Collision Game Dokumentasjon Rapport

*«Ideen bak dette spillet spillet kommer fra et brettspill (med vinkel) som jeg spilte en del hos beste foreldrene mine. Målet med dette spillet var å få høyest mulig poengsum ved å lande kuler i kurver rundt på brettet. Dette brettet hadde ca 20˚ helling.»*

Spillet jeg har laget «Collision Game» bruker basic gravitasjons kraft i tilleg til kollisjonen. For å oppdatere denne hastigheten og posisjonen bruker jeg disse formlene:

//FORMULA: vf = v0 + a \* t

Vector3 velocity = new Vector3(vx, vy) + gravVector \* Time.fixedDeltaTime;

//FORMULA: x(t) = x0 + v0 \* t + 1/2 \* a \* t^2

Vector3 newPosition = transform.position + velocity \* Time.deltaTime - 0.5f \* gravVector \* Time.deltaTime \* Time.deltaTime;

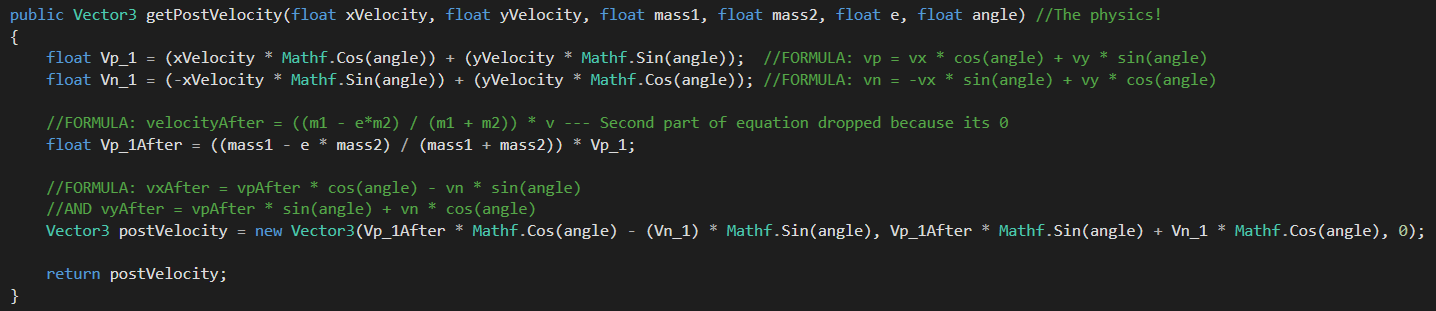
All fysikk og oppdatering skjer etter ballen blir «dropped», ved å trykke på Space knappen. Da kjører FixedUpdate() funksjonen.

For å oppdage kollisjon bruker jeg noe som kalles SphereCast, hvor stråler blir sendt ut fra spiller ballen, og når disse strålene merker at det skjer en kollisjon blir selve kollisjons koden kjørt.

if (Physics.SphereCast(transform.position, playerRadius, directionVector, out hit, directionVector.magnitude))

Vinkelen mellom spiller ballen og kollisjons objektet blir kalkulert ved hjelp av denne kodebiten:

angle = Mathf.Deg2Rad \* Vector3.Angle(Vector3.right, hit.normal);

Ved hjelp av denne vinkelen kan hastigeten og retningen etter kollisjonen blir kalkulert. For å finne ut av dette har jeg laget en funksjon kalt getPostVelocity() for finner ut av dette.

Her blir hastigheten langs «lineOfAction» kalkulert ved hjelp av formelen **under**, hvor lineOfAction er linja mellom de to objektene som kolliderer.

Hastigheten til normalen av denne trengs også å bli utregnet, og det blir gjort med denne formelen:

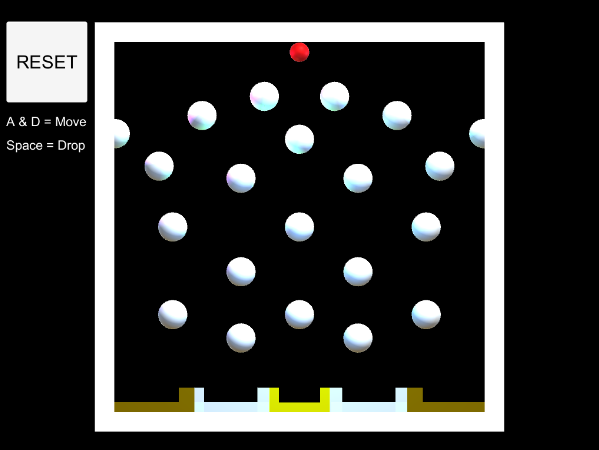
Det trengs også å finne ut hastigheten langs lineOfAction etter kollisjonen, noe som blir gjort med denne formelen:

Noen av formlene blir litt forkortet siden det er bare *en* av partene som flytter seg etter kollisjonen, nemlig spiller ballen.

Det gjenstår å finne farten i de forskjellige retningene, som i dette tilfellet er X og Y retningen, siden spillet bruker to dimentional fysikk og ikke tre dimentional, noe som hadde gjort det en del mer komplisert.

Tegnet «’», beskriver at det er komponenten etter kollisjonen har skjedd.

X og Y komponentene blir returnert fra denne funksjonen og brukt til å oppdatere hastigheten og posisjonen til ballen etter kollisjonen har skjedd.

Spillet endte opp med å se slik ut. For mer info om hva forskjellige variabler og andre elementer er og blir brukt til, så står det beskrevet i kommentarene i koden.