KODARKLUBBEN

ROBLOX

Lathund

Tips:

Alvinblox har många bra tutorials hur man kan bygga spel i roblox, exempelvis följande om grundläggande skripting: https://www.youtube.com/watch?v=yc_d8Zh95l0

Funktioner

Funktioner ger dig möjlighet att ändra många olika saker, men endast behöva ge en instruktion. Det gör att funktioner är väldigt bra om man vill göra samma sak många gånger utan att behöva kopiera samma kod, om och om igen.

Mer beskrivning hittar du om funktioner här:

https://developer.roblox.com/articles/Understanding-Functions-in-Roblox

Här är ett exempel på en funktion som skriver ut 4 rader text

```
function iLikeGame()
    print("Jag tycker om Jail Break!")
    print("Vilket är ditt favorit spel?")
end
```

Genom att man kör samma kommando två gånger:

```
iLikeGame()
iLikeGame()
```

Om man skulle vilja variera vad den skriver ut så kan man göra det genom att lägga till en parameter. Här är ett exempel på en funktion som skriver ut 3 rader text men byter favorit spel.

```
function iLikeGame(bestGame)
    print("Jag tycker om ", bestGame, "!")
    print("Vilket är ditt favorit spel?")
end
```

Genom att man kör ett kommando:

```
iLikeGame("Jail Break")
iLikeGame("Vehicle Simulator")
iLikeGame("McDonald's Tycoon")
```

Uppgift: Skapa ett ”dödarblock”

En spelare dör om man ändrar hälsan för spelaren till 0. Detta kan man göra genom att skapa en funktion enligt följande:

```
function kill(player)
    player.Parent.Humanoid.Health = 0
end
```

För att koppla funktionen till ett block så kan man använda händelsen Touched som inträffar så fort man rör vid blocket. Då kommer funktionen att köras när spelaren på något sätt rör delen.

```
script.Parent.Touched:Connect(kill)
```

Skriv både funktionen och denna kopplingskod i ett script för en ny “Part”, så har du skapat ett hinder som dödar spelaren så fort han rör vid blocket.

Utmaningar

1. Skapa ett block som bara skadar dig
   Tips: Istället för att sätta hälsa till 0. Minska hälsan med 10 varje gång man rör blocket:

   ```
   player.Parent.Humanoid.Health = player.Parent.Humanoid.Health - 10
   ```

2. Skapa ett block som återställer hälsan

3. Skapa ett block som ger dig superkrafter
   Tips: Titta på Humanoid vilka andra egenskaper än hälsa som du kan ändra.

Villkors satser

Om man vill kontrollera att något måste vara på ett visst sätt för att något annat ska få hända använder man villkors satser. Detta kapitel är en sammanfattning, men följ länken för mer detaljerad information: https://developer.roblox.com/articles/Conditional-Statements-in-Lua

Den enklaste varianten av en villkors-sats är

```
if "villkoret" then
    "vad man vill att ska hända när villkoret är uppfyllt"
end
```

Villkoret uttrycker man ofta som ett mattetal, exempelvis:

2 + 3 == 5

Exempelvis kunde man skriva ut raden genom att ha följande uttryck:

```
if 2 + 3 == 5 then
    print("två pluss tre är fem")
end
```

Man har satt olika regler på hur man ska uttrycka villkor, två likamedstecken betyder exempelvis att man kör satsen om de är lika. Detta kallar man inom kodning för en relations operator. I Roblox finns följande operatorer:

Operator	Betydelse	Exempel	
<	Mindre än	3 < 5	Sant
>	Större än	3 > 5	Falskt
<=	Mindre eller lika med	5 <= 5	Sant
>=	Större eller lika med	3 >= 5	Falskt
==	Lika med	3 == 5	Falskt
~=	Inte lika med	3 ~= 5	Sant

Följ länken för fler operatorer: https://developer.roblox.com/articles/Operators

Else

Om man vill göra något annat ifall villkoret inte är uppfyllt kan man använda en Else sats. Man forsätter if-satsen genom att lägga till ett else enligt följande:

```
if "villkoret" then
    "vad man vill att ska hända när villkoret är uppfyllt"
```

```
else
    ”vad som ska hända annars”
end
```

Så om man vill testa om variabeln tal är större eller mindre än 100 kan man göra enligt följande:

```
if tal > 100 then
    print("tal är större än 100")
else
    print("tal är mindre än 100")
end
```

Elseif

Man kan även kontrollera flera villkor i samma sats. Exemplet som togs ovan har ju ett fel, vad händer om tal är 100, då skrivs ju ”tal är mindre än 100” ut. För att rätta satsen kunde man lägga till ett till villkor enligt:

```
if tal > 100 then
    print("tal är större än 100")
elseif tal == 100 then
    print("tal är 100")
else
    print("tal är mindre än 100")
end
```

Uppgift: Skapa en dörr

Testa att skapa en dörr genom att skapa ett block som varannan gång man klickar på den gör den genomskinlig och genomtränglig så att det känns som om den är öppen, och varannan gång man klickar på den stänger den genom att visa blocket och inte göra så att man kan gå igenom det.

Dra ut en ny Part. Lägg till en ClickDetector och ett skript. Skriptet nedan gör precis just det.

```
function changeDoor()
  door = script.Parent              -- hämta skriptets förälder
  if door.Transparency == 0 then    -- om dörren är synlig
    door.Transparency = 1           --   gör den genomskinlig
    door.CanCollide = false         --   gör den genomtränglig
  else                              -- annars
    door.Transparency = 0           --   gör den synlig
    door.CanCollide = true          --   gör den inte genomtränglig
  end
end

-- gör så att funktionen kopplas till ett musklick
script.Parent.ClickDetector.MouseClick:connect(changeDoor)
```

Uppgift: Skapa en oändlig trappa

Skapa det nedersta trappsteget genom att dra ut en ny part, och skapa ett nytt skript under den. I skriptet, gör en ny funktion som kopierar steget med funktionen Part:Clone(), lägger till objektet under dess förälder och flyttar objektet i höjdled och sidled.

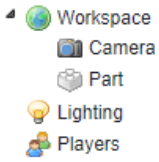
```
local nextStepCreated = false

function nextStep(player)
  if nextStepCreated == false then
    local part = script.Parent:Clone()
    part.Parent = script.Parent.Parent
    part.Position = script.Parent.Position + Vector3.new(-3,1,0)
    nextStepCreated = true
  end
end

script.Parent.Touched:Connect(nextStep)
```

Utmaningar

1. Gör så att dörren stängs automatiskt efter 10 sekunder.
   Tips: Använd wait(sekunder) funktionen
2. Ändra trappan så att om man klickar på ett steg så förstörs det
   Tips: Det finns en funktion Part:Destroy()

Hur förhåller sig objekt till varandra

För att komma åt att påverka hur ditt spel fungerar behöver du sätt där du kan peka på de objekt och delar som finns i din värld. Du kan tänka på detta på samma sätt som du ser på de objekt som syns i Explorer fönstret i Roblox Studio:

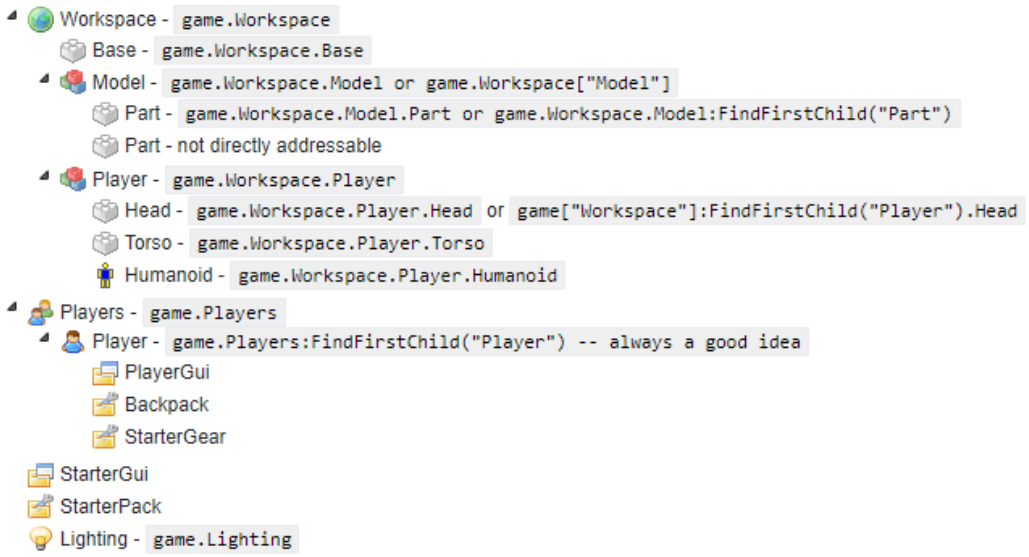


Att ta sig runt

Det finns olika sätt att komma åt objekten med olika genvägar. Men högst upp i hierarkin finns alltid ***game***. Objekt som man lägger under ett annat objekt kallar man för **barn** (på engelska ***Child***). Och om man befinner sig i ett barn, kan man komma åt objektet ovanför det genom att kalla på dess **förälder** (på engelska ***Parent***). På dessa sätt kan du komma åt ett barn:

```
parent.child
parent["child"]
parent:FindFirstChild("child")
```

Så för att nå följande objekt i vårt träd refererar vi till dem enligt följande:



Nedan visas några exempel på att ta sig tillbaka uppåt i trädet:

```
b = game.Workspace.Base
part = game.Workspace.Model.Part
p = game.Players.Player
gui = game.Players.Player.PlayerGui

gui.Parent -- same as game.Players.Player
```

```
gui.Parent.Parent  --same as p.Parent or game.Players
gui.Parent.Parent.Parent -- same as game
part.Parent.Parent.Base -- same as b
```

Variabel genvägar

I skript kan man använda speciella genvägar för att snabbar komma åt objekt.

```
game              -- är roten i hierarkin
workspace         -- är stället där vi normalt placerar de objekt som är synliga
script            -- är det skript där koden körs
```

Inbyggda funktioner

I Roblox finns även några inbyggda funktioner som kan vara användbara.

delay(sekunder,funktion)	Kör en funktion angivet antal sekunder senare.
elapsedTime()	Returnerar tiden i sekunder sen Roblox instansen startade.
tick()	Returnerar tiden i sekunder sen 1 Januari 1970 (Unix epoch)
time()	Returnerar tiden i sekunder sen nuvarande spel instans startade att köra.
wait(sekunder)	Pausar den körande koden i angivet antal sekunder och fortsätter sedan.
warn(text)	Gör ungefär samma sak som print, men i gul text och tillsammans med tid.

Part

Part är en del, såkallat fysiskt objekt. När det är i Workspace kommer det att flytta och interagera med andra delar. Delar är de grundläggande byggstenarna i en Roblox-plats. De kan göras väldigt stora och användas som exempelvis basplatta eller göras väldigt små för kombineras med andra delar och skapa ett häftigt utseende.

Du kan enkelt skapa nya delar genom kod som `Instance.new("Part")`

Part hittas på länken: https://developer.roblox.com/api-reference/class/Part

egenskaper

`PartType` Shape `[ notreplicated ]`	Ställer in typ av objektets buk.
`float` LocalTransparencyModifier `[ hidden ]` `[ notreplicated ]`	Bestämmer en multiplikator för `BasePart / Transparency` som endast är synlig för den lokala klienten
`float` Transparency	Bestämmer hur genomskinlig delen är
`Vector3` Size `[ notreplicated ]`	Bestämmer storleken för en del (längd, bredd, höjd)
`Vector3` Rotation `[ notreplicated ]`	Rotationen av delarna av de tre axlarna.
`float` Reflectance	Bestämmer hur mycket en del återspeglar skyboxen.
`Vector3` Position `[ notreplicated ]`	Beskriver positionen för delen i världen.
`Material` Material	Bestäm texturen och de grundläggande fysikaliska egenskaperna hos en del
`Color3` Color `[ notreplicated ]`	Bestämmer färgen på en del.
`bool` CanCollide	Bestämmer om en del kan kollidera med andra delar.

BrickColor BrickColor [notreplicated]	Bestämmer färgen på en del.
bool Anchored	Bestämmer om delen är fast i världen eller om den påverkas av fysik
Instance Parent	Bestämmer till vilken del den hör
string Name	Namnet på delen

funktioner

Instance Clone ( )	Skapar en djup kopia av ett Roblox objekt och dess barn, egenskaper och sätter Archivable = true.
void Destroy ( )	Sätter egenskapen Instance.Parent till nil, låser egenskapen Instance.Parent, tar bort alla kopplingar och kallar på Destroy för alla dess barn.
Objects GetTouchingParts ()	Returnerar en tabell över alla CanCollide sanna delar som korsar den här delen.
bool Resize (NormalId normalId , int deltaAmount)	Ändrar objektets storlek.

händelser

RBXScriptSignal Touched (Instance otherPart)	Avfyras när en del kommer i kontakt med en annan del
RBXScriptSignal TouchEnded (Instance otherPart)	Avfyras när en del slutar röra vid en annan del.

Humanoid

En humanoid är en varelse som till kroppsformen påminner om en människa. I Roblox är det karaktären som spelaren spelar. Det finns två olika humanoids, R6 som är uppdelad i 6 olika kroppsdelar (som armar och ben) och R15 som är uppdelad i 15 olika kroppsdelar.

Hittas på länken: https://developer.roblox.com/api-reference/class/Humanoid

egenskaper

float Health [notreplicated]	Humanoidens nuvarande hälsa, från 0 till MaxHealth. Om hälsovärdet når 0, kommer humanoiden att dö och dess anslutningsfogar kommer att tas bort.
float HipHeight	Bestämmer avståndet från marken som Humanoid.RootPart ska vara.
bool Jump [notreplicated]	Huruvida Humanoiden hoppar. Om den är satt till sann, kommer det att orsaka humanoiden att hoppa.
float JumpPower	Bestämmer hur mycket uppåtkraft som appliceras på humanoiden när man hoppar.
float MaxHealth	Det högsta värdet av en humanoids hälsa.
float MaxSlopeAngle	Den maximala lutningsvinkeln som en humanoid kan gå på utan att glida. Detta värde är fastklämt mellan 0 ° och 89 °.
Vector3 MoveDirection [readonly]	Beskriver den riktning som Humanoiden går in, som enhetsvektor längs X / Z-axeln.
BasePart SeatPart [readonly]	En hänvisning till sätet att en humanoid sitter i, om någon.
bool Sit	Beskriver huruvida Humanoid sitter för närvarande.
float WalkSpeed	Beskriver Humanoidens maximala rörelsehastighet i studs / sek.

funktioner

void AddAccessory (Instans tool)	Anger den angivna Accessory till Humanoid
void EquipTool (Instance tool)	Gör Humanoid utrusta det givna Tool

Array GetAccessories ()	Returnerar en uppsättning Accessories som Humanoid bär på
void Move (Vector3 moveDirection , bool relativeToCamera)	Orsakar Humanoid att gå i den givna riktningen
void MoveTo (Vector3- plats , Instansdel)	Förorsakar Humanoid att försöka gå till den angivna platsen genom att ställa Humanoid.WalkToPart egenskaperna Humanoid.WalkToPoint och Humanoid.WalkToPart
void RemoveAccessories ()	Ta bort alla Accessories bärs av Humanoid
void TakeDamage (float mängd)	Subtraherar skadan från den riktade humanoidens hälsa om inte karaktären har en ForceField .
void UnequipTools ()	Unequips alla Tool närvarande är utrustade med Humanoid

händelser

RBXScriptSignal Climbing (float speed)	Händer när den hastighet vid vilken en Humanoid klättrar ändras
RBXScriptSignal Died ( )	Händer när Humanoid dör
RBXScriptSignal FallingDown (bool aktiv)	Händer när Humanoid kommer in eller lämnar *FallingDown* HumanoidStateType
RBXScriptSignal HealthChanged (float health)	Händer när Humanoid's Humanoid.Health förändras till ett värde som är lika med eller lägre än Humanoid.MaxHealth värdet
RBXScriptSignal Jumping (bool active)	Händer när Humanoid kommer in och lämnar *hoppar* HumanoidStateType
RBXScriptSignal Running (float speed)	Händer när hastigheten där en Humanoid körs ändras
RBXScriptSignal Touched (Instance touchingPart , Instance humanoidPart)	Händer när en av Humanoid's extremiteter kommer i kontakt med ett annat BasePart