Cube Boy

Spielidee

Jump and Run
Eine Figur bewegt sich durch eine Ansammlung von Hindernissen und sammelt Münzen auf.

Featured Game

Dieses Spiel hat ein interessantes Bewegungskonzept.

Die Formensprache ist sehr einfach geometrisch gehalten. Die Farbgebung ist homogen.
Damit muss der Spieler sich nicht auf jeder Seite neu orientieren, sondern findet sich sofort zurecht.

Die Bewegungsmuster sind einfach und intuitiv verständlich.

Die Musik passt zu dem Spiel und ist weder zu aggressiv noch zu getragen.

Umsetzung der Bewegung

Bewegung links, rechts

Hier wird nicht die Figur bewegt, sondern die Welt bewegt sich. Will 'cube boy' nach rechts laufen, dann wird 'scrollX' kleiner und damit nach links gesetzt,

Die Hindernisse setzen ihre Position nach dem Muster

Wenn 'cume boy' also nach rechts läuft, wird scrollX kleiner. Damit verschiebt sich das Hindernis nach links: es sieht aus als ob 'cube boy' läuft.
Die '480*1', die zur Position dazugezählt werden, sind die die Verschiebung des Hindernisses auf die verschiedenen Seiten. 480 ist die Fensterbreite und die 1 entspricht der ersten Seite rechts vom Startbildschirm.

Bewegung oben, unten

'cube boy' fällt immer soweit, bis er den dunkelgrünen Untergrund berührt.

Das 'Warte' bewirkt, dass die Fallbewegung etwas gebremst wird.
Das bedeutet aber auch, dass jeder Bildschirm, jedes Hindernis diese Farbe aufweisen muss.

'cube boy' kann springen.

Hier gibt es eine Bedingung: springen darf man nur vom Boden aus.

Aufgaben

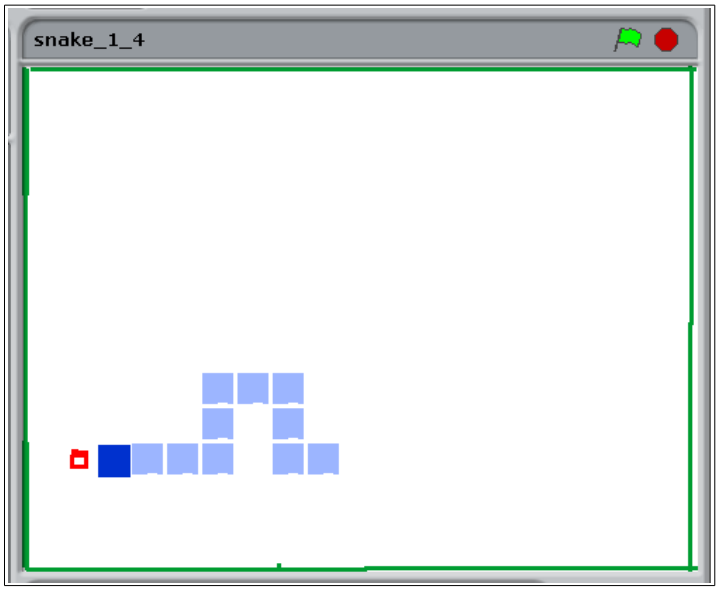
▶	Das Spiel in scratch2 (hier ist ein PC erforderlich) durchspielen. Scratch/featured_game/cube_boy/Cube Boy's Adventure!.sb2 http://scratch.mit.edu/projects/11790340/
▶ ▶	Die Grundidee des Spiels ist in scratch1.4 verfügbar. scratch\featured_game\cube_boy/hugo.sb Das Spiel in scratch laden.
▶	Bewegung links, rechts ausprobieren. Springen ausprobieren. Warum hoppelt die Figur die Rampe von rechts nach links herunter, aber von links nach rechts muss man hochspringen ? Was wäre hier eine bessere Lösung ?
▶	Baue einen neuen Bildschirm in das Spiel ein.

Zusatzaufgaben

▶	Das Spiel hat im Original eine Melodie hinterlegt. Die 'hugo'-Variante hat zwei Melodien. Wenn die Musik-Variable kleiner 10 ist, dann ein hiphop-Thema, sonst eine eher mittelalterliche Variante. Auch die Lautstärke ist unterschiedlich. Denke Dir zu jeder Melodie eine kurze Geschichte aus (wer ist 'hugo', warum ist er unterwegs und welche Abenteuer muss er bestehen? Zwei, drei Sätze reichen hier).
▶	Wenn man die Seiten wechselt, dann sieht man noch einen Rest der Hindernisse der vorherigen Seite. Wie kann man das verhindern ?

▶ ▶	Wie könnte eine Spezial-Spielekonsole für dieses Spiel aussehen ?

Snake und UML-Verknotungen



Spielidee

Eine Schlange bewegt sich in x,y-Richtung. Wenn das rote Element getroffen wird, dann wird die Schlange um ein Element länger.
Die Schlange darf sich selber oder den Rand nicht treffen; passiert das, so ist das Spiel zu Ende.

Featured Game

Das Spiel kann mit relativ wenigen Elementen in Scratch umgesetzt werden.

Es zeigt

- wie Listen eingesetzt werden
- scratch-spezifische Elemente wie 'Stempel' zum Einsatz kommen
- und gibt Anlass zur Erwähnung von UML zur Beschreibung der Abläufe.

Umsetzung

Die Schlange wird bewegt, in dem sich der Kopf 'head' um eine Stelle weiterbewegt.

Dann wird ein Abdruck 'stamp' an die Stelle bewegt, wo vorher der Kopf war, und ein Abdruck hinterlassen. Der 'stamp' wird dann wieder unsichtbar.

Und zuletzt wird das Ende der Schlange ausradiert. Ein Radierer 'wipe', einfach ein weisses Element, wird über das letzte Element bewegt und ein Abdruck hinterlassen. Der 'wipe' wird dann wieder unsichtbar. Dieses Element hat auch ein schwarzes Kostüm: bei der Entwicklung wurde das benutzt, um herauszufinden wo es sich befindet.

Damit das Ende der Schlange gelöscht werden kann, hat das Spiel eine zwei Listenvariablen für die x- und y-Koordinaten der Elemente.

Bei einer Länge von 3 ist die Liste wie folgt belegt:

Liste[1] ist das letzte Element

Liste [2] das mittlere Element
der Kopf liegt nicht in der Liste

Der Kopf bewegt sich: die Position des Kopf wird in die Liste angefügt.

Liste[1] ist das letzte Element
Liste [2] das mittlere Element
Liste [3] Position des Kopfes vor der Bewegung
neue Position des Kopfes

Jetzt muss ein Abdruck erzeugt werden 'stamp', es wird die letzte Stelle der Lsite verwendet.

Liste[1] ist das letzte Element
Liste [2] das mittlere Element
Liste [3] Position des Kopfes vor der Bewegung → Abdruck
neue Position des Kopfes

Und nun wird das Ende der Schlange radiert; 'wipe' benutzt das erste Element

Liste[1] Löschabdruck
Liste [2] das mittlere Element
Liste [3] Abdruck
neue Position des Kopfes

Und zuletzt wird das erste Element aus den Listen gelöscht. Die anderen Elemente rutschen zurück, und das Spiel kann wieder von vorne losgehen.

Liste[1] ist das letzte Element
Liste [2] das mittlere Element
der Kopf liegt nicht in der Liste

Wenn die Schlange das rote Element futtert, dann wird das Löschen des Endes ausgesetzt, damit verlängert sich die Schlange.

Das Spiel wird beendet, wenn der 'head' (dunkelblau) den Rand (grün) oder den Körper 'stamp' (hellblau) berührt. Das ist zwar sehr einfach zu programmieren, aber wenn das Spiel neu gestartet wird und schon Elemente auf der Bühne liegen, dann kann 'head' schon ein übriggebliebenes 'stamp' berühren und das Spiel ist schon zu Ende bevor es angefangen hat. Deshalb übernimmt hier der Hintergrund 'Bühne' die Steuerung der Startsequenz.

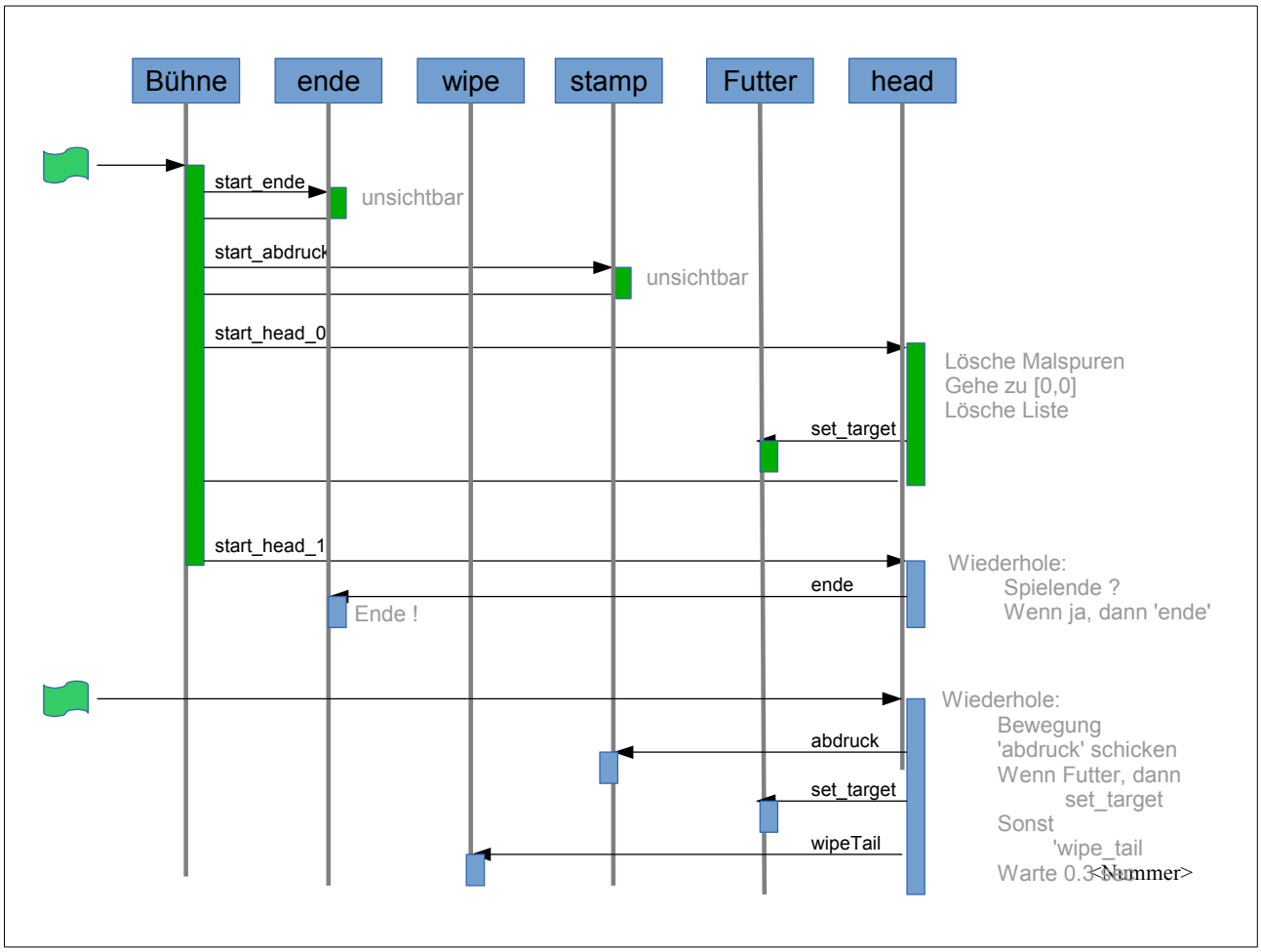
Anmerkung

Um das Verhalten von Software zu beschreiben wurden viele Methoden entwickelt: Flussdiagramme, Nassi-Shneiderman-Diagramme, und eine Vielzahl von anderen Methoden.
Heute hat vor allem die von Grady Booch, Ivar Jacobson und James Rumbaugh entwickelte Unified Modeling Language 'UML' eine herausragende Rolle.

Diese Beschreibungsmethode ist deswegen so wichtig, da sich Programmentwickler damit wie in einer Geheimsprache unterhalten können. Die Funktion und das Zusammenspiel der Programmobjekte kann einfach und übersichtlich dargestellt werden, ohne sich durch die vielen Details ablenken zu lassen.

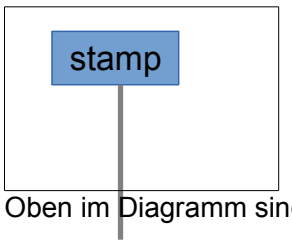
Das folgende Diagramm ist ein UML 'Interaktionsdiagramm' oder 'Sequenzdiagramm', das den Ablauf

des 'snake'-Spieles beschreibt.

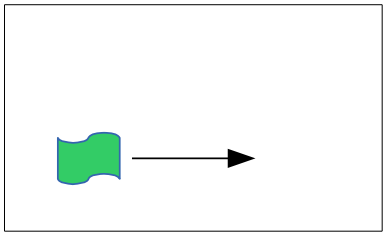


Zur Verdeutlichung werden hier im Diagramm dunkelgrüne Balken für Initialisierungen verwendet.
Die Steuerung über die Pfeiltasten ist nicht eingezeichnet.

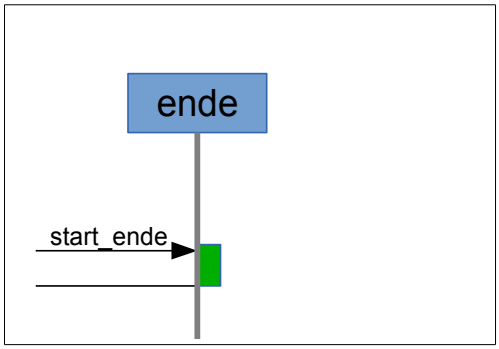
Die einzelnen Elemente der Grafik sind im Detail:



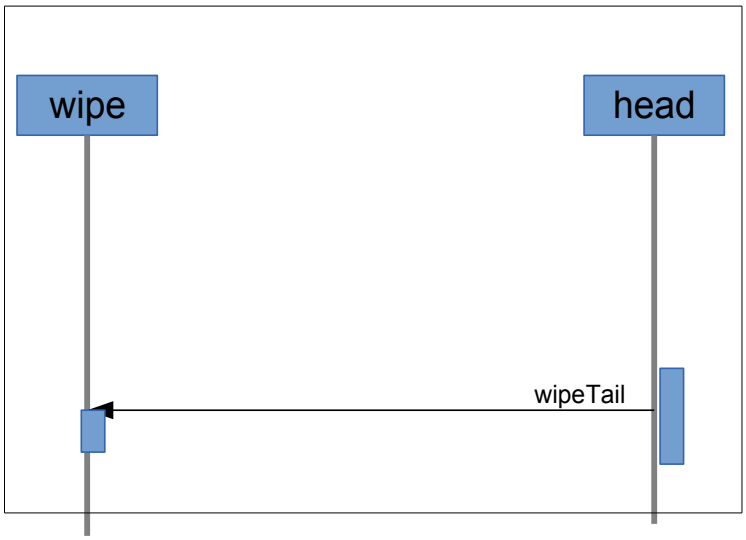
Oben im Diagramm sind die verschiedenen Objekte angeordnet.



Signale von aussen, wie z.B. die Flagge, werden mit Pfeilen von aussen dargestellt.



Wenn ein Objekt etwas arbeitet, also ein Skript ausführt, dann wird diese Aktion als Balken dargestellt, hier grün gezeichnet.
In diesem Beispiel wird die Aktion über das Scratch-Element „sende 'start_ende' und warte“ gestartet. Dieser Aufruf ist 'synchron' oder 'im Takt' oder 'der Reihe nach'. Das aufrufende Skript wartet geduldig so lange, bis hier das 'ende'-Objekt fertig gearbeitet hat.
Synchrone Aufrufe haben unten eine zweite Linie, die zum Aufrufer zurück geht.



<Nummer>

Hier schickt 'head' das Signal 'wipeTail' an 'wipe'. Das ist asynchron: das Objekt 'head' wartet nicht, bis 'wipe' mit seiner Aufgabe fertig ist, sondern macht mit seinen Aufgaben sofort weiter.
Asynchrone Aufrufe haben nur einen start-Pfeil, keine zurücklaufende Linie.

Aufgaben

▶	Das Spiel in scratch1.4 anspielen.
▶	Warum übernimmt die Bühne hier die Steuerung des Startablaufes?
▶ ▶	Wie könnte hier eine Spezial-Spielekonsole für das Spiel aussehen ?
▶	.

Zusatzaufgaben

<Nummer>

▶	Die Grafiken für das Spiel sind schlecht gezeichnet. Korrigiere die Rechtecke, zeichne das Futter als Kreis, Apfel, oder ähnlich, korrigiere den Rand. Kopiere das Spiel, so dass es in die Materialien für den nächsten Kurs aufgenommen werden kann.
▶	Was wäre eine geeignete Melodie für das Spiel ?
▶	Erweitere das Spiel um eine Start-Meldung (wie heisst das Spiel, wie bewegt man sich, was ist das Ziel)

Lokomotive, Interpreter

Die Aufgabenstellung war, dass ein Fahrzeug flexibel einen Kurs absolviert. Allerdings sollte der Kurs einigermassen flexibel einstellbar sein und es sollen Verzweigungen möglich sein.
Um das zu erzielen wurde der Kurs in Vektoren zerlegt. Diese Vektoren werden in einer Tabelle abgespeichert. So kann durch einfache Änderung der Tabelle ein neuer Kurs erstellt werden.

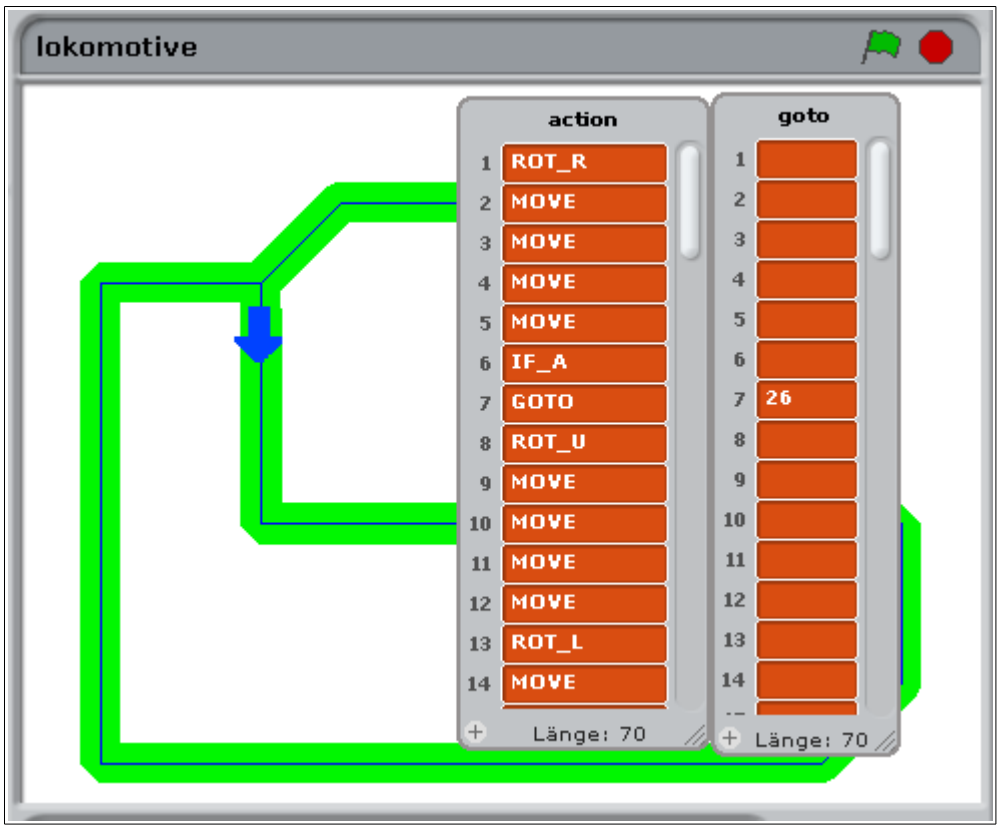
Das Programm liest und interpretiert die Anweisungen zur Laufzeit.
Die Befehle sind

Bewegung	**MOVE**	Bewegt sich einen Schritt in die eingestellte Richtung
Drehrichtung	**ROT_L**	Setzt die Bewegungsrichtung nach Links (Left)
	ROT_D	nach unten (Down)
	ROT_U	nach oben (Up)
	ROT_R	nach rechts (Right)
	ROT_-1-1	nach links, unten
Ablauf	**GOTO**	Spinge an eine bestimmte Stelle, die den nächsten Befehl enthält.
Verzweigung	**IF_A**	wertet die Variable 'condition_A' aus. Wenn '1', dann wird der nächste Befehl überspringen. Wenn '0', dann wird der nächste Befehl ausgeführt, der dann einen GOTO Befehl enthalten kann.
	IF_B	wertet die Variable 'condition_B' aus.

Alle Befehle (bis auf GOTO) senden ein Signal 'an alle und warte'. Damit kann die Implementierung relativ einfach verändert werden.
Die IF_A und IF_B sind Sonderfälle, da weitere Bedingungen im Code des Interpreters angepasst werden müssen.

Die Daten sind in zwei Tabellen organisiert, die Befehle in 'action' und die Sprungziele 'goto'.

Beispiel:



Das 'Fahrzeug' zieht eine Zeichenspur 'pen down'. Mit dieser Spur kann die Fahrbahn nachgezeichnet werden.