Labo Pong

Inleiding

Robin en ik hebben één van de allereerste arcadegames uitgewerkt met een unieke twist: Pong dat werd uitgegeven door Atari in 1972. Hoewel het concept identiek is als het originele spel hebben we de spelbesturing compleet aangepakt. Pong werd klassiek gespeeld met twee toetsen (op en neer), of een hendeltje die een soortgelijke werking had. Daar tegenwoordig iedereen thuis een generieke controller heeft liggen, zijnde van een Playstation of Xbox, of Nitendo, hebben we besloten om hiervoor functionaliteit in te bouwen. Daarbovenop kan de andere speler het spel bedienen met een Leap Motion zodat geen fysiek contact met een controller nodig is. Onze uitwerking van Pong kan dus gespeeld worden door twee spelers: één speler die speelt door zijn hand op of neer te bewegen, de andere door met de generieke controller hetzelfde te doen.

GameControlPlus is een library dat allerlei controllers en diverse apparaten kan gaan aansturen. Dit Apparaten variëren van simpele muizen naar joysticks tot gamepads. Deze library kan niet alleen zelf het aantal knoppen en sensoren gaan detecteren, maar ook de gepaste waardes retourneren. Zo zullen knoppen simpele 'pressed' events bevatten, en sliders een bepaald bereik hebben.

Omdat we ons project in slechts 15 minuten kunnen uitleggen, leveren we al een deel van onze broncode aan. De code hiervoor kan je in het bronbestand 'Pong.pde' vinden.

Uitwerking

Stap 1: connecteren van controllers

Deze stap bestaat uit twee delen: één voor elke controller. Sedert de bibliotheken voor beide controllers geïmporteerd zijn kunnen we allereerst de objecten gaan initialiseren. De LeapMotion kunnen we gaan initialiseren door een nieuwe instantie aan te maken van 'LeapMotionP5', net zoals we in een eerdere laboles gezien hebben. Om het gamepad aan te sturen moet echter eerst een object aangemaakt worden waarin alle apparaten die de ControlIO bibliotheek ondersteund worden ondergebracht. Dit doen we door een de ControlIO.getInstance(this) functie toe te passen op een ControlIO object. Eens dat gedaan is kunnen we een object 'ControlDevice' gaan maken en daarin de gewenste controller steken. Dit kan op basis van een unieke controller ID/string, of door een systeemafhankelijke identifier. Je kan een lijst opvragen van compatibele, geconnecteerde, apparaten met behulp van de ControlDevice.getInputs() functie. In ons voorbeeld draagt onze controller ID 6.

Connecteer nu beide controllers.

```
//Library objects
ControlIO controlIO;
ControlDevice gamepad;
LeapMotionP5 leap;
```

```
public void setup()
{
  controlIO = ControlIO.getInstance(this);
  leap = new LeapMotionP5(this);
  gamepad = controlIO.getDevice(6);
```

Stap 2: aansturen van knoppen en bewegen van de playerbars

De status van de knoppen op een controlIO apparaat kunnen zonder enige configuratie worden opgevraagd. Het zijn booleans die waar zijn als een bepaalde knop wordt ingedrukt. Als voorbeeld nemen we de gamehat (één fysieke knop links, die 4 richtingen kan gaat voorstellen). Om te kijken of de rechterzijde van deze knop wordt ingeduwd gaan we volgende functie in een controlestructuur plaatsen: **gamepad.getHat(ID).RICHTING().**

```
if(gamepad.getHat(0).right())
```

Nu we weten hoe deze knoppen worden aangestuurd, kunnen we onze playerbars gaan verplaatsen. Allereerst maken we een globale variabele aan waarin de positie van elke speler wordt in bijgehouden. Omdat deze variabele 2 punten zal moeten gaan bijhouden, raden we dan ook een PVector of array aan. Speler 2 kan, in deze oefening, in vier richtingen gaan bewegen. Bij verticale beweging zal de bar met 5 pixels bewegen, bij laterale beweging is dit slechts 1 pixel. U bent uiteraard vrij om deze snelheid zelf te kiezen.

De Leap Motion waardes vertalen we gewoon bij elke draw-cycle, de waarde in leap.getPosition is namelijk in al functie van de schermgrootte.

```
if(hands.size() == 1)
{
    for (Hand h : hands) { p1Pos.y = leap.getPosition(h).y; }
}
```

Nu de posities van beide spelers gekend zijn, en bij elke draw-cycle bijgewerkt worden, kunnen we dit gaan tekenen.

```
//Render player bars
fill(p1Color);
rect(p1Pos.x, handPos.y-p1Size.y/2, p1Size.x, p1Size.y);
fill(p2Color);
rect(p2Pos.x, p2Pos.y-p2Size.y/2, p2Size.x, p2Size.y);
```

Stap 3: bal beweging

Nu de spelers zich kunnen bewegen is het tijd om het spel interactief te maken. We introduceren dan ook een bal die zich in alle richtingen kan gaan voortbewegen. In essentie is dit niets meer dan een cirkel (eigenlijk een ellips) met een vooraf gedefinieerde radius waarvan de as-waardes bij elke draw-cycle lineair veranderen om zo de illusie van beweging te gaan creëren.

```
//Render ball
fill(128,128,128);
ellipse(ballPos.x, ballPos.y, ballDia, ballDia);
```

De snelheid zetten we hier ook vooraf vast. We raden een snelheid tussen 3 en 10 pixels per draw-cycle aan, afhankelijk van de draw-snelheid natuurlijk. Belangrijk hier is dat de snelheid op twee assen wordt vastgelegd! Er is namelijk zowel laterale als verticale beweging. Je kan het spel iets onvoorspelbaarder maken indien je de snelheid willekeur genereert. Let ook op dat je snelheid NOOIT hoger ligt dan de breedte van je playerbar, zo dadelijk zie je waarom.

```
ballPos.x += ballSpeed.x;
ballPos.y =+ ballSpeed.y;
```

Stap 3: botsingdetectie

Uiteraard is het zo dat de bal binnen de spelomgeving moet blijven. Natuurlijk mag de bal ook niet door de spelersbar gaan. Door een controlestructuur te generen die de hoek zal gaan inverteren indien de bal zich in een van deze toestanden bevindt kunnen we een illusie generen dat deze solide oppervlaktes zijn. De botsingdetectie voor top en dal zijn makkelijk te controleren, enerzijds kijken we of de balpositie groter is dan de maximale hoogte, anderzijds kijken we of de balpositie kleiner is dan de minimale hoogte. Indien dit zo is gaan we de positie gaan inverteren.

```
if (ballPos.y > height || ballPos.y < 0 )
{
  ballSpeed.y *= -1;
  ballPos.y += ballSpeed.y;
}
```

De laterale controle is iets ingewikkelder. Hier werken we namelijk met een beperkte oppervlakte die ook nog eens variabel is. Bekijk onderstaande code:

```
//Player 1 / left player collision detection
if (ballPos.x <= p1Pos.x && ballPos.x >= p1Pos.x-p1Size.x && ballPos.y > handPos.y-p1Size.y/2 && ballPos.y < handPos.y + p1Size.y/2)
{
  ballSpeed.x *= -1; // Invert ball
  ballPos.x += ballSpeed.x;
```

De eerste twee controles in deze structuur zijn nodig omdat de snelheid van de bal steeds hoger zal liggen dan 1 pixel per tijdseenheid. Dit wil zeggen dat de bal in de oppervlakte van een playerbar terecht zal komen. Dit is direct de reden waarom de snelheid nooit hoger mag liggen dan de oppervlakte van een speler, de kans dat de bal achter de speler terecht komt is dan ook aanzienlijk. Het tweede stuk van deze controlestructuur zal gaan kijken of de bal in de lengte van de speler terecht komt.