MoleMash

Schlag den Maulwurf

1. Vorbereitung und Benutzeroberfläche

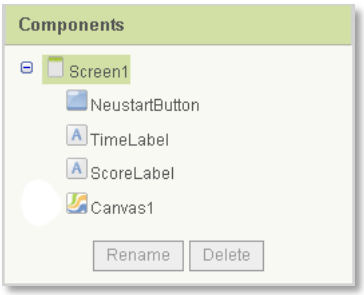
a) Lege ein neues Projekt an und gib ihm einen Namen (z.B. *MoleMash*). Die App wird nachher unter diesem Namen auf dem Smartphone/Tablet erscheinen.

b) Entscheide dich, ob das Spiel im Quer- oder Hochformat gespielt werden soll. Das Format kann angepasst werden, indem man unter *Components* auf *Screen1* klickt und unter *ScreenOrientation* die Bildschirmausrichtung *Portrait* oder *Landscape* auswählt.

c) Lade ein Bild und einen Ton hoch: Klicke dazu unten bei *Media* auf *Upload File*. Einige Bilder und Töne findest du auf dem Desktop in den Ordnern *Bilder* bzw. *Sounds*.

d) Ziehe folgende Bildschirmelemente aus der Palette auf die Bildschirmfläche:

- 1 *Button*
- 2 *Labels*
- 1 *Canvas* (zu finden unter *Drawing and Animation*)



Der Canvas ist die Hauptkomponente für das Spiel und sollte **so viel Platz wie möglich** einnehmen.
Um die Elemente schöner zu anzuordnen, kannst du *SceenArrangements* verwenden (in der Palette zu finden unter *Layout*). Außerdem kann jedem Bildschirmelement unter *Properties* eine Höhe (*Height*) und eine Breite (*Width*) zugewiesen werden.
Benenne die Elemente so wie im Bild.

e) Starte den Emulator über *Connect* → *Emulator* und teste deine Oberfläche.

2. Der Maulwurf

a) Ziehe ein *ImageSprite* aus der Palette (unter *Drawing and Animation*) auf den *Canvas*. Benenne es in *MoleSprite* um.
Wähle unter *Picture* das in 1c) hochgeladene Bild aus. Passe die Größe über die Felder *Height* und *Width* an.

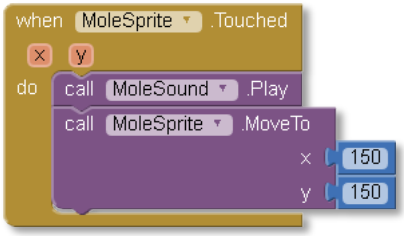
b) Ziehe ein Sound-Element aus der Palette (unter *Media*) auf die Benutzeroberfläche. Benenne ihn in *MoleSound* um.
Weise ihm dann unter *Source* den in 1c) hochgeladenen Ton zu.

c) Wechsle oben rechts zur *Blocks*-Ansicht. Klicke (unter *Screen1*) auf *MoleSprite* und ziehe die Funktion when **MoleSprite.Touched** auf die Arbeitsfläche.
Dann klicke auf das Sound-Element *MoleSound* und ziehe die Funktion call **MoleSound.Play** auf die Arbeitsfläche und hänge sie in die *Touched*-Funktion ein.

3. Bewegung

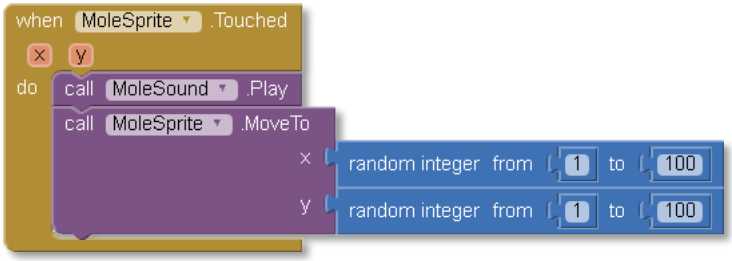
Der Sprite soll bei Berührung an eine zufällige Position auf dem *Canvas* springen.

a) Klicke unter *Screen1* → *Canvas1* → *MoleSprite* auf die Funktion call **MoleSprite.MoveTo** und hänge sie in die *Touched*-Funktion ein.
Dieser Baustein lässt den MoleSprite an eine andere Position springen, die du über die x- und y-Koordinate festlegen kannst.



b) Teste die App für verschiedene x- und y-Koordinaten.

c) Um den Sprite an eine **zufällige** Position springen zu lassen, kannst du die Funktion **random integer** verwenden. Diese findest du unter *Built-In* → *Math*. Die Funktion generiert eine Zufallszahl zwischen zwei Werten (anfänglich zwischen 1 und 100).
Ersetze die festen Werte für die x- und y-Koordinaten mit dem Aufruf dieser Funktion und teste die App.



Das ist die Basisversion von MoleMash. Nun bist du gefragt! Überlege dir, wie du das Spiel spannender machen kannst.
Im Folgenden findest du eine Reihe von Ideen inklusive Beschreibung, wie du diese umsetzen kannst.

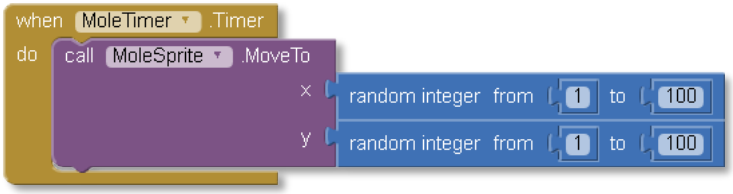
Erweiterungsideen

1. Der Maulwurf springt alle x Sekunden von alleine, nicht nur beim Antippen.

a) Ziehe eine *Clock* aus der Palette (*Sensors*) auf die Benutzeroberfläche und nenne sie *MoleTimer*.

b) Ziehe im *Blocks*-Editor die Funktion when **MoleTimer.Timer** auf die Arbeitsfläche und lass den Maulwurf so springen, wie du es in 3c) bereits getan hast.

Die *Clock* führt in regelmäßigen Zeitabständen den Inhalt der **Timer**-Funktion aus. Diesen Zeitabstand kannst du im *Designer*-Editor bei den Properties des *MoleTimers* unter *TimerInterval* einstellen. Willst du den Maulwurf zum Beispiel alle 100 ms springen lassen, so musst du für das Intervall den Wert 100 wählen.



2. Punktezähler

Benutze das in 1d) erstellte *Label*, um die Anzahl der getroffenen Maulwürfe anzuzeigen. Dazu brauchst du eine Variable, die jedes Mal wenn der Maulwurf getroffen wird, mitzählt.

a) Klicke unter *Built-In* → *Variables* auf

. Benenne die Variable von „name“ in „score“ um, und gib ihr den Startwert 0.

b) Jedes Mal, wenn der Maulwurf getroffen wird, sollte die Variable um 1 erhöht werden. Dies funktioniert nach dem Schema:

$$score = score + 1$$

Klicke auf *Built-In* → *Variables* und ziehe

auf den Arbeitsbereich. Wähle die eben erstellte Variable *score* aus. Danach muss hier die obige Gleichung eingehängt werden:
Den **+**-Baustein findest du unter *Built-In* → *Math*. Den Wert der Variablen kannst du über *Built-In* → *Variables* →

auslesen.

c) Das Label soll nun die in global **score** gespeicherte Punktzahl anzeigen. Klicke dazu auf *Screen1* → *ScoreLabel* und ziehe set **ScoreLabel.Text** to auf die Arbeitsfläche. Hänge dann den Wert der Variablen über get **global score** ein.

d) Beide Blöcke sollten in die **MoleSprite.Touched**-Funktion eingefügt werden.

3. Zeitmessung

Eine Zeitanzeige soll zeigen, wieviele Sekunden seit Spielstart schon verstrichen sind.

a) Ziehe eine *Clock* aus der Palette (unter *Sensors*) auf die Benutzeroberfläche und nenne sie *GameTimer*. Wechsle zum *Blocks*-Editor.

b) Erstelle eine Variable (*Built-In* → *Variables*) und nenne sie *time*. Diese soll die verstreichenden Sekunden speichern. Gib ihr den Startwert 0.

c) Ziehe die Funktion when **GameTimer.Timer** auf die Arbeitsfläche. Diese Funktion wird standardmäßig alle 1000 ms (= 1 Sekunde) ausgeführt – perfekt für einen Sekundenzähler.

d) Bei jedem Aufruf soll nun die Variable *time* um 1 erhöht werden:

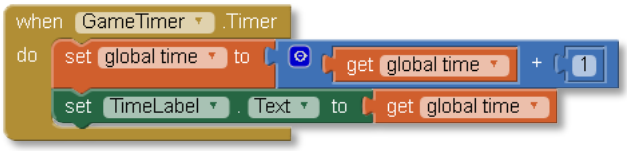
$$time = time + 1$$

Klicke dazu auf *Built-In* → *Variables* und ziehe auf den Arbeitsbereich. Wähle *global time* aus. Dann sollte hier die obige Gleichung eingehängt werden.
Den **+**-Baustein findest du unter *Built-In* → *Math*. Den Wert der Variablen kannst du über *Built-In* → *Variables* → auslesen.

Falls du anstatt eines Sekundenzählers lieber einen Countdown hättest, musst du nur den Anfangswert der Variable *time* auf den Startwert des Countdowns setzen und dann in jedem Schritt 1 davon abziehen:

$$time = time - 1$$

e) Um die Zeit anzuzeigen, wähle ein *Label* von deiner Benutzeroberfläche (hier *TimeLabel*) und setze den Text auf den Wert der Variablen *time*.

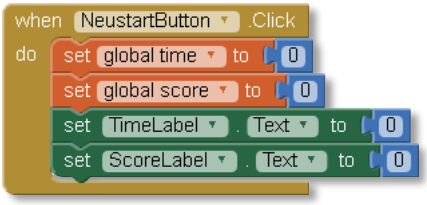


4. Neustartknopf

Damit man die App nicht jedes Mal neu starten muss, wenn man ein neues Spiel anfangen will, benötigst du einen Neustartknopf, der den Zustand des Spiels auf die Anfangswerte zurücksetzt.

a) Klicke im *Blocks*-Editor unter *Screen1* → *NeustartButton* auf when **NeustartButton.Click**.

b) Setze in diesem Block nun alle Werte auf ihren Anfangswert, wie zum Beispiel die Punkte, die verstrichene Zeit oder ggf. auch die Position des Maulwurfs.



5. Spielende

Ist das Spielende erreicht, soll das Spiel angehalten werden.

a) Um den Spielfluss zu pausieren, brauchst du als erstes eine Variable:
Klicke im *Blocks*-Editor unter *Built-In* → *Variables* auf

. Nenne die Variable *playing*.
Gib ihr den Wert **true** (*Built-In* → *Logic,* **true** & **false** sind **keine** Text-Bausteine).

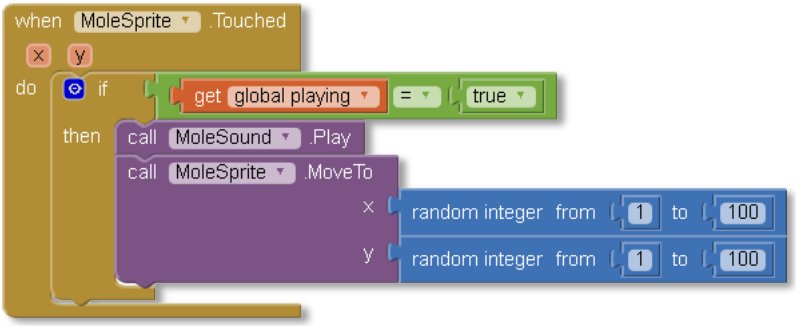
Diese Variable gibt an, ob das Spiel gerade läuft (*playing* = **true**) oder beendet ist (*playing* = **false**).

b) Alle Aktionen die während des Spiels möglich sind, sollen nun von dieser Variable abhängig gemacht werden und nur ausgeführt werden, wenn *playing* = **true** ist, also das Spiel noch läuft.
Dies lässt sich mit **if**-Abfragen realisieren. Diese überprüfen eine Bedingung und führen Anweisungen aus, wenn diese Bedingung wahr (**true**) ist.

c) Klicke auf *Built-In* → *Logic* → =, um die Bedingung zu formulieren. Fülle sie mit der Variablen *playing* und dem *Logic*-Wert **true**.

d) Ziehe eine **if**-Abfrage (*Built-In* → *Control*) auf die Arbeitsfläche und stecke oben die eben erstellte Bedingung ein. Alles in dem **then**-Block wird nur ausgeführt, wenn *playing* = **true** ist, also das Spiel gerade läuft.

e) Um nun beispielsweise das Antippen des Maulwurfs von der *playing* abhängig zu machen, musst du den **MoleSprite.MoveTo**- und sinnvollerweise auch den **MoleSound.Play**-Aufruf mit dem **if**-Block „umschließen“:

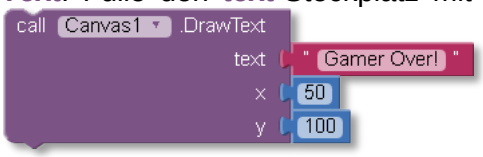


f) Jetzt musst du dir nur noch überlegen, wann das Spiel enden soll.
Erstelle dir dafür dann analog eine **if**-Abfrage mit einer Bedingung, die im **then**-Block die Variable *playing* = **false** setzt.

6. Text auf dem Bildschirm anzeigen

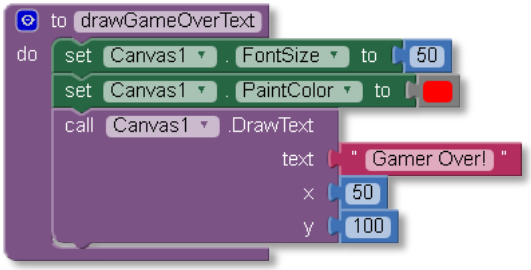
Ein Text, wie z.B. „Game Over“ soll auf dem Bildschirm angezeigt werden.

a) Klicke auf *Screen1* → *Canvas1* → **Canvas1.DrawText**. Fülle den **text**-Steckplatz mit einem Text-Block (*Built-In* → *Text*).
Die Koordinaten **x** und **y** geben die Position an, an der der Text erscheinen soll.



b) Um Größe und Schriftfarbe zu verändern, musst du über den **DrawText**-Aufruf die Anweisung set **Canvas1.FontSize** to bzw. set **Canvas1.PaintColor** to einfügen. Farben findest du unter *Built-In* → *Colors*.

c) Erstelle nun eine *Procedure*, indem du den Block to **procedure** do (*Built-In* → *Procedure*) auf die Arbeitsfläche ziehst.
Benenne sie von „procedure“ in beispielsweise „drawGameOverText“ um und hänge dort alle eben erstellten Blöcke ein.



Wann immer der Text angezeigt werden soll, muss jetzt einfach nur noch über

(*Built-In* → *Procedures*) die *Procedure* aufgerufen werden.

<u>Hinweis</u>: Eine *Procedure* fasst mehrere Operationen zusammen, die mit einem einzelnen Aufruf von überall ausgeführt werden können. Auf diese Weise muss nicht jede Operation einzeln eingefügt werden, was das Programm übersichtlicher macht.

d) Der erstellte Text bleibt solange auf dem Bildschirm stehen, bis er mit dem Aufruf

(*Screen1* → *Canvas1*) entfernt wird. Setze diesen Block an eine geeignete Stelle in deinem Programm (falls du den Neustartknopf implementiert hast, etwa in den when **NeustartButton.Click**-Block).

7. Freie Arbeit

Erweitere dein Spiel nach deinen Vorstellung und Wünschen. Was fehlt, um es wirklich optisch ansprechend, interessant, spaßig oder anspruchsvoll zu machen?
Für den Fall, dass dir auf Anhieb nichts einfällt, gibt es im Folgenden eine kleine Liste mit Anregungen und Hinweisen.

Anregungen:

- **Schönere Zeitanzeige:**
  Die Funktionen **modulo**- und **quotient** unter *Built-In* → *Math* ermöglichen dir eine Zeitanzeige im „Minuten : Sekunden“-Format.

- **Mehr Maulwürfe:**
  Du kannst mehr Maulwürfe einfügen, indem du mehr *Sprites* auf das *Canvas* ziehst. Im *Blocks*-Editor unter *Any component* findest du Funktionen, die die Arbeit mit mehreren *Sprites* einfacher machen.
  Du kannst die Anzahl der Maulwürfe auch während des Spiels ändern, indem du das Attribut *visible* der *Sprites* veränderst.

- **Unberechenbares Verhalten der Maulwürfe:**
  Du könntest nach jedem Sprung das Intervall der Maulwurf-*Clock* zufällig anpassen.

- **Minuspunkte durch Igel**
  Du kannst Igel einfügen, die sich ähnlich verhalten, wie die Maulwürfe, aber Minuspunkte geben, wenn man sie berührt.

- **Verschiedene Schwierigkeitsgrade oder Spielmodi:**
  Ihr könnt einige Buttons und globale Variablen einfügen, über die sich das Verhalten des Maulwurfs oder auch die Punktevergabe ändern lässt.

- **Automatische Geschwindigkeitsanpassung:**
  Du kannst die Geschwindigkeit automatisch anpassen lassen, wenn der Spieler einen Maulwurf trifft oder verfehlt. Je nachdem wieviel schneller bzw. langsamer das Spiel pro Treffer bzw. Fehlschlag wird, sind besonders einfache oder besonders anspruchsvoll Schwierigkeitsgrade möglich.

- **Punktespeicherung:**
  Du kannst mit *Buttons*, einem *Notifier* und einer *TinyDB* eine Highscoreliste speichern und wieder auslesen.
  Den Inhalt der Highscoreliste kannst du über ein großes Label wieder ausgeben, wenn du ihn mit den Bausteinen unter *Built-In* → Text formattierst.
  Um alles übersichtlich zu halten, kannst du auch Bildschirmelemente (auch *ScreenArrangements*) ein- und ausblenden über das Attribut *visible*.

- **Leben / alternatives Spielende:**
  Du kannst mit einer weiteren globalen Varialbe und **if-else**-Abfragen dem Maulwurf eine bestimme Anzahl an Leben geben und das Spiel enden lassen, sobald alle Leben weg sind, anstatt nach einer festen Zeit.
  Oder ihr gebt dem Spieler nur eine begrenzte Anzahl von Versuchen mit denen er möglichst viele Maulwürfe treffen soll.

- **„Laufende“ Maulwürfe:**
  Du kannst die Maulwürfe auch laufen anstatt springen lassen, indem du die Werte *Heading* und *Speed* der *Sprites* anpasst. Die Methode call **ImageSprite.Bounce** lässt in Kombination mit when **ImageSprite.EdgeReached** ein Sprite von einer Kante des Canvas abprallen.