

Blockprogrammierung mit **Moenagade**



Kurs Unterlagen
für die Klasse 5G

Inhaltsverzeichnis

1. Einführung in die Blockprogrammierung.....	4
2. Moenagade: das „Warum“	5
3. Konzepte.....	6
4. Kompatibilität.....	8
5. Bedienelemente.....	9
6. Arbeiten mit den Blöcken.....	10
7. Tipps & Tricks.....	12
8. Aufgaben rund um die Welt.....	13
8.1. Hello world!.....	13
8.2. Welt wechsele dich	14
8.3. Welt mit Hund.....	15
8.4. Welt mit Hund 2.....	16
8.5. Der Hund läuft weg.....	17
8.6. Links und rechts.....	18
8.7. Dein Welt, deine Figur!.....	18
8.8. Gefangener Hunde.....	19
8.9. Achtung Hund!.....	19
8.10. Das Auto rollt von selbst.....	20
8.11. Neues Auto.....	21
8.12. Hundelimits.....	22
8.13. Auto zählen.....	23
8.14. Speed up!.....	24

1. Einführung in die Blockprogrammierung

Programmieren bedeutet, einem Computer ganz genau zu erklären, was er tun soll. Dafür müssen wir Anweisungen formulieren und sie in einer bestimmten Reihenfolge ausführen.

Am Anfang kann das sehr schwierig und angsteinflößend sein: Programmiersprachen wie Python oder Java haben eine genaue Schreibweise. Ein fehlendes Zeichen, ein falscher Buchstabe oder eine Klammer an der falschen Stelle kann bereits dazu führen, dass ein Programm nicht funktioniert.



Hier kommt die Blockprogrammierung ins Spiel.

Bei der Blockprogrammierung werden Programme nicht durch das Schreiben von Text erstellt. Stattdessen setzt man farbige Programmierblöcke zusammen. Wie genau das funktioniert, wird in den kommenden Kapiteln genau erklärt.

Jeder Block steht für eine bestimmte Anweisung. Zum Beispiel kann ein Block sagen:

- „Führe diese Anweisung aus“
- „Wiederhole etwas“
- „Wenn etwas passiert, dann ...“
- „Setze eine Variable auf einen bestimmten Wert“
- „Spiele einen Ton ab“
- „Bewege eine Figur“

Die Blöcke können miteinander verbunden werden und bilden dadurch ein vollständiges Programm. Dieses Skript erklärt Blockprogrammierung an Hand des Programms „Moenagade“.

Wenn man eine neue Programmiersprache lernt, muss man gleichzeitig zwei Dinge lernen: Wie man programmiert und wie man die Programmiersprache richtig schreibt. Für Anfänger kann das aber schnell zu viel werden.

Bei der Blockprogrammierung muss man sich zunächst weniger um die genaue Schreibweise kümmern sondern kann sich stattdessen auf das eigentliche Programmieren und logische Denken konzentrieren.

2. Moenagade: das „Warum“ ...

„Moenagade“ ist das Akronym für „**m**ouse **e**nabled **g**ame **d**evelopment“. Wenn du Probleme mit der Aussprache hast, denke einfach an den Kellner aus der berühmten Zeichentrickserie Die Simpsons.

Visuelle Programmiersprachen¹, manchmal auch „Block-Programmierungswerkzeuge“ genannt, werden heutzutage immer beliebter. Ihr Hauptvorteil besteht darin, dass diese Anwendungen den meist komplexen Quellcode automatisch erzeugen und dem Benutzer gleichzeitig eine einfach zu bedienende grafische Oberfläche zur Verfügung stellen, mit der sich schnell ansprechend aussehende Anwendungen erstellen lassen.



Es gibt bereits eine Reihe sehr guter Werkzeuge, aber keines davon entsprach wirklich den Anforderungen für den Unterricht der Klasse 5G:

- ein einfach zu bedienendes Werkzeug für Anfänger,
- das die visuelle Vermittlung von Kontrollstrukturen ermöglicht,
- das letztendlich zu einer voll funktionsfähigen Anwendung führt,
- das die Motivation der Schüler auf einem hohen Niveau hält und
- einen einfachen Übergang von „den Blöcken“ zum „Code“ bietet.

In der Tat erzeugt Moenagade im Hintergrund Java Code, so wie er in höheren Klassen (CI, GIN, GIG, ...) von den Schüler „per Hand“ geschrieben wird. Wer also Lust und das nötige Know-How hat, kann den eigentlichen Programmcode des erzeugten Spiels einsehen.

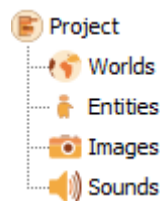
Das Programm Moenagade kannst du hier runter laden:

<https://moenagade.fisch.lu/>

¹ https://en.wikipedia.org/wiki/Visual_programming_language

3. Konzepte

Um mit Moenagade arbeiten zu können, ist es hilfreich, die Konzepte zu verstehen, auf denen das Programm basiert. Tatsächlich basiert alles auf den folgenden fünf Elementen:



1. Projekt [project]

Da Moenagade seine Dateien in Projekten organisiert, steht das Projektelement hier für die Anwendung. Es verfügt über eine zugehörige Block-Arbeitsfläche, auf der anwendungsbezogene Blöcke platziert werden können. Im generierten Java-Code entspricht es dem Hauptfenster.

Klicke auf „Projekt“, um dessen Blockcode zu bearbeiten.

2. Welten [worlds]

Ein Projekt kann mehrere Welten enthalten. Du kannst sie dir auch als „Bildschirme“ vorstellen. Innerhalb jeder Welt können Einheiten platziert werden. Jede Welt verfügt ebenfalls über eine zugehörige Block-Arbeitsfläche. Bei der Codegenerierung wird sie in eine Java-Klasse übersetzt.

Du kannst Welten hinzufügen und auf sie klicken, um den jeweiligen Code zu bearbeiten.

3. Einheiten [entities]

Einheiten sind Elemente, die innerhalb von Welten platziert werden können. Sie können entweder statische oder bewegliche Elemente darstellen. Jede Einheit verfügt über eine zugehörige Block-Arbeitsfläche und wird bei der Codegenerierung ebenfalls in eine Java-Klasse übersetzt.

Du kannst Einheiten hinzufügen und auf sie klicken, um den jeweiligen Code zu bearbeiten.

4. Bilder [images]

Bilder sind statische Ressourcen, die im Projekt gespeichert werden und entweder in eine Welt oder in eine Entität geladen werden können.

Klicke auf ein Bild, um es anzuzeigen.

5. Sounds

Sounds sind statische Ressourcen, genau wie Bilder.

Klicke auf einen Sound und anschließend in den Programmierbereich, um ihn anzuhören.

4. Kompatibilität

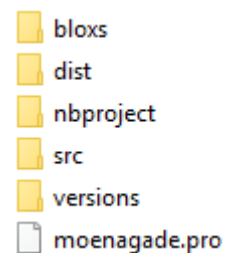
Wie bereits in der Einleitung erwähnt, wurde Moenagade entwickelt, um Schülern einen einfachen Einstieg in die Java-Programmierung zu ermöglichen und dabei den Übergang zum manuellen Schreiben von Quellcode so einfach wie möglich zu halten.

Aus diesem Grund verwendet Moenagade dieselbe Projektstruktur wie NetBeans² und Unimozer³. Moenagade Projekte können in beiden genannten Anwendungen geöffnet werden.



Achtung: Bitte beachte, dass die Codegenerierung nur in eine Richtung funktioniert: von den Blöcken zum Java-Quellcode. Sobald du den Quellcode eines Projekts in einem anderen Editor verändert hast, werden **alle** deine Codeänderungen zerstört, wenn du das Projekt erneut in Moenagade öffnest!

Beim Speichern enthält ein Moenagade-Projekt die folgenden Dateien und Verzeichnisse:



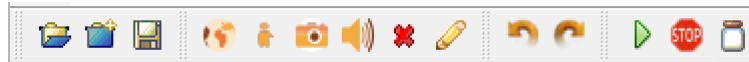
- **bloxs:** Dieses Verzeichnis ist spezifisch für Moenagade und enthält alle projektbezogenen Dateien, also die statischen Bild- und Soundressourcen sowie die blockbasiert programmierten Elemente.
- **dist:** Dieses Verzeichnis wird beim Erstellen eines Projekts angelegt. Es enthält anschließend die ausführbare JAR-Datei, die als eigenständige Anwendung ausgeführt werden kann.
- **nbproject:** Dieses Verzeichnis und sein Inhalt werden benötigt, um Projekte in NetBeans zu öffnen.
- **src:** Dieses Verzeichnis enthält den generierten Quellcode.
- **versions:** Bei jedem Speichern des Projekts wird eine Sicherungskopie des Quellcodes und des Blockcodes in diesem Verzeichnis archiviert.
- **moenagade.pro:** Dies ist eine Markierungsdatei, anhand derer Moenagade seine eigenen Projekte erkennt.

² <https://netbeans.org/>

³ <http://unimozer.fisch.lu/>

5. Bedienelemente

Die am häufigsten verwendeten Bedienelemente befinden sich als Schaltflächen in der Symbolleiste, sind aber auch im Anwendungsmenü vorhanden.



Die erste Gruppe umfasst die bekannten Dateioperationen:

- Ein vorhandenes Projekt öffnen.
- Ein neues Projekt erstellen.
- Das aktuelle Projekt speichern.

Die nächste Gruppe bezieht sich auf das Projekt:

- Eine neue Welt zum Projekt hinzufügen.
- Eine neue Entität zum Projekt hinzufügen.
- Eine neue Bildressource zum Projekt hinzufügen.
- Eine neue Soundressource zum Projekt hinzufügen.
- Das angewählte Element löschen.
- Das angewählte Element umbenennen.

Die nächsten beiden Funktionen gelten nur für das Blockfenster und stellen die Funktionen „Rückgängig“ und „Wiederholen“ dar. Beide sind klassenabhängig, was bedeutet, dass die Rückgängig-Zustände gespeichert bleiben, selbst wenn du den Inhalt des Blockeditors wechselst, beispielsweise indem du auf eine andere Entität klickst.

Die letzte Gruppe enthält folgende Befehle:

- Die Anwendung starten.
- Die Anwendung unterbrechen.
- Eine ausführbare Java-Datei (eine JAR-Datei) erstellen. Sie befindet sich im Verzeichnis „dist“ des Projekts.

6. Arbeiten mit den Blöcken

Zur einfacheren Verwendung sind die vordefinierten Blöcke in verschiedene Kategorien eingeteilt:

Strukturen, Werte, Logik, Ereignisse, Methoden, Eingabe, Mathematik, Variablen, Attribute, Sound, Objekt, Entität, Welt, Grafisch, Anwendung, Timer, Maus und Debug.

Structures	Values	Logical
Events	Methods	Input
Math	Variables	Attributes
Sound	Object	Entity
World	Graphical	Application
Timer	Mouse	Debug

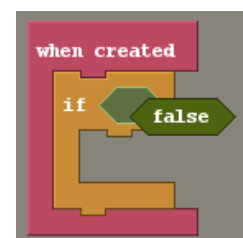
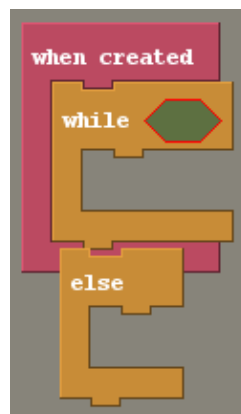
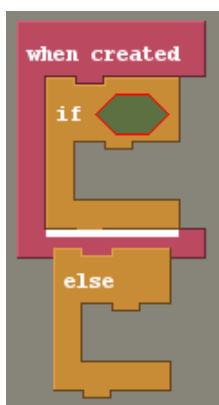
Blöcke können sich auf die Anwendung, eine Welt oder eine Entität beziehen. Das bedeutet, dass ein bestimmter Block abhängig von dem Element, das du gerade im Blockeditor bearbeitest, beim Auswählen einer Kategorie angezeigt werden kann oder nicht.

Jeder Block kann Folgendes besitzen:

- einen oberen Anschluss, was bedeutet, dass er mit einem anderen Block verbunden werden muss;
- einen unteren Anschluss, was bedeutet, dass ein anderer Block daran angeschlossen werden kann;
- einen Anschluss im Inneren, was bedeutet, dass andere Blöcke darin platziert werden können.

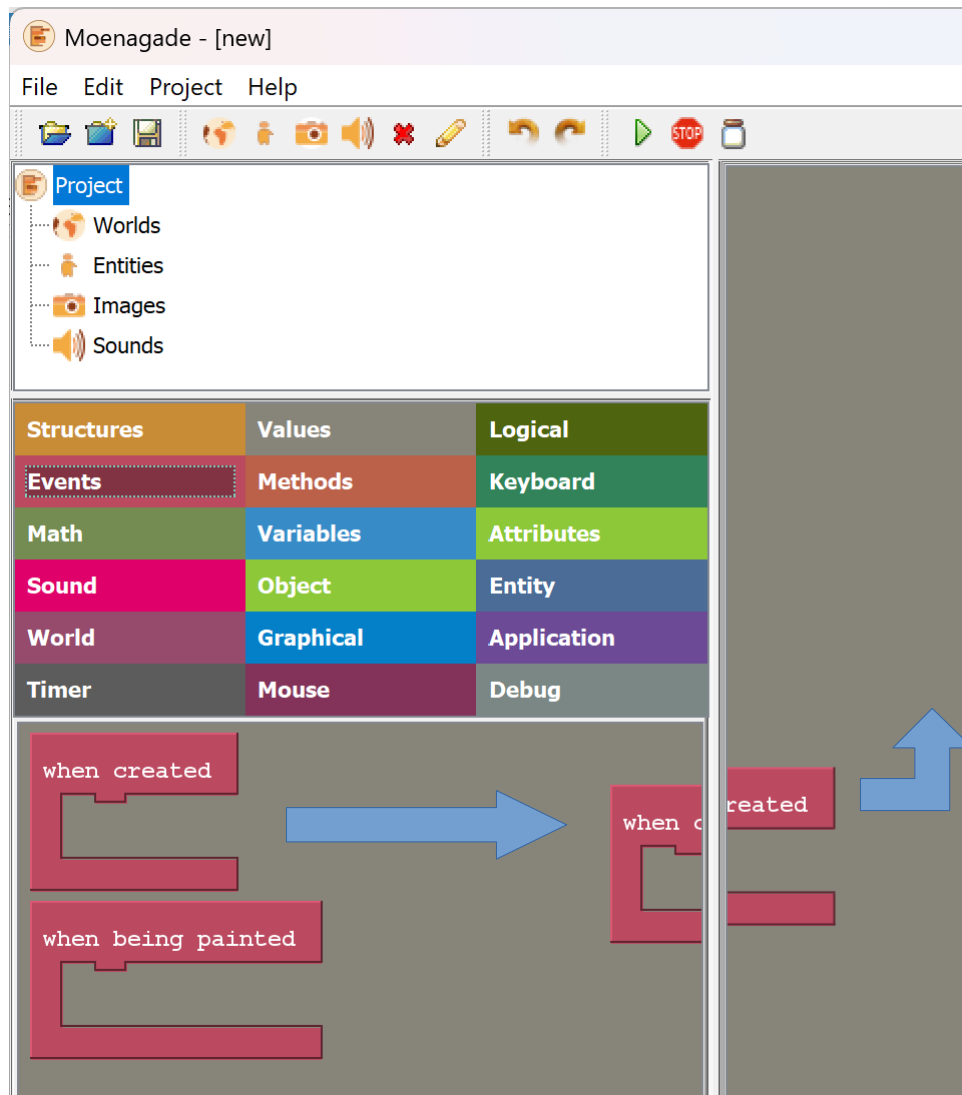
Beim Verbinden von Blöcken muss sich die obere linke Ecke des bewegten Elements innerhalb des Einrastbereichs des Blocks befinden, mit dem es verbunden werden soll. Dies wird durch einen weißen Balken angezeigt. Wenn kein weißer Balken erscheint, kann der verschobene Block an dieser Stelle nicht angeschlossen werden.

Entsprechend wird der Rand des Bereichs, in den ein Wert oder Ausdruck eingefügt werden kann, grün, wenn der verschobene Block hineinpasst, und bleibt rot, wenn dies nicht der Fall ist. Das sieht dann z.B. so aus:



Wie genau die einzelnen Blöcke funktionieren und wo du sie findest wirst du beim Durcharbeiten der einzelnen Aufgaben erlernen. Was du aber im Voraus wissen musst, ist folgendes:

- Um einen **neuen Block einzufügen**, klicke mit der Maus auf einen Block unten links und verschiebe ihn einfach in die Programmierfläche.



- Um einen Block wieder aus der Programmierfläche zu **entfernen**, also um ihn zu löschen, ziehe ihn einfach mit der Maus in die untere linke Ecke. Alle Blöcke welche sich in oder an dem ausgewählten Block befinden, werden mit gelöscht.
- Um eine bestehenden Block in der Programmierfläche zu **kopieren**, drücke die `Ctrl`-Taste gedrückt bevor du den Block anklickst. Es wird dann eine Kopie erstellt, welche du mit der Maus einfach an eine andere Stelle verschieben kannst.

7. Tipps & Tricks

Wenn du beginnst, mit Moenagade Spiele zu programmieren, kann es hilfreich sein, einige grundlegende Schritte zu befolgen:

- Das von der Moenagade-Anwendung erzeugte Fenster kann nicht in seiner Größe verändert werden, passt sich aber an die Größe des geladenen Hintergrundbildes an. Deshalb wird empfohlen, zunächst nach einem geeigneten Hintergrundbild zu suchen oder eines zu entwerfen und erst in einem zweiten Schritt die darauf zu platzierenden Elemente sowie gegebenenfalls weitere Hintergrundbilder für andere Welten auszuwählen.
- PNG-Bilder mit Transparenz werden vollständig unterstützt. Die meisten Einheiten fügen sich besser in ihre Umgebung ein, wenn sie einen transparenten Hintergrund besitzen. Wähle daher entweder ein entsprechendes Bild aus oder erstelle selbst eines.
- Abhängig vom Spiel, das du entwickelst, kann es sinnvoll sein, ein Storyboard zu erstellen, das beschreibt, in welcher Reihenfolge oder abhängig von welchen Aktionen die Welten geladen werden. Beim Debuggen oder Testen deiner Anwendung ist dies möglicherweise nicht besonders hilfreich. Denke daher daran, dass du jederzeit auf „Projekt“ klicken und die Startwelt ändern kannst.
- **Vergiss nicht, dass du für jedes Projekt folgende Sachen programmieren musst:**
 - 1. Im Projekt selbst, also beim Öffnen des Spiels, muss du die erste Welt laden.**
 - 2. Für jede deiner Welten musst du eine Grafik laden.**
 - 3. Für jede deiner Einheiten muss du auch eine Grafik laden.**

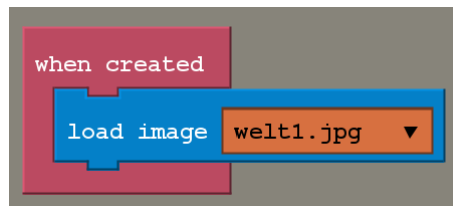
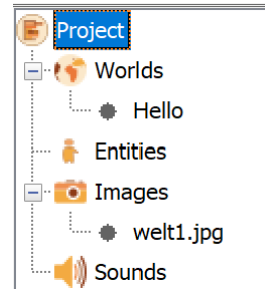
8. Aufgaben rund um die Welt

Zum Lösen folgender Aufgaben werden dir diverse Ressourcen (Bilder, Sounds, ...), welche du auf der Seite <https://5g.fisch.lu> herunterladen kannst. Speichere am besten alle Dateien, welche du benötigst, in deinem Verzeichnis ab.

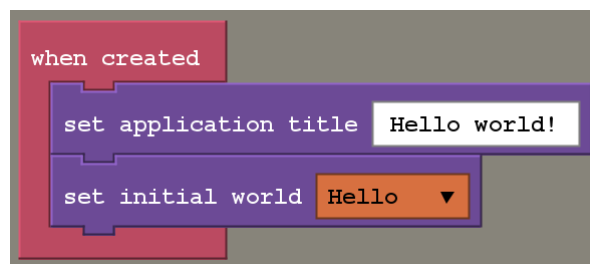
8.1. Hello world!

Jede gute Programmiersprache braucht eine „Hello world“ Anwendung. Aus diesem Grund ist dies auch die erste Aufgabe.

1. Erstelle ein neues Projekt.
2. Lade das Bild `welt1.jpg`
3. Erstelle die Welt `Hello` so dass darin das Bild `welt1.jpg` geladen wird. Dafür musst du folgende Blockprogrammierung innerhalb der Welt `Hello` einfügen:



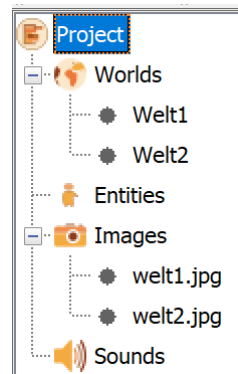
- Klicke die Welt `Hello` an.
 - Den Block `when created` findest du bei `Events`.
 - Den Block `load image` findest du bei `Graphical`.
 - Klick auf den kleinen schwarzen Pfeil nach unten und dann auf den Namen des Bildes das zu laden ist.
4. Nun muss das Projekt selbst noch wie folgt programmiert werden. Den Block zum Setzen der Welt findest du bei `World`, den anderen bei `Application`.



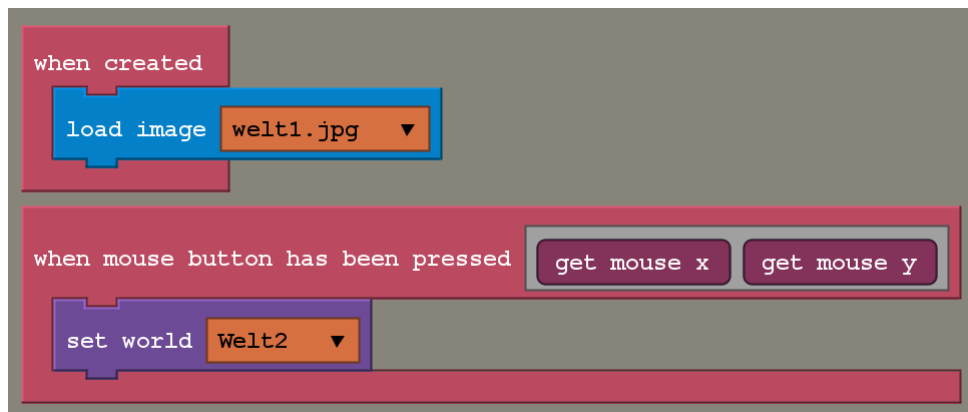
8.2. **Welt wechsle dich ...**

Als nächstes sollst du ein Projekt erstellen, welches zwei Welten beinhaltet: Welt1 und Welt2.

- Beim starten wird Welt1 geladen.
- Klick man irgendwo in die Welt1, wird Welt2 geladen.
- Klick man irgendwo in die Welt2 wird Welt1 geladen.



Die Programmierblöcke der Welt1 sehen wie folgt aus. Finde selbst heraus wie jene der Welt2 aussehen müssen.



8.3. Welt mit Hund

Erstelle eine Welt mit einem Hund. Das soll dann so aussehen:

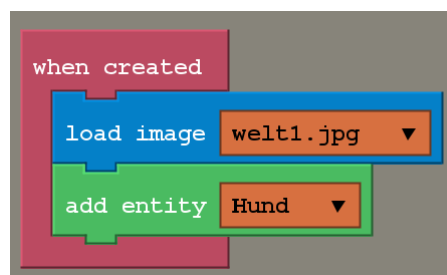
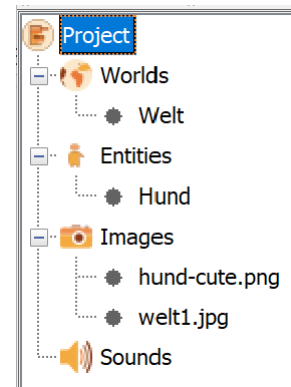


Hierfür benötigst du die Bilder `welt1.jpg` und `hund-cute.png`. Das Bild des Hundes ist im PNG Format, welches einen transparenten Hintergrund erlaubt, was für die Darstellung hier sehr wichtig ist, da sonst ein schwarzer oder weißer Rahmen um den Hund herum wäre.

Neben der Welt `Welt`, muss du nun auch eine Einheit `Hund` erstellen.

In der Programmierfläche des Hundes, muss natürlich das Hundebild geladen werden.

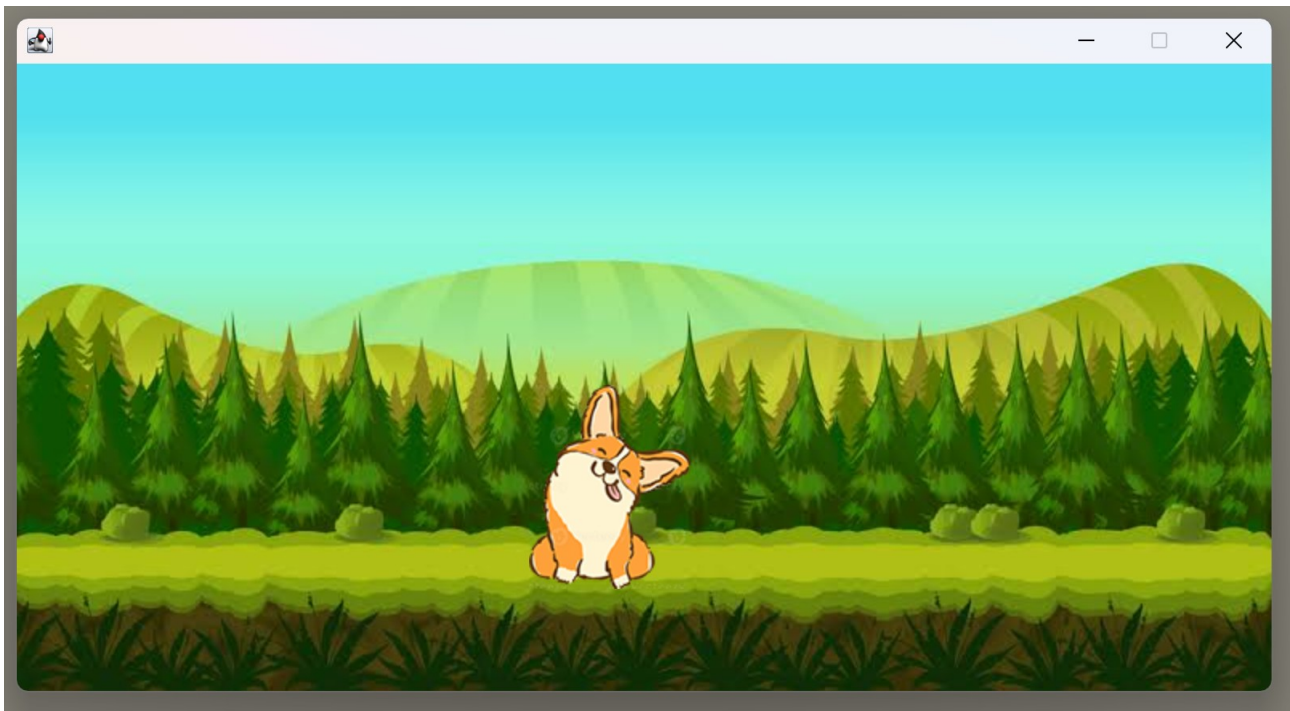
In jeder der Welt, muss das Hintergrundbild geladen werden. Du musst aber hier auch einen weiteren Block hinzufügen, welcher eine Hunde-Einheit einfügt. Du findest diese unter `Entity`. Das Ganze muss dann so aussehen:



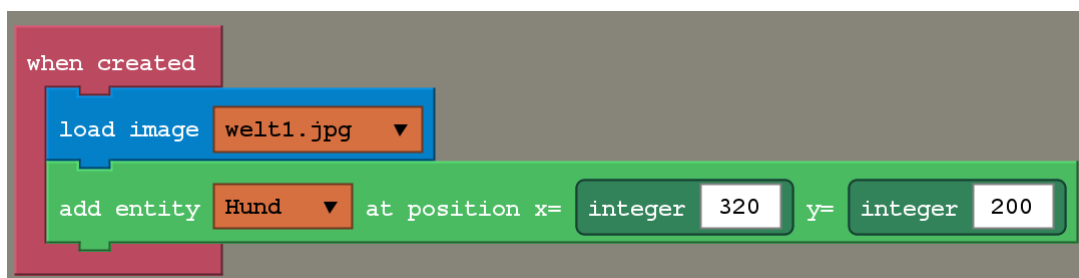
8.4. Welt mit Hund 2

Hunde können bekanntlich nicht fliegen. Also wäre es sinnvoll den Hund auf den Boden zu setzen anstatt dass er sich im Himmel befindet. Erstelle eine Kopie des vorherigen Aufgabe und öffne dann die Kopie in Moenagade um die entsprechenden Anpassungen darin vor zu nehmen.

Das Endresultat soll so aussehen:



Um den Hund an die entsprechende Stelle der Welt zu setzen, brauchst einen Block, der es erlaubt eine Position an zu geben. Das soll dann so im Programmierbereich der Welt aussehen:



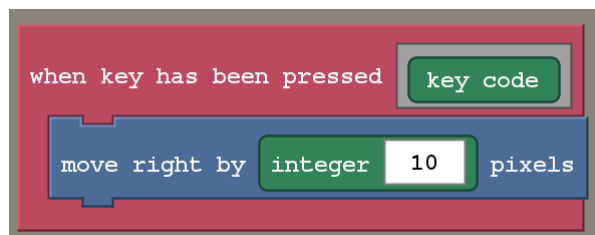
Zum Einfügen der numerischen Werte muss du als erstes unter `Values` zwei `integer` Blöcke an die entsprechenden Stellen ziehen. Mit einem Klick in die weiße Schreibfläche kannst du dann den Inhalt verändern.

8.5. Der Hund läuft weg

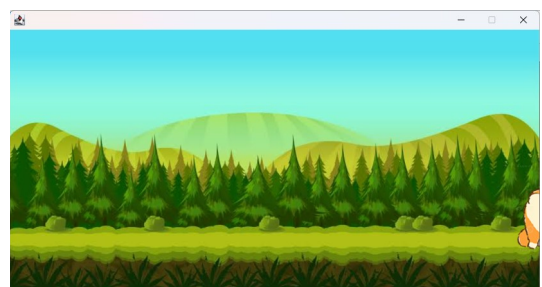
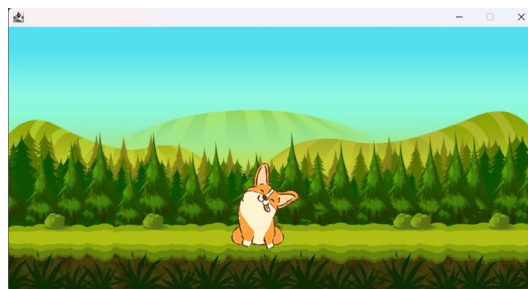
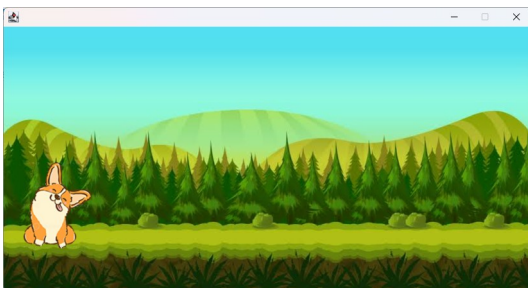
Erstelle erneut eine Kopie der vorherigen Aufgabe und mache folgende Anpassungen:

1. Der Hund erscheint nicht in der Mitte des Schirms sondern am linken Rand.
2. Sobald der Benutzer eine Taste drückt, bewegt der Hund sich um 10 Pixel nach rechts.

Um den Punkt zwei zu kodieren, brauchst du folgende Blöcke im Programmierbereich des Hundes:



Den Block `when key has been pressed` findest du bei Events und der Block `move right by` ist bei Entity zu finden.

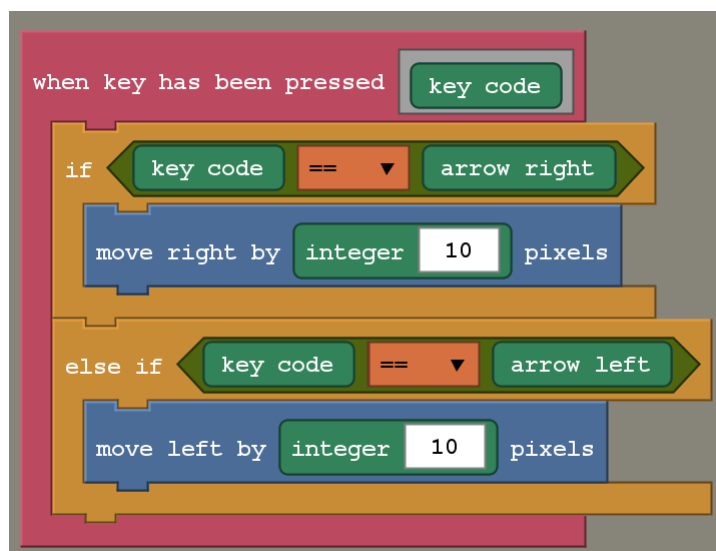


8.6. Links und rechts

Hunde laufen bekanntlich nicht nur in eine Richtung. In dieser Aufgabe geht es also darum, dass der Hund auch nach links laufen kann. Erstelle eine Kopie der vorherigen Aufgabe. Lade die Kopie und verändere diese.

Damit der Hund in zwei Richtungen laufen kann, muss kontrolliert werden, welche Taste der Benutzer drückt. Die Idee ist, dass wenn er die Feil-Taste „nach rechts“ drückt, der Hund nach rechts läuft und wenn er die „nach links“ Taste drückt, der Hund nach links läuft.

Hierzu wird eine `if` Block, der unter `Structures` zu finden ist, so wie ein `else if` Block gebraucht. Das soll dann so aussehen:



Der Block zum Vergleichen zweier Werte befindet sich bei `Logical`.



`key code` kann einfach aus der Block `when key has been pressed` nach unten gezogen werden. Der Block `arrow left` befindet sich bei `Keyboard`.

8.7. Dein Welt, deine Figur!

Erstelle ein neues Spiel! Suche dir im Internet eigenen Grafiken dafür, sprich einen Hintergrund für deine Welt so wie ein Bild für deine Figur.

Du darfst auch gerne andere Anpassungen machen, wie z.B. dass deine Figur sich noch nach oben und unten bewegen kann.

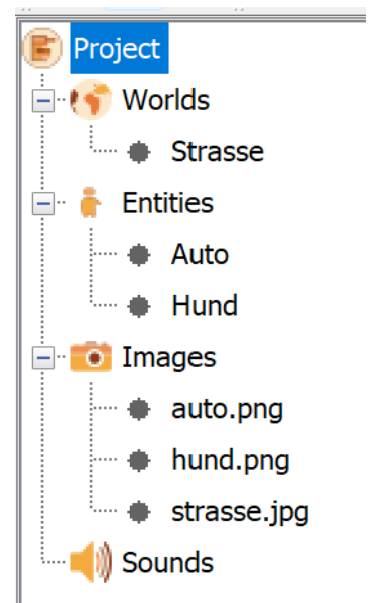
8.8. Gefangener Hunde

Erstelle eine Kopie der Aufgabe 6 dieses Kapitels und passe es so an, das der Hund nicht mehr links oder rechts aus dem Bildschirm herauslaufen kann. Das heißt also, dass der Hund zwar bis zum Bildschirmrand laufen kann, dann dort aber stehen bleibt.

Der Hund muss also zu jedem Zeitpunkt komplett sichtbar sein!

8.9. Achtung Hund!

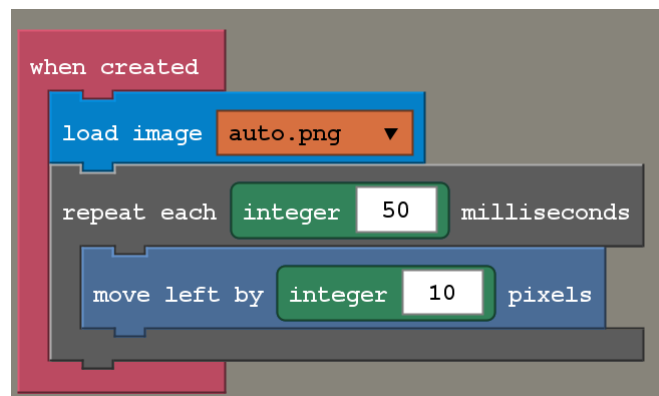
1. Lade die drei Bilder `hund.png`, `auto.png` und `strasse.jpg`.
2. Erstelle die Welt `Strasse` und lade darin das entsprechende Bild.
3. Erstelle die Einheit `Hund` und lade darin das entsprechende Bild.
4. Erstelle die Einheit `Auto` und lade darin das entsprechende Bild.
5. Wenn das Programm startet, soll die Welt `Strasse` geladen werden und der Titel des Spieles soll « Achtung Hund ! » anzeigen.
6. Beim Erstellen der Welt `Strasse` soll ein `Auto` an der Position (630,30) und ein `Hund` an der Position (20,40) eingefügt werden.
7. Kodiere den `Hund` so, dass wenn der Benutzer die Pfeiltaste „runter“ drückt, der Hund sich um 5 Pixel nach unten bewegt.
8. Kodiere das `Auto` so, dass wenn der Benutzer die Pfeiltaste „links“ drückt, das Auto sich um 10 Pixel nach links bewegt.



8.10. Das Auto rollt von selbst

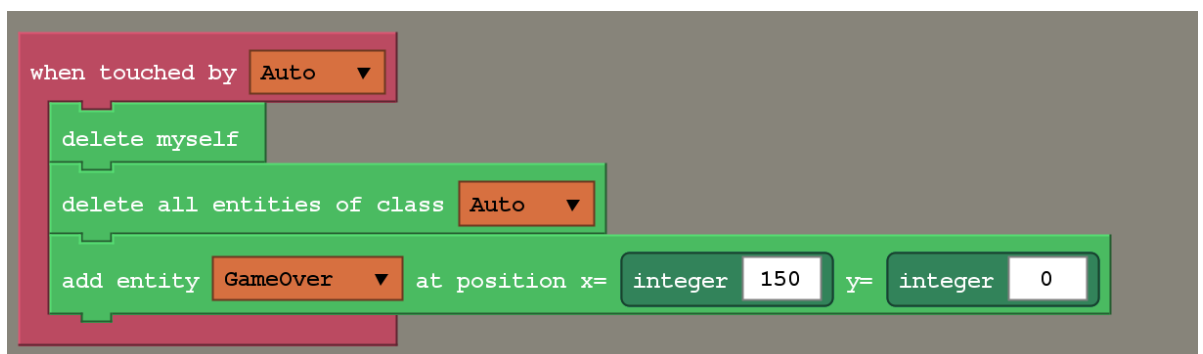
Die vorherige Aufgabe ist eigentlich langweilig. Es wäre doch sinnvoll, wenn das Auto nicht vom Spieler bewegt werden würde, sondern der Computer es selbst bewegen könnte.

1. Erstelle eine Kopie der vorherigen Aufgabe und lade die Kopie.
2. Lösche das Event `when key has been pressed` aus dem Auto.
3. Passe das `when created` Event wie folgt an:



Die `repeat each` Schleife, welche im Bereich `Timer` zu finden ist, führt den darin enthaltene Blöcke in regelmäßigen Zeitabständen aus. Hier wird das `Auto` jede 50 Millisekunden um 5 Pixel nach links verschoben. Das Verschieben startet sobald die Welt geladen wird.

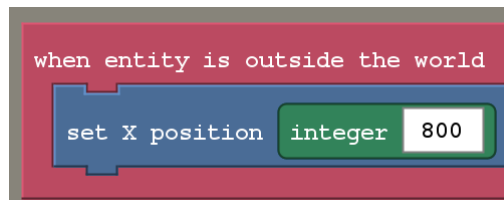
4. Wenn das Auto den Hund anstößt, sollte das Spiel sofort beendet werden. Um dies zu erreichen führe nun folgende Schritt aus:
 1. Lade das Bild `game-over.png`.
 2. Erstelle die Einheit `GameOver` und lade darin das Bild `game-over.png`.
 3. Füge dem Hund folgendes Event hinzu:



8.11. Neues Auto

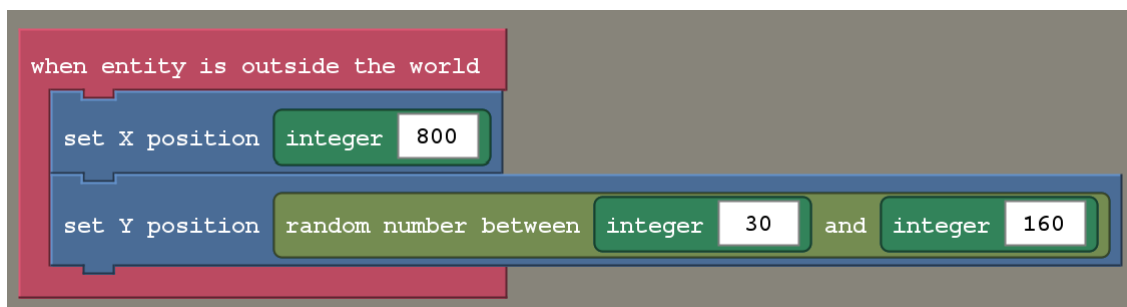
Wenn der Hund aus der vorherigen Aufgabe dem Auto ausweicht, passiert im Spiel nichts mehr. Es wäre wünschenswert, wenn – sobald das Auto vom Bildschirm verschwindet – es wieder am rechten Rand auftaucht.

Um dies zu erreichen musst du folgende Blöcke dem `Auto` hinzufügen:

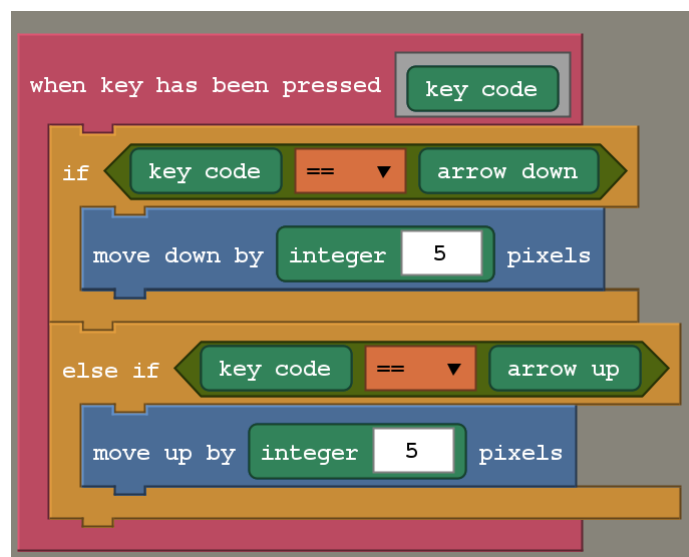


Leider taucht das Auto mit dieser Programmierung immer an der gleichen Stelle auf.

Um das Auto an immer einer anderen Stelle der `Strasse` auftauchen zu lassen, füge folgende Blöcke hinzu. Der Block `random number between` befindet sich im Bereich `Mathe`.



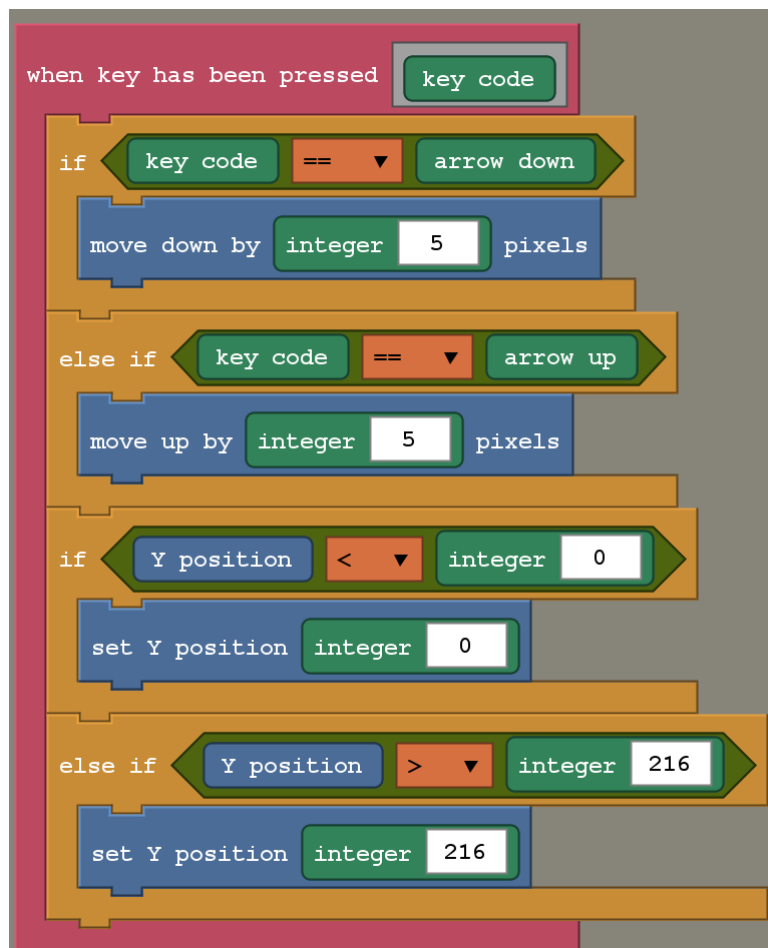
Programmiere auch den `Hund` so, dass der Benutzer ihn auch nach oben bewegen kann:



8.12. Hundelimits

Im Moment kann der Hund aus dem Spielfeld heraus laufen, so dass man ihn nicht mehr sieht und er auch nicht mehr vom Auto getroffen werden kann.

Passe die Programmierung des Hundes so an, dass dieser die Welt `Strasse` nicht mehr verlassen kann.

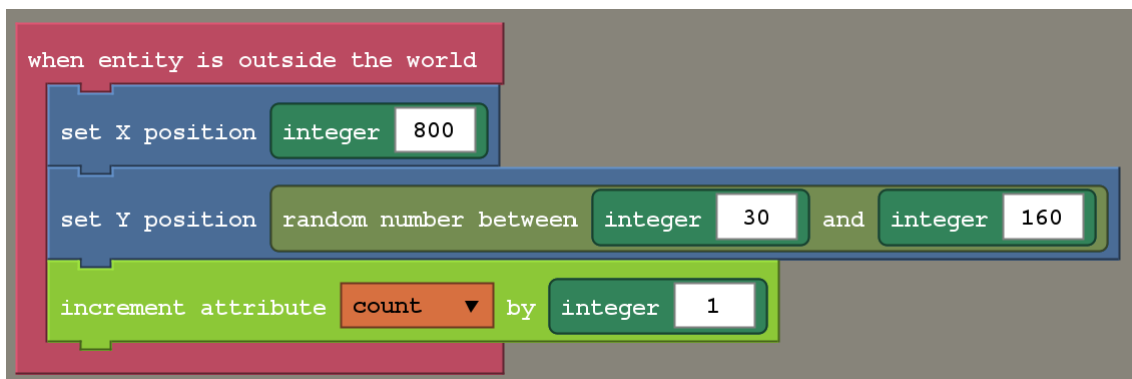


8.13. Auto zählen

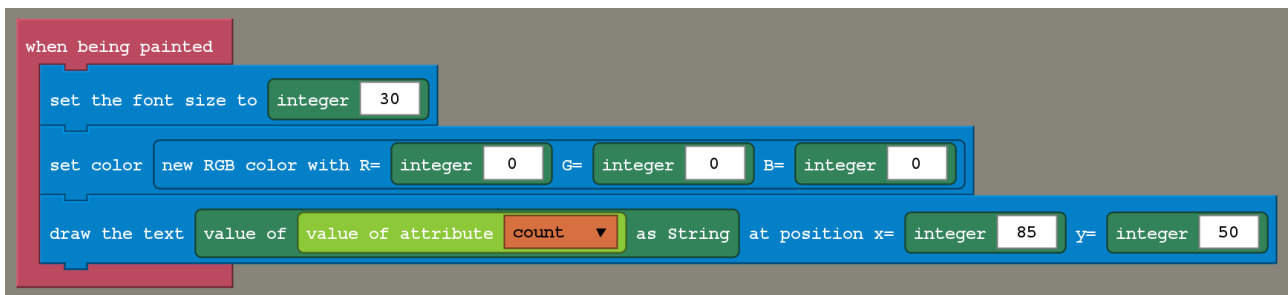
Jedes Auto, das über die Straße fährt, soll gezählt werden. Um dies zu erreichen brauchen wir einen Zähler. Dafür musst du in der Einheit `Auto` ein sogenanntes Attribut definieren:



Dieses wird dann jedes Mal wenn das Auto die Straße verlässt um 1 erhöht.



Der Wert dieses Attributes kann z.B. auf dem Dach des Auto angezeigt werden.

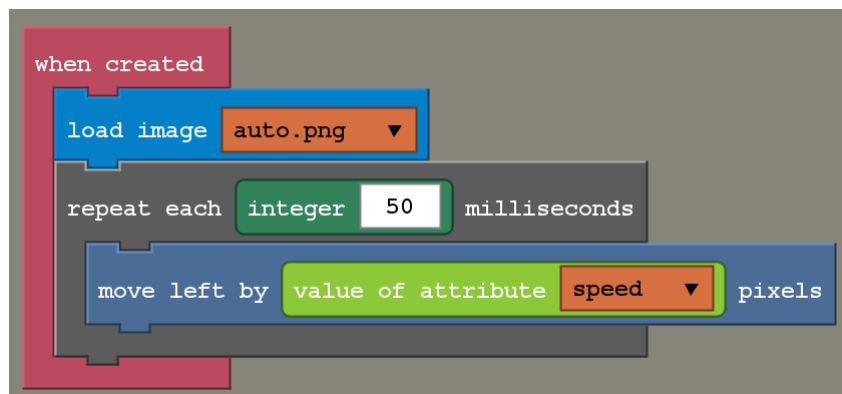


8.14. ***Speed up!***

Eine tolle Erweiterung ist jene, dass das jedes neue Auto schneller ist als das vorherige. Um dies zu erreichen, müssen wir ein weiteres Attribut in der Einheit Auto definieren.



Dieses Attribut muss dann natürlich auch zur Fortbewegung des Auto eingesetzt werden.



Außerdem muss sein Wert bei jedem neuen Auto erhöht werden, damit das nächste Auto dann schneller ist.

