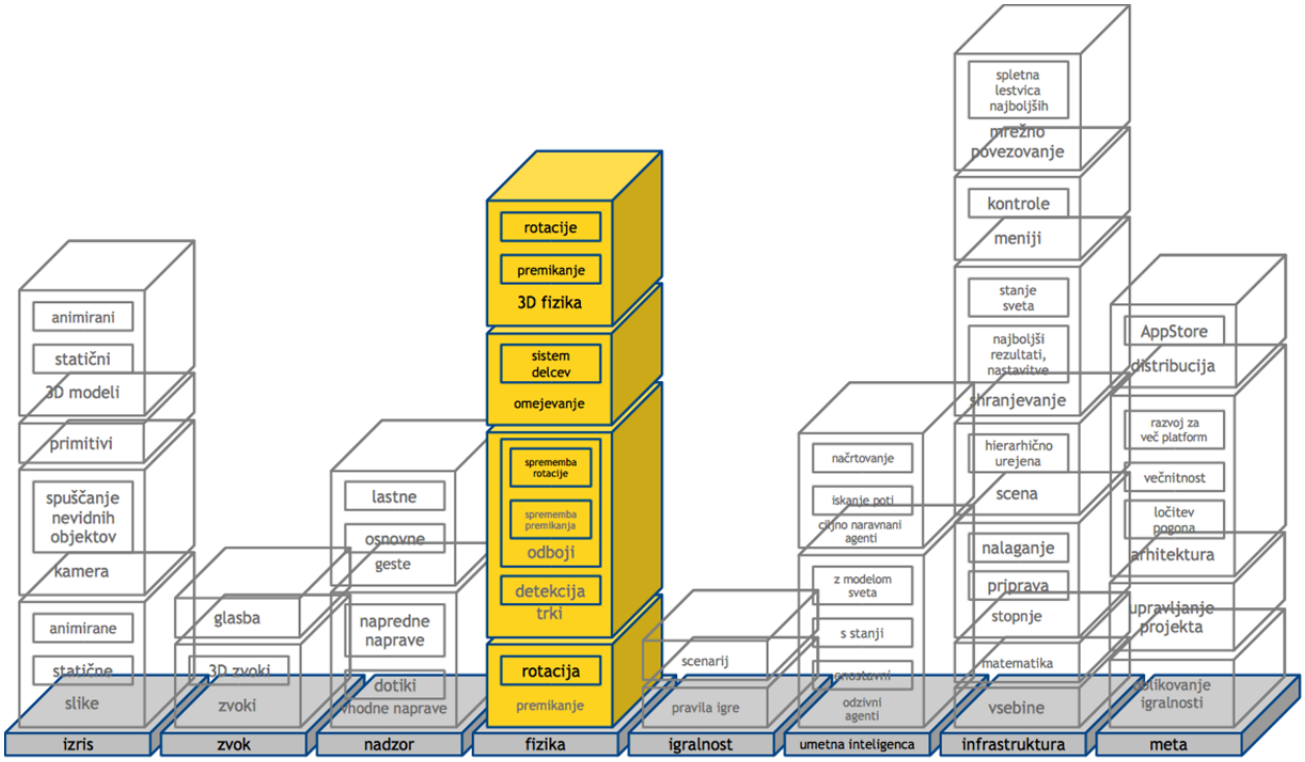
Sklop 8


animirani
statični
3D modeli
primitivi
spuščanje nevidnih objektov
kamera
animirane
statične
slike
izris
glasba
3D zvoki
zvoki
zvok
lastne
osnovne geste
napredne naprave
dotiki
vhodne naprave
nadzor
rotacije
premikanje
3D fizika
sistem delcev
omejevanje
sprememba rotacije
sprememba premikanja
odboji
detekcija trki
rotacija
premikanje
fizika
scenarij
pravila igre
igralnost
načrtovanje
iskanje poti
cilj no naravnani agenti
z modelom sveta
s stanji
odzivni agenti
umetna inteligenca
spletna lestvica najboljših
mrežno povezovanje
kontrole
meniji
stanje sveta
najboljši rezultati, nastavitve
shranjevanje
hierarhično urejena
scena
nalaganje
priprava
stopnje
matematika
vsebine
infrastruktura
AppStore
distribucija
razvoj za več platform
večnitnost
ločitev pogona
arhitektura
upravljanje projekta
oblikovanje igralnosti
meta

Rotacija

Izris zarotiranih slik

Veliko vizualnih učinkov in načinov igralnosti zahteva izris z rotacijo. Pri prvih gre običajno zgolj za kozmetično spremembo, kjer pri draw klicu *SpriteBatch* izriše teksturo zarotirano za nek kot. Če hočemo, da se objekt vrti s konstantno hitrostjo, je najenostavnejše uporabiti *totalGameTime* vrednost spremenljivke *gameTime*:

```
float angleSpeed = M_PI * 2; // 2 pi radians (360 degrees) per second
float angle = gameTime.totalGameTime * angleSpeed;
Vector2 *origin = [Vector2 vectorWithX:texture.width / 2.0f y:sprite.texture.height / 2.0f];

[spriteBatch draw:texture to:position fromRectangle:nil tintWithColor:[Color white]
         rotation:angle
           origin:origin
    scaleUniform:1 effects:SpriteEffectsNone layerDepth:0];
```

Pri tem se rotacija izvaja glede na *origin*. Če uporabimo privzeto vrednost *[Vector2 zero]*, se bo tekstura vrtela okoli zgornjega levega kota, če pa uporabimo točko na sredini teksture, bomo dobili najobičajnejši želeni učinek. Ne pozabimo, da vrednost pod *to:* določa, kje na ekranu bo izrisana točka *origin* teksture (le-ta ob rotaciji ostaja nepremična).

Fizikalno obravnavanje rotacije

Če hočemo k rotaciji pristopiti več kot kozmetično, moramo vzporedno s spremenljivkami in fizikalnimi izračuni za premikanje obravnavati tudi rotacijo.

Vrednosti pri tem so analogne premikanju, le da namesto večdimenzionalne vrednosti premikanja potrebujemo samo enodimenzionalne vrednosti rotacije (v 2D prostoru):

Premikanje	**Rotacija**
pozicija ($\vec{x}$)	orientacija (φ)
hitrost ($\vec{v}$)	kotna hitrost (ω)
pospešek ($\vec{a}$)	kotni pospešek (α)
masa (m)	vztrajnostni moment (J)
gibalna količina ($\vec{G}$)	vrtilna količina (Γ)
sila ($\vec{F}$)	navor (M)

Podobno velja za enačbe, ki jih potrebujemo za enostavno spremembo gibanja:

Premikanje	**Rotacija**
$\vec{x} = \vec{x}_0 + \vec{v}\Delta t$	$\varphi = \varphi_0 + \omega\Delta t$
$\vec{v} = \vec{v}_0 + \vec{a}\Delta t$	$\omega = \omega_0 + \alpha\Delta t$
$\vec{a} = \vec{a}_0 + \frac{F}{m}$	$\alpha = \alpha_0 + \frac{M}{J}$

Tudi pri trkih veljajo podobni zakoni (zakon o ohranitvi vrtilne količine). Še vedno nas zanima izmenjava sunka sile v točki trka, vendar moramo po novem upoštevati, da se zaradi vrtenja telesa

točke oddaljene od centra rotacije (masnega središča) gibajo hitreje. Hitrost točke telesa izven središča je povezana z ročico (razdaljo med središčem in točko):

$$v_\varphi = |\vec{r}|\omega$$

S tem da je usmerjena pravokotno na ročico (tangenta krožnice gibanja točke):

$$\vec{t} \perp \vec{r}$$

$$\vec{v}_\varphi = \vec{t} v_\varphi$$

Nova hitrost točke trka je tako:

$$v_{trk} = (\vec{v} + \vec{v}_\varphi) \cdot \vec{n}_{trk}$$

Spomniti se moramo, da vsaka sila ustvarja navor glede na ročico ($\vec{r}$) od središče mase telesa do točke, kjer sila deluje:

$$M = \vec{F} \times \vec{r}$$

Upoštevati moramo, da sila usmerjena proti središču ne bo ustvarila navora. Za rotacijo je namreč sprememba odvisna od tega, pod kakšnim kotom deluje izmenjava sil napram ročici (pravokotno nanjo bo sila trka ustvarila največji navor):

$$r = |\vec{r} \times \vec{n}|$$

Tako lahko sestavimo popravljeno enačbo izračuna sunka sile vzdolž normale trka (pri tem so hitrosti v nove hitrosti z dodatkom zaradi rotacije):

$$F\Delta t = \frac{-(e+1)(v_1 - v_2)}{\frac{1}{m_1} + \frac{1}{m_2} + \frac{r_1^2}{J_1} + \frac{r_2^2}{J_2}}$$

Podobno kot s sunkom sile spremenimo hitrost telesa:

$$v_{kon} = v_{zač} + \frac{F\Delta t}{m}$$

moramo zdaj popraviti še kotno hitrost:

$$\omega_{kon} = \omega_{zač} + \frac{|\vec{r} \times \vec{n}| \cdot F\Delta t}{J}$$

Naloga: rotacija

Izris predmetov z rotacijo je s *SpriteBatch*om zelo enostaven. Naprednejši klici *draw* namreč sprejmejo parameter, za koliko radianov naj bo izrisana slika zarotirana. Tako lahko najpreprostejše vrtenje naredimo že, če namesto ničle v parameter *rotation* pošljemo *gameTime.totalGameTime*.

Če po drugi strani želimo pripraviti podlago za fizikalno pravilne rotacije, potrebujemo podobne spremenljivke kot pri premikanju (le da je rotacija v 2D enodimenzionalna lastnost). Pot postane kót in namesto hitrosti imamo kotno hitrost. Tako izdelamo protokole za *Rotation*, *AngularVelocity* in *Rotatable*, ki združuje oboje.

V fizikalni pogon tako dodamo izračun, ki vsem objektom, ki implementirajo *Rotatable,* spremeni kót rotacije glede na kotno hitrost.

Naloga: sprememba rotacije

Za fizikalno pravilno spremembo rotacije pri trkih moramo v izračun vpeljati količine kot so vztrajnostni moment, kotni pospešek in navor. Ker pa je navor odvisen od ročice, ki jo predstavlja vektor med središčem rotacije in točko trka, moramo po novem pri detekciji trka izračunati tudi slednjo. Na podlagi le-te lahko dodatno popravimo hitrosti obeh predmetov (zdaj nas zanimajo hitrosti lokalnega dela telesa v točki trka) ter pri izračunu impulza sile upoštevati vztrajnostna momenta teles.

Na ta način z novo izračunanim impulzom poleg posodobitve hitrosti centra mase teles ustrezno spremenimo še kotne hitrosti.

3D fizika

Svet 3D fizike je veliko področje, ki se ga bomo tukaj zgolj dotaknili. Kar se tiče premikanja smo vse enačbe do sedaj izpeljali vektorsko in delujejo enako v dveh ali treh dimenzijah. Tako lahko naredite preprost 3D fizikalni pogon že tako, da dosedanje 2D pozicije, hitrosti in pospeške nadomestite s 3D vektorji.

Celo detekcija trkov med krogi v ravnini in kroglami v prostoru je enaka. Fizika trkov med z osmi poravnanimi kvadri zgolj doda še eno dimenzijo preverjanja glede na poravnane pravokotnike. Pri poljubnih kvadrih in na splošno konveksnih poliedrih imamo namesto polravnin prave ravnine v prostoru. Od tu naprej se kompleksnost samo še potencira. Še posebej rotacije so v 3D precej nehvaležne, saj moramo namesto preprostih skalarnih vrednosti začeti operirati s kvaternioni (4D vektorji za predstavitev orientacije v 3D), npr. vztrajnostni moment postane tenzor (matrika) itd.

Vsekakor se je v napredna področja 3D fizike dobro spustiti šele, ko obvladamo dogajanje v dveh dimenzijah, kjer je vse izračune veliko lažje ponazoriti.

Naloga: 3D premikanje

Izkaže se, da je sprememba premikanja pri prehodu iz 2D v 3D zelo enostavna. Razlog leži v tem, da je že vsa matematika za 2D fiziko sestavljena kot kombinacija izračunov v dveh enodimenzionalnih oseh. Tako je zelo enostavno vsem izračunom poleg osi x in y dodati še tretjo os z.

V praksi to pomeni, da vse vektorje tipa Vector2 zamenjamo z Vector3. Izračuni pozicije glede na hitrost, pospeške in sile se bodo avtomatično izšli tudi v 3D. Podobno velja pri detekciji trkov, le pri z osmi poravnanimi telesi moramo dodati kodo, ki uporabljene račune izvaja tudi po z-komponenti vektorjev.