die Wiederholung und Vertiefung von Klassen- und Objektdiagrammen ebenso möglich. Wie Abbildung 4 S. 10[4] zeigt, gibt es auch 2 Klassen in *Scratch*, mit denen man arbeiten kann. Allerdings sind solche Aufgabenstellungen wie bei *RobotKarol* beim Üben der Algorithmik nur schwierig auszuführen, dafür ermöglicht *Scratch* aber wiederum auch viele andere.

Eine große Aufgabenvielfalt dürften *Turtle-Grafiken* bieten. Bei einer *Turtle-Grafik* bewegt sich ein Objekt über die Bühne und setzt je nach Befehl einen Stift ab. Auf die 7. Jahrgangsstufe bezogen werden sich zwar nur sehr einfache *Turtle-Grafiken* erstellen lassen, aber das Ergebnis dürfte trotzdem sehr motivierend sein.

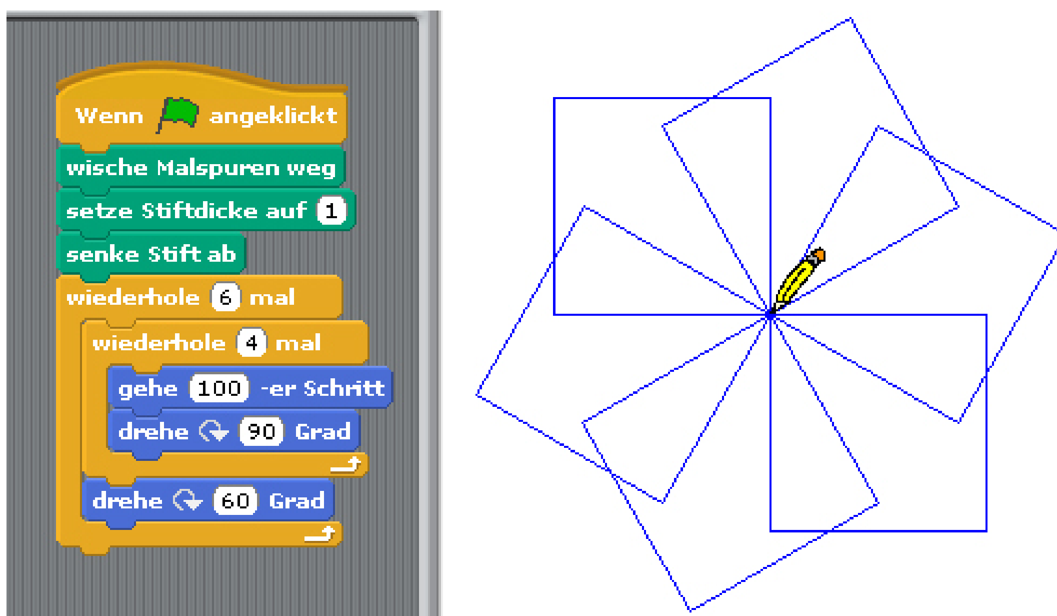


Abbildung 9: Beispiel einer Turtle-Grafik

Für den Fall, dass in diesem Beispiel von Abbildung 9 verschachtelte Schleifen noch zu schwer für die Schülerinen und Schüler sein sollte, könnte man das Material in *BYOB* didaktisch aufbereiten und vorher die Sequenz, die das Quadrat zeichnet, zu einem Block *zeichne Quadrat* zusammenfassen, wie es in Abb. 6 S. 11 zu sehen ist. Natürlich kann unter dem Punkt *Bearbeiten* → *Ausführen in Einzelschritten* in einen Modus gewechselt werden, so dass bei Skriptausführung klar verständlich ist, an welcher Stelle sich das Programm befindet.

Ansonsten ist der Pool an möglichen Aufgaben durch die vorgefertigten Blöcke von *Scratch* sehr groß. So kann beispielsweiße die Aufgabe, ein Objekt soll den Mauszeiger verfolgen, sehr einfach gelöst werden.

[4]Obwohl sich diese Abbildung auf *BYOB* bezieht, ist diese gleichermaßen für *Scratch* gültig

3.2 8. Klasse

In der 8. Klasse kommt kein Informatikunterricht an bayerischen Gymnasien vor. Dies ist leider sehr unvorteilhaft, da dadurch viel Inhalt bei den Schülerinnen und Schüler vergessen wird. Aus diesem Grund wäre es sehr gut, ein Wahlkurs Informatik für interessierte Schülerinnen und Schüler anzubieten.

Der Unterricht könnte von der Lehrkraft sehr frei gestaltet werden, da es keinen Lehrplan gibt, an den er sich halten muss. Da die Schülerinnen und Schüler zu diesem Zeitpunkt noch eher wenige tiefergehende informatische Konzepte kennen, ist die mögliche Aufgabenvielfalt jedoch eingeschränkt. Allerdings bietet *Scratch* unter diesem Aspekt noch viele Möglichkeiten an. Eine dieser Möglichkeiten wäre es, einen kleinen Animationsfilm zu erstellen. Jeder Schüler könnte sich eine eigene Story überlegen und diese in *Scratch* umsetzen. Eine weitere Möglichkeit wäre es, Objekte auf dem Bildschirm auf Befehl tanzen oder das Kostüm wechseln zu lassen. Dabei wird weiterhin das Verständnis für Objekte gefördert, da der Schüler sich über deren Attribute Gedanken machen muss, beispielsweise welches Kostüm sie tragen oder an welcher Stelle sie sich befinden sollen und natürlich auch über deren Methoden, die Attribute zu verändern. Weiterhin kann das *message passing* eingeführt werden, damit Objekte miteinander kommunizieren können. Es besteht auch die Möglichkeit, auf *http://scratch.mit.edu/* Ideenanregungen für den Unterricht zu finden. Es gibt hier also viele Möglichkeiten, Informatik den Schülern näher zu bringen und es sind kaum Grenzen gesetzt außer der eigenen Kreativität.

3.3 10. Klasse

In der 10. Klasse ändert sich in der gängigen Praxis der Informatikunterricht dahingehend, dass alles aufeinander aufbaut und Schüler schnell den Anschluss verlieren können. Es wird nun in der Regel mit einem Editor programmiert und die Schülerinnen und Schüler müssen sehr viele Informationen verarbeiten. Sie sollen die Syntax verstehen und viel wichtiger die Semantik.

Das erste große Thema in der 10. Klasse ist Objekte und Abläufe. Dieses beginnt mit der Zusammenfassung und Festigung der bisher erlernten objektorientierten Konzepte (vgl. Lehrplan Bayern Jgst. 10, S. 59f). Die Schüler wiederholen die Begriffe Objekt, Methode, Attribut und Klasse (vgl. ebd.), so wie sie es aus dem früheren Informatikunterricht kennen. Hierfür ist *BYOB* sehr gut geeignet, da man z.B. die Klasse *Sprite* aufstellen kann und auch ein Objektdiagramm dazu zeichnen kann wie in Abb. 4 S. 10 gezeigt ist. So lernen sie, dass ein Objekt eine Kombination aus Attributen und Me-

thoden (vgl. ebd.) ist. Auch die konkrete Einführung von Methoden kann mit Hilfe von *BYOB* erfolgen. Methoden können in Form von Blöcken mit Eingangsparametern gebaut werden, welche optional einen Rückgabewert haben.

Weiter geht es mit Zustände von Objekten und algorithmische Beschreibung von Abläufen (vgl. ebd.). „Die Schüler lernen, die Veränderungen von Objekten mithilfe von Zuständen und Übergängen zu beschreiben sowie mit Zustandsübergangsdiagrammen zu dokumentieren“ (ebd.). Ein Zustand eines Objekts wird beschrieben in der Gesamtheit aller Attributwerte. Durch Aufrufen einer Methode, welche mindestens einen Attributwert ändert, befindet sich das Objekt in einem Zustandsübergang und nach Beendigung der Methode in einem neuen Zustand. Auch dies ist mit *BYOB* sehr gut möglich, da jedes *Objekt* Systemattribute wie *X-Koordinate* oder *Richtung* hat, mit welcher man das Konzept Zustand sehr gut verdeutlichen kann. Didaktisch sinnvoll ist es, alle zu betrachtende Attribute währenddessen einzublenden. Dies erreicht man, indem man in *BYOB* unter der Befehlsgruppe *Bewegung* einen Hacken neben den Attributen setzt. Allerdings kann *BYOB* hier auch nur als Hilfestellung dienen, da der Lehrplan weiter vorschreibt: „Lebenszyklus von Objekten von der Instanzierung über die Initialisierung bis zur Freigabe“ (ebd.). Da keine konkreten Klassen und damit kein Konstruktor vorhanden sind, lässt sich das Objekt nicht initialisieren und anschließend instanzieren. Eine Möglichkeit wäre es, um eine Klasse zu simulieren, einen Prototypen zu bauen, diesen zu verstecken und dann von diesem Objekte zu klonen.

In diesem Themenbereich soll auch der Datentyp *Feld* besprochen werden. Diesen Datentyp wird man in *BYOB* nicht direkt finden. Man kann sich allerdings mit Hilfe von Listen ein mehrdimensionales Array nachbauen[5].

Als nächstes größeres Thema geht es um Kommunikation zwischen Objekten durch Aufruf von Methoden (vgl. ebd.). Dabei sollen Schnittstellen definiert werden, welche einen Eingangs- und Ausgangswert haben. Ganz am Anfang dieses Themas könnte man die Kommunikation zwischen Objekten mit dem Versenden von Nachrichten in *BYOB* bzw. durch Aufrufen von Methoden (siehe Abbildung 7 S. 12) dargestellt werden. Allerdings ist es in *BYOB* nicht möglich, eine *enthält-Beziehung* zu bauen, welche die häufigste Art von Kommunikation von Objekten ist. Somit kann *BYOB* am Anfang lediglich als Einstiegshilfe dienen, mehr in diesem Fall leider nicht.

[5]Ein Beispiel ist bei Modrow et al. (2011 S. 38) zu finden.

Gegen Ende des Schuljahres wird noch das Thema Generalisierung und Spezialisierung angesprochen (vgl. ebd.). Die Schüler sollen mit Hilfe der Vererbung hierarchische Strukturen bauen, Gemeinsamkeiten von Klassen mit Hilfe der Generalisierung erkennen und deren Unterschiede durch Spezialisierung verfeinern. Da *BYOB* prototypenbasiert ist, findet sich kaum eine konkrete Vererbung, wie sie aus einer klassenbasierten Objektorientierung wie von *Java* bekannt ist. Zwar ändern sich Methoden, wenn der Prototyp geändert wird, dies ist aber vielmehr so, als wenn von einer einzelnen Klasse eine Methode geändert wird und alle Objekte nach erneuter Instanzierung diese Änderung übernehmen. Man könnte Vererbung allenfalls daran deutlich machen, dass man ein geklontes Objekt in irgendeiner Weiße ändert und davon wieder mehrere Objekte klont.

Letztendlich kann *BYOB* am Anfang der 10. Klasse helfen, bestimmte Sachverhalte zu verdeutlichen, es genügt aber insgesamt den Anforderungen des momentanten bayerischen Lehrplans nicht, da schlichtweg eine klassenbasierte Objektorientierung vorgeschrieben ist. Allerdings muss dem Lehrkörper auch bewusst sein, dass der Umstieg von einer visuellen Programmierung zur Programmierung in einem Editor eine große Herausforderung ist. Es wird für die Schüler vor allem ungewohnt sein, nicht nur mit der Semantik richtig umzugehen, sondern auch mit der Syntax, welche eine häufige Fehlerquelle ist. Dem kann man mit einer Anpassung der Blockbennenung entgegenwirken[6], jedoch wird es für den Schüler immer sehr schwer sein, anschließend in einem Editor zu programmieren.

[6]Die Datei zum Anpassen der Bezeichnung ist unter *BYOB/locale/de.po* zu finden

3.4 11. Klasse

Trotz der Grenzen von *BYOB*, welche im vorigen Abschnitt aufgezeigt wurden, bietet das Programm sich immer noch als Hilfestellung an, komplizierte Sachverhalte vor allem visuell darzustellen und dem Schüler damit einen leichteren Zugang zum Lehrinhalt zu ermöglichen.

Dies ist auch in der 11. Klasse der Fall. Nachdem die Schülerinnen und Schüler die Datenstruktur *Schlange* untersucht haben , lernen sie daraufhin die dynamische Datenstruktur *Liste* kennen (vgl. Lehrplan Bayern Jgst. 11, S. 107f). Ein große Bedeutung spielt dabei das allgemeine Prinzip und die rekursive Struktur einer einfach verketteten Liste (vgl. ebd.). Jedoch ist Rekursion ein sehr schwieriges Thema an sich, da es eine neue Denkweise fordert, die man aus dem Alltag nicht kennt. Daher ist es didaktisch sinnvoll, diese an sich erst einmal visuell zu veranschaulichen. Das folgende Beispiel ist aus dem Handbuch von *BYOB* entnommen. Aufgabe ist es, einen stark vereinfachten Baum aus einzelnen Strichen zu malen, so wie er in Abbildung 10 zu sehen ist.

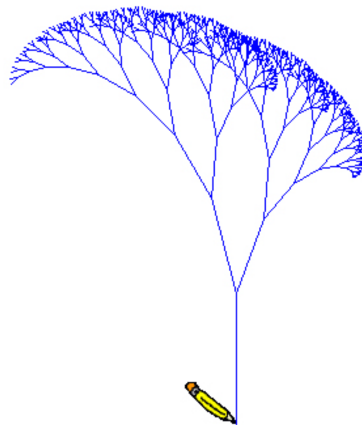


Abbildung 10: Beispiel von Rekursion
Quelle: vgl: Harvey, Mönig, 2011,onl.

Die Schülerinnen und Schülern sollen verstehen, dass eine strukturierte Programmierung in diesem Falle nicht sinnvoll ist und es in komplizierteren Fällen nicht möglich ist, dieses Bild ohne Fehler nach zu malen. Eine Analyse des Baums zeigt, dass jeder Ast wiederum einen kleineren Baum darstellt usw. Anschließend soll Rekursion am Code dieses Beispiels erläutert werden. Dieser ist in Abbildung 11 S. 19 dargestellt.

Abbildung 11: Beispielcode von Rekursion
Quelle: vgl: Harvey, Mönig, 2011 ,onl.

Der Ablauf soll hier nur kurz erläutert werden. Im linken Teil der Abbildung 11 wird ein neuer Block gebaut. Dieser hat 2 Eingabeparameter names *depth* und *size* und keinen Rückgabewert. *Depth* gibt die Tiefe, also die Anzahl der Verzweigungen, an und *size* die Länge des ersten Stammes. Anschließend wird als erstes wesentliches Element die Abbruchbedingung definiert, dass $depth > 0$ sein muss. Falls dies der Fall ist, wird ein Strich, beim ersten mal also der Stamm, gemalt, anschließend wird um 15 Grad nach links gedreht und wieder die Methode *tree* aufgerufen. Dieser wird natürlich wieder eine verminderte Tiefe und eine verkleinerte Größe übergeben. Nach Ausführen aller Rekursionsschritte wird anschließend genau so der rechte Ast gezeichnet. Anschließend geht es zurück an den jeweiligen Ausgangspunkt. Auf der rechten Seite der Abb. 11 sieht man die konkrete Ausführung im Programmbereich.

Nach der Veranschaulichung der Rekursion geht es nun um das Thema *einfach verkettete Liste* (vgl. ebd.). Diese soll mit Hilfe der Rekursion implementiert werden. Zwar bietet *BYOB* auch die Datenstruktur *Liste* an, jedoch erkennt man nicht, wie *BYOB* mit diesen arbeitet, sondern nur die Funktionalität an sich. So kann man bestenfalls mit Hilfe von *BYOB* verdeutlichen, was man erreichen will, wie z.B. das Einfügen eines Elements in eine Liste, mehr jedoch auch nicht.

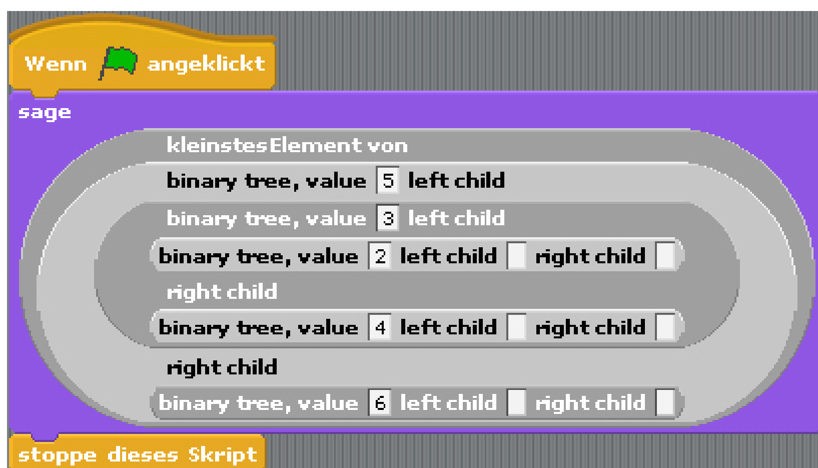
Als weitere große Datenstruktur kommen Bäume als spezielle Graphen (vgl. ebd.). Für die Beschreibung der Funktionsweise eines Baums ist *BYOB* ungeeignet. Für eine konkrete Implementierung bietet *BYOB* sich auch nicht an, da man im Grunde nur über eine dynamischen Datenstruktur in Form der *Liste* verfügt. Theoretisch kann man sich über eine Liste jegliche Datenstruktur in *BYOB* nachbauen, wie Abbildung

12 zeigt, jedoch ist es eleganter, dies in einer Sprache zu realisieren, welche klassen-objektorientiert ist. Trotzdem soll kurz anhand eines Beispiels aufgezeigt werden, dass dies grundsätzlich möglich ist.

Es soll das kleinste Element eines binären Suchbaums gefunden werden. Bevor dies geschieht, wird erst einmal ein Knoten als Block gebaut, wie in Abbildung 12 gezeigt.

Abbildung 12: Binary Tree - Erstellung des Knotens
Quelle: (Harvey, Mönig, 2011, S. 9, onl.)

Da wie oben erwähnt als einzige dynamische Datenstruktur eine *Liste* zur Verfügung steht, muss man sich aus dieser mit Hilfe eines selbstgebauten Blocks eine solche erstellen. Als erstes Element dieser Liste wird der Information halber **binary-tree** eingetragen. Dieser Wert ist also später unveränderlich. Der erste wirkliche Parameter ist *value*, welcher den Wert des Knotens speichert. Um das Beispiel zu vereinfachen, wird davon ausgegangen, dass immer eine Zahl übergeben wird. Die nächsten beiden Parameter sind *left* und *right* und speichern das jeweils linke oder rechte Kind.



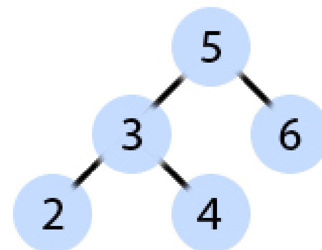


Abbildung 13: Binary Tree - Implementierung eines Suchbaums

Daraus wird nun der eigentliche Baum wie in Abb. 13 im Programmbereich erstellt. In diesem Fall ist der Knoten mit dem Wert *5* die Wurzel des Baumes, welcher dann noch ein left-child und ein right-child hat. Der Knoten mit dem Wert *3* hat auch nochmal 2 Kinder. Sobald ein Knoten kein Kind mehr hat, wird nichts mehr in die entsprechende Stelle der Liste eingetragen. Leider mangelt es an übersichtlichkeit in dieser Darstel-

lungsweise von *BYOB*. Der Baum ist schon geordnet, wie man in Abbildung 13 sieht, und das kleinste Element befindet sich ganz links, wie man es erwarten würde.

Abbildung 14: Binary Tree - Finden des kleinsten Elements

Das Auffinden des Knotens mit dem kleinsten Wert beruht auf folgenden Prinzip: der Algorithmus, welcher in Abb. 14 gezeigt ist und mittels eines Reporter-Blocks gebaut wird, untersucht zunächst an der Wurzel des Baumes , ob der Knoten wirklich ein **binary-tree** ist. Falls dann noch ein linkes Kind vorhanden ist, also deren Länge der Liste > 0 ist, wird der Algorithmus rekursiv auf das linke Kind angewendet. Dies wiederholt sich so lange, bis kein linkes Kind bei einem Knoten vorhanden ist, also ein leeres Feld in der Liste vorhanden ist. Dies entspricht der Abbruchbedingung und der Wert des aktuellen Knotens wird mittels *berichte* zurückgegeben, in diesem Fall also 2. Dieser Code würde in *Java* vom logischen Aufbau her nicht viel anders programmiert werden. Allerdings sind die Voraussetzungen einer konkreten Implementierung in *BYOB* wie gezeigt sehr umständlich. Es ist aber also durchaus möglich, kompliziertere Datenstrukturen mit *BYOB* zu realisieren.

Weiterhin könnte man auf ähnliche Weise Graphen nachbilden, allerdings würde diese Ausführung den Rahmen dieser Arbeit sprengen. Es sollte nur gezeigt werden, dass es mittels Listen grundsätzlich möglich ist, Datenstrukturen nachzubauen.

Am Ende der 11. Klasse erfolgt noch das Thema Softwaretechnik mit einer praktischen Umsetzung (vgl. ebd.). Auch hier ist wie aus dem gleichen Grund in der 10. Klasse angeraten, eine Sprache mit einer klassenbasierten Objektorientierung wie z.B. *Java* anzuwenden.

4 Interdisziplinäres Beispiel

4.1 Projektidee

An dieser Stelle soll nun ein einfaches Spiel realisiert werden, um die Möglichkeiten und Grenzen von *Scratch* und *BYOB* aufzuzeigen. In diesem Falle wird mit *BYOB* programmiert, da es wie schon aufgezeigt, mehr Möglichkeiten in der Programmierung bietet.

Es wird, wie in Abbildung 15 gezeigt, eine Art *Breakout*-Klon entwickelt. Ziel ist es, mit einem Ball alle so genannten Bricks abzuschießen. Sobald der Ball einen Brick berührt, prallt er ab und der Brick verschwindet. Dabei darf allerdings der Ball nicht den Boden berühren, sondern muss vom Player mittels des Balkens, der nur horizontal bewegt werden kann, in der Luft gehalten werden. Sobald der Ball den Boden berührt oder alle Bricks abgeschossen wurden, erscheint auf der Bühne eine jeweilige Meldung und das Spiel ist zu Ende.

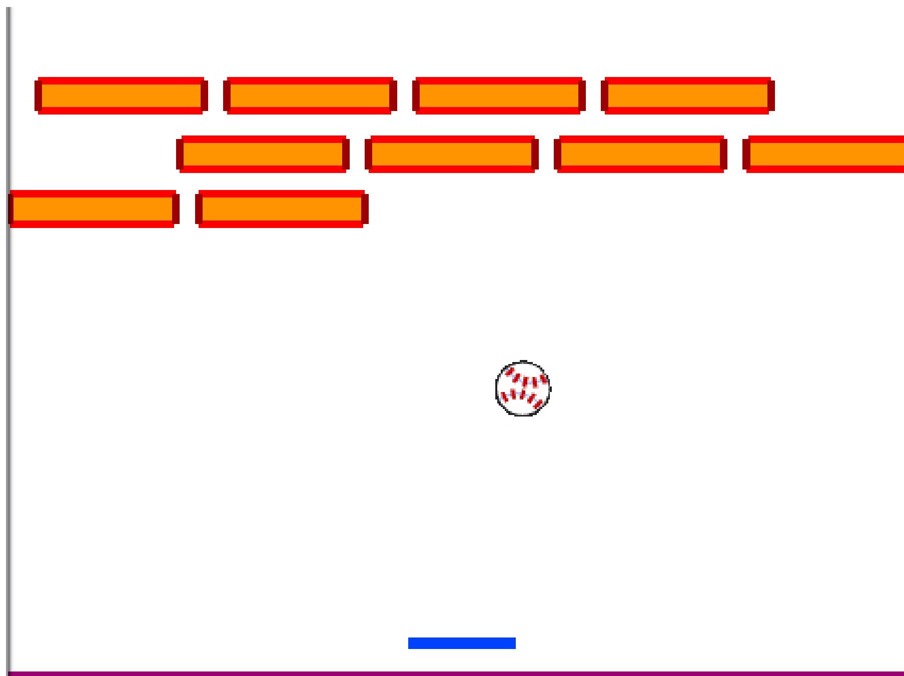


Abbildung 15: Spielidee

Das Spiel soll so dynamisch wie möglich programmiert werden. Daher soll es möglich sein, eine beliebige Anzahl von Bricks zu erzeugen und auf der Bühne automatisch anzuordnen. Dadurch wird die Programmierung allerdings etwas komplexer, als wenn man eine feste Anzahl von Bricks benutzt, da bei einer festen Anzahl die Verwaltung aller erzeugten Bricks nicht realisiert werden muss.

4.2 Sachanalyse

Jedes Spiel folgt mathematischen und auch oft physikalischen Regeln. So ist es auch in diesem Beispiel. Der Ball muss am *Player*, an den *Bricks*, wie an der *Wand* abprallen. Dies kann durch verschiedene Möglichkeiten erreicht werden. In diesem Beispiel wird die Geschwindigkeit in 3 Faktoren zerlegt, nämlich in das Tempo, die X-Komponente und die Y-Komponente. Zunächst soll die Mathematik zu dieser Überlegung beschrieben werden:

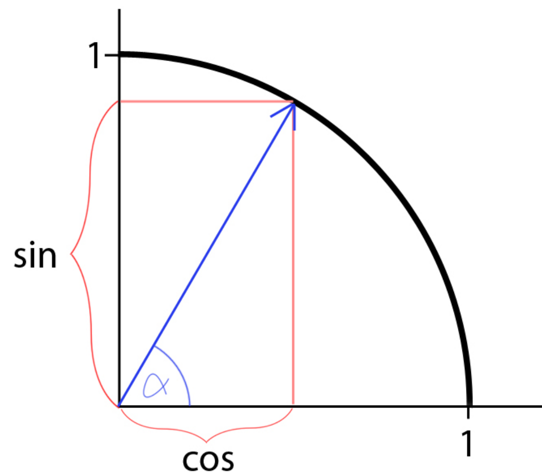


Abbildung 16: Der Einheitskreis

Jeder 2-dimensionaler Vektor kann in seine x-Komponente und y-Komponente zerlegt werden. Der Betrag des Vektors wäre in diesem Beispiel das Tempo. Die x-Komponente erreicht man durch

$$x_{Komponente} = sin(Richtung) * Tempo \tag{1}$$

Gleiches gilt für die y-Komponente

$$y_{Komponente} = cos(Richtung) * Tempo \tag{2}$$

unter der Annahme, dass $Richtung = \alpha$ ist.

Die Bewegung des Balls resultiert aus der Änderung der X-Koordinate um die X-Komponente und der Y-Koordinate um die Y-Komponente. Durch simple Vektoraddition fliegt somit der Ball mit dem richtigen Tempo in die richtige Richtung. Das Prinzip von Kräftezerlegung in einfachen Fällen[7] haben die Schülerinnen und Schüler im Phy-

[7]Das Prinzip der Zerlegung eines Vektors findet bei Kräften wie auch der Geschwindigkeit gleichermaßen Anwendung

sikunterricht der 9. Klasse schon kennengelernt und sind somit damit vertraut (vgl. Lehrplan Bayern Jgst. 9, S. 47).

Der Vorteil dieser zunächst kompliziert scheinenden Berechnung der Bewegung des Balls liegt darin, dass das Abprallen von Wand, Player und Bricks erheblich eleganter berechnet werden kann, welches immer nach dem gleichen Prinzip abläuft. Prallt der Ball, welcher z.B. von schräg oben nach unten über den Bildschirm fliegt, gegen den Player, dreht sich der Vektor der Y-Komponente der Geschwindkeit um, er negiert sich also. Die X-Komponente bleibt unbeeinflusst. In diesem Fall würde also nach Kollisionserkennung gelten:

$$y_{Komponente} = y_{Komponente} * (-1) \tag{3}$$

Nach dem gleichen Prinzip geschieht es auch mit der X-Komponente, wenn der Ball z.B. gegen die Seitenwand prallt.

4.3 Implementierung

An dieser Stelle wird nun die konkrete Implementierung des Spiels erfolgen: vorerst werden 3 Objekte erzeugt, welche zu einem der *Player*, der *Ball* und ein *Brick* sind. Das Objekt *Brick* soll dann als Prototyp für die restlichen *Bricks* dienen.

Als erstes wird der *Player* erstellt.
Dieser besteht aus einem blauen Balken, der sich horizontal über den Bildschirm bewegen kann, allerdings nicht die Bühne verlassen darf. Die Steuerung soll über die linke und rechte Pfeilstaste erfolgen.

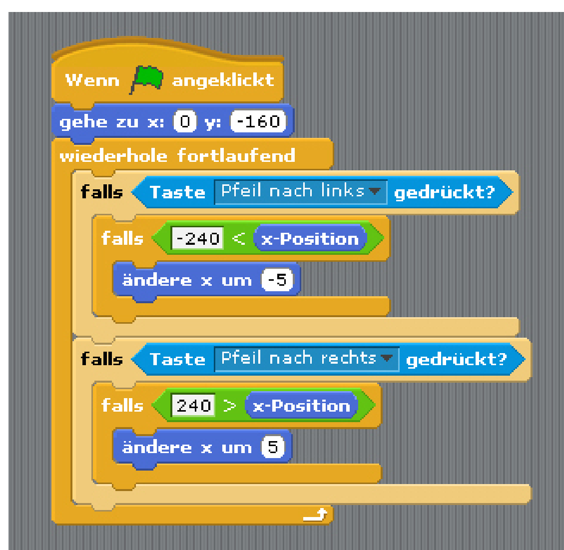


Abbildung 17: Skript von *Player*

Der Code in Abb. 17 S. 24 weist keine komplexeren Elemente auf und sollte daher relativ selbsterklärend sein. Zu Beginn wird der Player auf eine bestimmte Koordinate gesetzt. Anschließend wird in einer Dauerschleife geprüft, ob eine der Pfeiltasten gedrückt wurde. Sollte sich der Player dann noch im erlaubten Bereich, also innerhalb der Bühne befinden, bewegt er sich in die gewünschte Richtung.

Anschließend wird der Ball programmiert. Dieser muss viele Anforderungen erfüllen: Er darf den Boden nicht berühren und muss vom Player, der restlichen Wand und von den Bricks richtig abprallen. Aus diesem Grund wird für die x-Richtung und die Y-Richtung jeweils eine Variable erstellt, welche den Kosinus bzw. den Sinus der Richtung beinhalten. Multipliziert mit dem Tempo ergibt sich daraus die jeweilige Änderung von den Koordinaten X und Y.

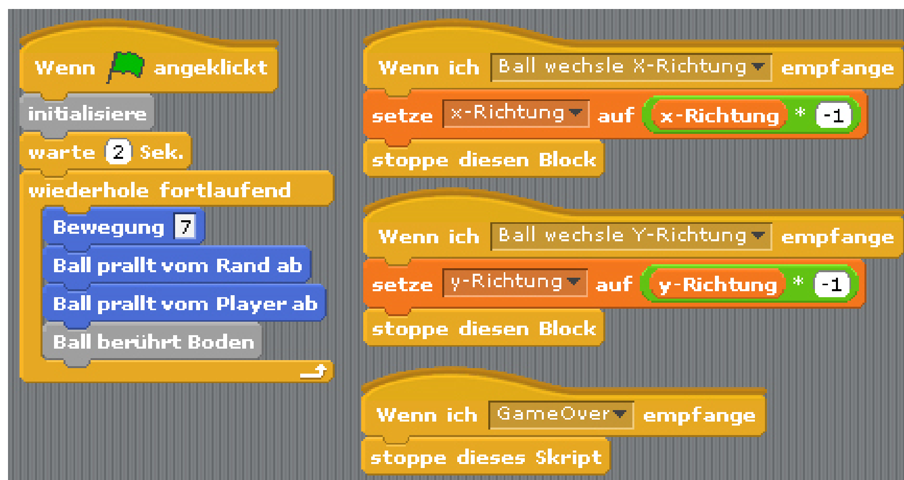


Abbildung 18: Skript von *Ball*

Der Übersicht halber sind die einzelnen Anforderungen an Ball in Methoden bzw. Blöcken zusammengefasst, die im Anschluss erläutert werden. Bei Spielstart wird der Ball initialisiert. Anschließend prüft eine Dauerschleife die einzelnen Anforderungen an den Ball.

Sobald der Ball eine Richtungsänderung empfängt, negiert er dementsprechend die x-Richtung bzw. die y-Richtung. Bei der Nachricht *GameOver* stoppt das Skript und somit auch die Bewegung des Balles.

Abbildung 19: Der Block *initialisiere*

Der Block *initialisiere* in Abb. 19: Als erstes wird der Ball auf einen bestimmten Punkt gesetzt und eine Richtung per Zufall innerhalb eines bestimmten Bereichs ermittelt. Dann wird die x-Richtung und y-Richtung gesetzt.

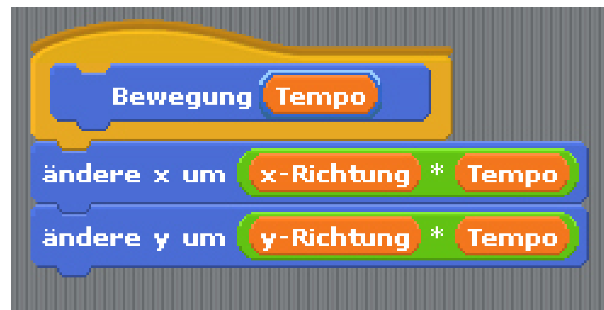


Abbildung 20: Der Block *Bewegung*

Der Block *Bewegung* in Abb. 20: Dem Block wird als Parameter das Tempo übergeben. Dieser wird mit der X-Richtung und der Y-Richtung multipliziert, die die konkreten Änderungen der Ortskoordinaten sind und damit die Bewegung darstellen.

Abbildung 21: Der Block *prallt vom Rand ab*

Der Block *prallt vom Rand ab* in Abb. 21: Wenn der Ball die Bühne aufgrund seiner Bewegung verlassen sollte, prallt er ab, in dem die entsprechende Bewegungsrichtung wie oben beschrieben umgekehrt wird.

Abbildung 22: Der Block *prallt vom Player ab*

Der Block *prallt vom Player ab* in Abb. 22: Falls der Ball den Player berühren sollte, wird die Y-Richtung umgekehrt.

Abbildung 23: Der Block *Ball berührt Boden*

Der Block *Ball berührt Boden* in Abb. 23: Sobald der Ball den Boden berührt, wird *GameOver* an alle gesendet, was von der Bühne empfangen wird.Das Spiel wird beendet.

Als nächstes Objekt wird *Brick* erstellt. Eine der Anforderungen an das Spiel ist es, es so dynamisch wie möglich zu realisieren, also beliebig viele *Bricks* generieren und anordnen zu können. Nun ist es nicht wie bei einer klassenbasieren Objektorientierung möglich, beliebig viele Objekte zu instanzieren, sondern man muss mit Klonen arbeiten. Die Idee ist also, einen *Brick* als Prototype zu erstellen, und von diesem aus beliebig viele *Bricks* zu klonen. Dabei ergeben sich allerdings gewisse Probleme. Wenn ein *Brick* abgeschossen wurde, muss er sich selbst löschen, der Prototype allerdings nicht. Im Falle eines *GameOver* wegen Bodenberührung des *Balls* sind aber noch *Bricks* auf der Bühne übrig. Wird das Spiel neu gestartet, bleiben diese „alten“ *Bricks* übrig und es werden wiederum neue erstellt. Durch die schon vorhandenen *Bricks* wird nun eine Kettenreaktion von Klonen ausgelöst. Daher ist es essentiell, dass alle *Bricks* nach Beendigung des Spiels wieder gelöscht werden. So werden alle Klone über eine Liste *Klone* verwaltet und von dieser aus im Falle eines *GameOver* gelöscht.

Der *Brick*, der als Prototyp dient, wird so gezeichnet, dass jeweils oben und unten eine Farbe ist, welche in diesem Spiel nur einmal vorkommt, genauso wie an den linken

und rechten Seiten. Dies hat den Vorteil, dass nach Kontakt mit dem Ball nicht mathematisch aufwendig ausgerechnet werden muss, an welcher Seite der Kontakt stattgefunden hat, um den Ball in die richtige Richtung abprallen zu lassen.

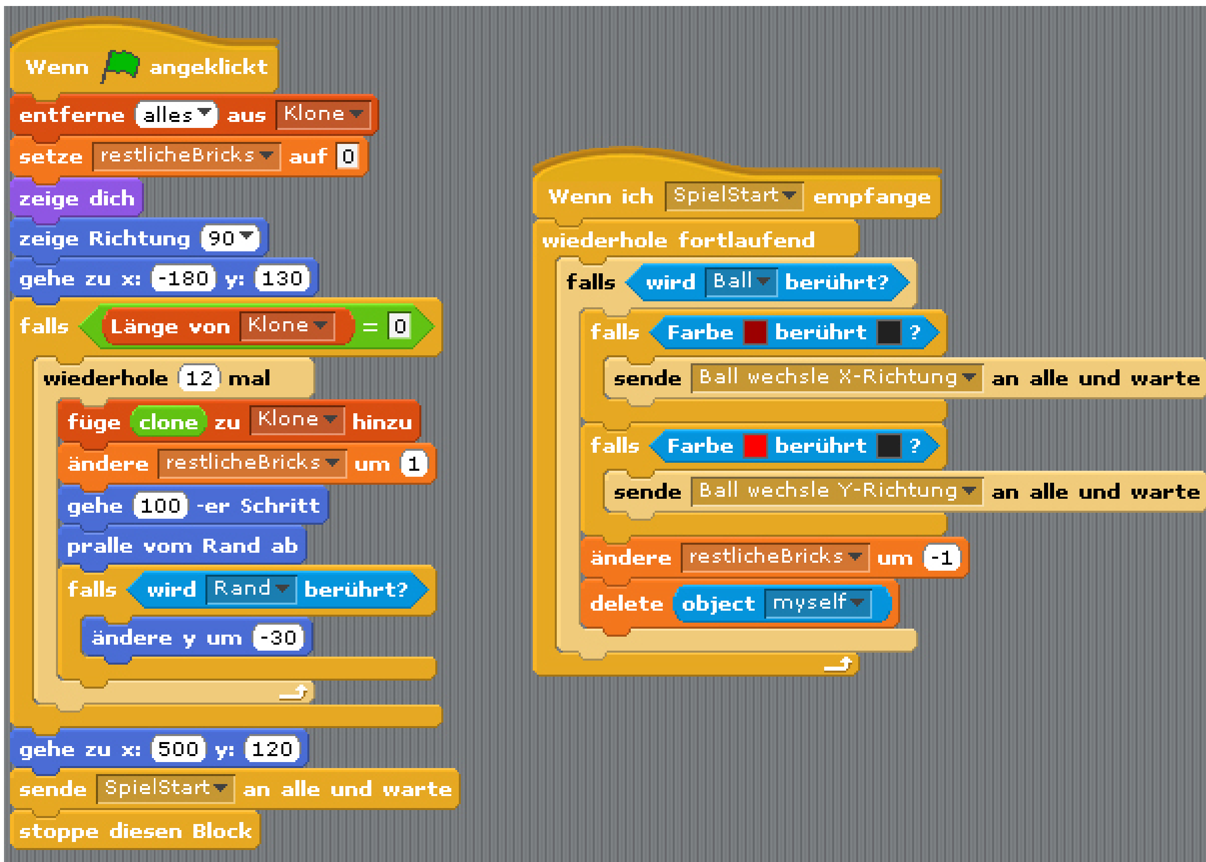


Abbildung 24: Skript von *Brick*

Zu Beginn des Spiels müssen die Klone erstellt werden. Zuerst wird die Liste zur Verwaltung der Klone geleert und die Variable *restlicheBricks* auf 0 gesetzt. Nachdem der Prototype auf eine bestimmte Stelle der Bühne gesetzt wurde, fängt der eigentliche Klonvorgang an. Es wird geprüft, ob die Liste *Klone* schon Elemente beinhaltet. So wird verhindert, dass schon erstellte Klone wiederum neue Klone erstellen.

In der Konfiguration der Schleife wird angegeben, wie viele *Bricks* erstellt werden sollen. Innerhalb der Schleife wird also ein Klon erstellt, indem er der Liste *Klone* zugefügt wird und die Anzahl der *Bricks* wird um 1 nach oben gesetzt. Ein erstellter Klon wird immer an die Position seines *Parent* gesetzt. Daher ändert der Prototyp anschließend seine Position auf der Bühne. Nachdem alle Klone erstellt worden sind, verlässt der Prototype die Bühne[8], um nicht selbst vom Ball getroffen zu werden und sendet Spiel-

[8]Um die zu ermöglichen, muss unter *Bearbeiten* → *Objekte dürfen die Bühne verlassen* ein Häckchen gesetzt sein

start an sich selbst. Damit werden alle *Bricks* aktiv.

Anschließend prüft jedes *Brick* dauerhaft, ob es Kontakt mit dem Ball hatte. Ist dies der Fall, wird über die Farben ermittelt, auf welcher Seite dies stattgefunden hat. Dann wird eine Nachricht abgesendet, welche vom Ball empfangen wird und signalisiert, dass der Ball seine Richtung dementsprechend ändern muss. Da der Ball den Brick abgeschossen hat, wird die Brickanzahl um 1 vermindert und das Brick löscht sich selbst.

Anschließend wird die Bühne erstellt, die recht einfach gehalten wird. Es ist nur darauf zu achten, dass auf dem Boden der Bühne ein Strich mit einer einzig vorkommenden Farbe ist, um dem Kontakt zwischen *Ball* und dem Boden zu prüfen[9]. Bei Berührung werden alle verbliebenen *Bricks*, außer der Prototype selbst, gelöscht und das Spiel beendet.

Abbildung 25: Skript von *Bühne*

Insgesamt gibt es 3 Hintergründe, einen für das laufende Spiel, einen, nachdem gewonnen wurde und einen für GameOver. Bei Spielstart findet ein Wechsel auf den Hintergrund statt, wie in Abb. 25 zu sehen ist. Anschließend gibt es wieder eine Dauerschleife, die prüft, ob alle *Bricks* abgeschossen wurden. In diesem Fall endet das Spiel, der Hintergrund wechselt zum Gewinnerbild und alle Skripte werden gestoppt.

Falls die Bühne *GameOver* mittels einer gesendeten Nachricht vom *Ball* empfängt, werden die verbliebenen geklonten *Bricks* gelöscht, der GameOver-Hintergrund kommt zum Vorschein und alle Skripte werden gestoppt.

[9]Diese Prüfung wird vom *Ball* durchgeführt

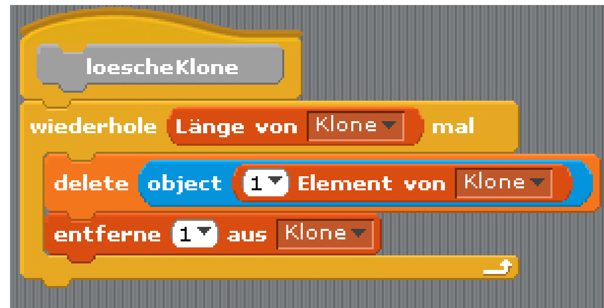


Abbildung 26: Der Block *loescheKlone*

Nun soll noch kurz der Block *loescheKlone* erläutert werden, welcher in Abb. 26 dargestellt ist. Über die Länge der Liste *Klone* wird jeweils das Objekt selbst, welches als erstes in dieser Liste gespeichert ist, gelöscht. Anschließend wird der Eintrag dieses Objekts aus der Liste an sich herausgelöscht. Durch diesen Vorgang rutscht jedes Element nach und nach an die erste Stelle, so dass am Ende alle Objekte, welche in der Liste gespeichert waren, gelöscht sind.

Es ist also mit den Mitteln von *BYOB* gut möglich, ein Spiel zu schreiben, allerdings wird es schnell unübersichtlich, sobald mehrere Objekte geklont werden. Das Debuggen in *BYOB* ist allerdings sehr anstrengend, da z.B. bei einem logischen Fehler der Klonvorgang sich verselbstständigen kann und anschließende mehrere hundert Objekte im Objektbereich zu finden sind. Das Programmieren dieses Spieles wäre für die Schule geeignet, wenn man sich auf eine feste Anzahl an *Bricks* einlassen würde. Allerdings erkennt man bei einer dynamischen Verwaltung der *Bricks* auch den Vorteil einer klassenbasierten Objektorientierung, da bei dieser die Verwaltung der *Bricks* sehr viel einfacher ist. Weiterhin wird auch deutlich, dass kaum ein Spiel ohne eine Mindestanforderung von Mathematik auskommt.

5 Fazit und Ausblick Lego NXT

Wie es sich zeigt, bieten *Scratch* und *BYOB* viele didaktische Vorteile, um den Einstieg in die Programmierung zu erleichtern.

> *Zu Beginn des Informatikunterrichts besitzen die meisten Schüler recht geringe Fähigkeiten in der Programmierung. Diese können sie nach eigenen Erfahrungen anhand von Scratch schnell und bleibend ausbauen [...] (Romeike, 2010, S. 47).*

So haben sogar Studenten der Informatik an Harvard ihre ersten Programmiererfahrungen mit *Scratch* gemacht, damit sie eine mentale Vorstellung beispielsweise von

Schleifen gewonnen und diese anschließend in einer anderen Pogrammiersprache umgesetzt haben (ebd.). So berichten Modrow et al. (2011), dass in Berkeley die Standard-Informatikvorlesung mit Hilfe von *BYOB* gehalten wird.

Allerdings sind *Scratch* und *BYOB* an bayerischen Gymnasien nur bedingt einsetzbar. Für die 7. Klasse erfolgt keine Einschränkung, *Scratch* kann den inhaltlichen Stoff dieser Jahrgangsstufe genauso gut verdeutlichen wie RobotKarol. Für die 8. Klasse bietet sich für ein ansprechendes Projekt mit den Schülerinnen und Schülern *Scratch* auch an. In der 10. Klasse erscheinen dann zum ersten Mal die Grenzen von *BYOB*. Aber obwohl *BYOB* eine prototypenbasierte Objektorientierung besitzt, leistet es gute Hilfestellung, um verschiedene inhaltliche Themen noch zu verdeutlichen, bevor auf eine Programmiersprache umgestiegen wird, die eine klassenbasierte Objektorientierung besitzt.

Der nächste Schritt in der visuellen Programmierung wäre dann, reale Objekte mittels eines Computers zu steuern. Und genau dies bietet Lego Mindstorms. Mit Hilfe von Lego Mindstorms kann man sich aus verschiedenen Bauteilen einen Roboter zusammensetzen und mit Hilfe der Software LegoNXT diesen Roboter programmieren. So wird auf der Website beschrieben:

> *With LEGO MINDSTORMS you can build and program robots that do what you want! With the contents in the set you get everything you need to build and program your very own intelligent LEGO robot, and make it perform loads of different operations. (Lego Mindstorms)*

Es werden beispielsweise ein NXT mikro-Computer (der das Gehirn des Systems bildet), 2 Berührungssensoren, ein Ultraschallsensor und vieles mehr mitgeliefert (vgl. ebd.). Diese Elemente können dann über die Software *NXT-G*, die mitgeliefert wird und für Windows und OSX verfügbar ist, angesprochen werden und Anweisungen erteilt werden. All diese Anweisungen werden ähnlich wie bei *Scratch* über eine visuelle Programmierung erteilt. Dies dürfte einen sehr guten Anreiz für Schüler bieten, sich weiterhin mit Informatik zu befassen.

6 Literaturverzeichnis

Gedruckte Quellen

Baumann (2009)
Baumann R.: *Sprechende Katzen und Zeichenschildkröte* In: LOG IN, 29. Jg., Heft 156, S. 51–58, 2009.

Modrow et al. (2011)
Modrow E., Mönig J., Strecker K.: *„Wozu JAVA?“* In: LOG IN, 31. Jg., Heft 168, S. 35–41, 2011.

Romeike (2010)
Romeike R.: *„Das bessere Werkzeug“* In: LOG IN, 30. Jg., Heft 163/164, S. 43–48, 2010.

Romeike, Wollenweber (2009)
Romeike R., Wollenweber M.: *Katzenfreunde in Bochum* In: LOG IN, 29. Jg., Heft 157/158, S. 4–7, 2009.

Stoll et al. (2008)
Stoll T., Thalheim K., Timmermann B.: *Die Katze im Computer.* In: LOG IN, 28. Jg., Heft 154/155, S. 81–90, 2008.

Online-Quellen

Harvey, Mönig (2011)
Harvey B., Mönig J.: *BYOB Reference Manual Version 3.1*, 2011, URL: `http://byob.berkeley.edu/BYOBManual.pdf`, Verifizierungsdatum 15.01.2012

Lego Mindstorms
Website der Firma Lego Mindstorms, Grasbrunn URL: `http://mindstorms.lego.com/en-us/history/default.aspx`, Verifizierungsdatum 15.01.2012

Lehrplan Bayern Jgst. 7
Lehrplan der 7. Jahrgangsstufe an bayerischen Gymnasien URL: `http://www.isb-gym8-lehrplan.de/contentserv/3.1.neu/g8.de/data/media/26418/Lehrplaene/Jgst_7.pdf`, Verifizierungsdatum 15.01.2012

Lehrplan Bayern Jgst. 9
Lehrplan der 9. Jahrgangsstufe an bayerischen Gymnasien URL: `http://www.isb-gym8-lehrplan.de/contentserv/3.1.neu/g8.de/data/media/26418/Lehrplaene/Jgst_9.pdf`, Verifizierungsdatum 15.01.2012

Lehrplan Bayern Jgst. 10
Lehrplan der 10. Jahrgangsstufe an bayerischen Gymnasien URL: http://www.isb-gym8-lehrplan.de/contentserv/3.1.neu/g8.de/data/media/26418/Lehrplaene/Jgst_10.pdf, Verifizierungsdatum 15.01.2012

Lehrplan Bayern Jgst. 11
Lehrplan der 11. Jahrgangsstufe an bayerischen Gymnasien URL: http://www.isb-gym8-lehrplan.de/contentserv/3.1.neu/g8.de/data/media/26418/Lehrplaene/Jgst_11_12.pdf, Verifizierungsdatum 15.01.2012

Monroy-Hernández, Resnick (2008)
Monroy-Hernández A., Resnick M.: *Empowering Kids to Create and Share Programmable Media*, 2008, URL: http://info.scratch.mit.edu/sites/infoscratch.media.mit.edu/files/file/interactions_acm_2007_monroy-hernandez_resnick.pdf, Verifizierungsdatum 15.01.2012

Monroy-Hernández, Hill (2010)
Monroy-Hernández A., Hill B.M.: *Cooperation and Attribution in an Online Community of Young Creators*, 2010, URL: http://info.scratch.mit.edu/sites/infoscratch.media.mit.edu/files/file/cscw2010poster2.pdf, Verifizierungsdatum 14.01.2012

Resnick et al. (2009)
Resnick M., Maloney J., Monroy- Hernández A., Rusk N., Eastmond E., Brennan K., Millner A., Rosenbaum E, Silver J., Silverman B, Kafai Y.: *Scratch: Programming for All*, 2009, URL: http://web.media.mit.edu/~mres/papers/Scratch-CACM-final.pdf, Verifizierungsdatum 15.01.2012

Scratch Reference Guide (2012)
Scratch Reference Guide, 2012, URL: http://info.scratch.mit.edu/sites/infoscratch.media.mit.edu/files/file/ScratchReferenceGuide14.pdf, Verifizierungsdatum 14.01.2012

Website Scratch
Website von Scratch, Cambridge URL: http://scratch.mit.edu/, Verifizierungsdatum 15.01.2012