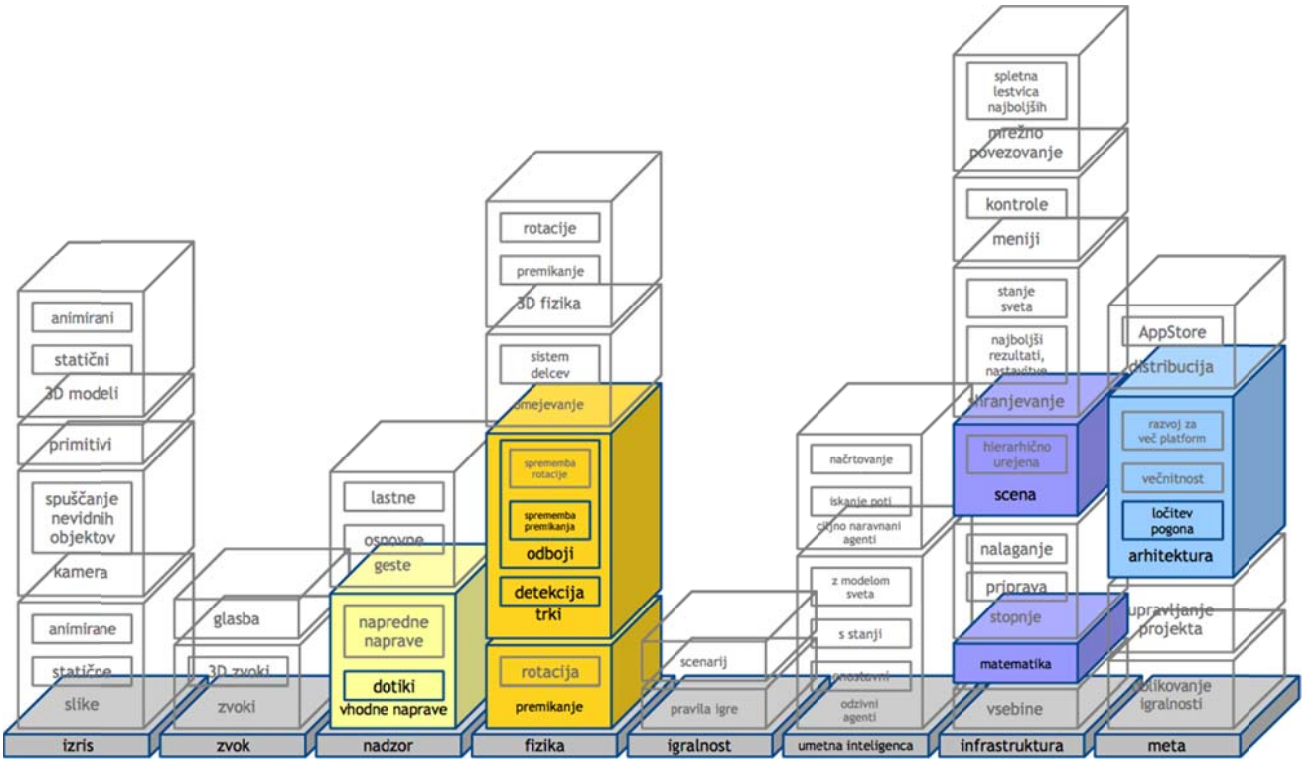
Sklop 3 (nadaljevanje)



Trki

Ko se predmeti začnejo premikati po sceni, prej ali slej pride do trenutka, ko se dva objekta srečata in kot duhca preletita eden skozi drugega. Običajno na tem mestu želimo zaznati trk, da se lahko igra ustrezno odzove. Ko Pacman pride do krogca, mora namreč izginiti s scene. Če pa pride do stene, se mora ob njej tudi zaustaviti. Na ta način ločimo delo s trki na njihovo detekcijo in reakcijo – odboje.

Trke vedno obravnavamo na paru objektov. V najosnovnejši različici se čez sceno sprehodimo z dvema for-zankama, čeravno bomo na ta način vsak par računali dvakrat. Nujno pa se moramo izogniti vsaj preverjanju trka s samim seboj.

```
for (id item1 in scene) {
    for (id item2 in scene) {
        if (item1 != item2) {
            [Collision collisionBetween:item1 and:item2];
        }
    }
}
```

Detekcija trkov

Zaznavanje trkov je stvar matematike, natančneje geometrije. Ko smo v simulacijskem koraku izračunali nove pozicije predmetov na sceni, nas zanima, ali so v tem novem stanju kateri od predmetov trčili. V 2D prostoru je to enako vprašanju, ali se dva lika prekrivata.

Jasno je torej, da morajo objekti, s katerimi želimo zaznati trke, imeti definirano geometrično obliko. Da so izračuni prekrivanja čim manj potratni, uporabimo čim bolj preproste, začenši s krogi (particle), ravninami (half-plane) in pravokotniki (rectangle) ter na drugi strani s poljubnimi konveksnimi večkotniki (convex). Marsikaj se še posebej poenostavi, če uporabimo z osmi poravnane različice objektov (axis-aligned half-plane, axis-aligned rectangle).

Izbira algoritma za trk

Da bo fizikalni pogon vedel, kakšno obliko za namene trkov uporablja določen objekt, zopet pripravimo protokole. *IParticleCollider* predstavlja okrogel delec z določenim polmerom. *IHalfPlaneCollider* je v 2D svetu postavljena premica, ki svet razdvoji na dve polovici (polravnini). Trki se torej dogajajo z eno od polravnin. Z osjo poravnana različica je *IAxisAlignedHalfPlaneCollider*. Prav tako sta tu *IAxisAlignedRectangleCollider* (pravokotnik s širino in višino) ter *IRectangleCollider*, ki je lahko še poljubno zarotiran. *IConvexCollider* je najbolj zapleten od likov in vsebuje podatke o poljubnem konveksnem večkotniku, ki lahko precej natančno obriše dejansko obliko predmeta. Še več, niti ni nujno, da gre za zaprt večkotnik; če mu kakšna stranica manjka, se v tisti smeri razprostira v neskončnost.

Poudarimo še, da je IRectangleCollider lahko samo posebna verzija IConvexColliderja, ki ima oglišča postavljena v pravokotnik. Prav tako je IAxisAlignedRectangleCollider samo IRectangleCollider z rotacijo nastavljeno na 0. Torej bi lahko z IConvexColliderjem opisali vse prej omenjene oblike, kakor tudi IHalfPlaneColliderja, če v IConvexCollider damo samo eno stranico. Za specializirane tipe se lahko odločimo zato, da hitreje in bolj semantično opišemo geometrijsko obliko objekta na sceni ter tudi za to, da bomo v fizikalnem pogonu za bolj preproste oblike lahko uporabili učinkovitejše ali preprostejše, lažje razumljive algoritme. Recimo detekcija trka s tlemi, opisanimi z enačbo y=0, je zelo enostavna, saj samo gledamo, če je y koordinata najnižje točke predmeta večja od nič.

V naši metodi *collisionBetween:item1 and:item2* zdaj preprosto pogledamo, kakšnim protokolom ustrezata podana predmeta in pokličemo najustreznejši algoritem, ki bo izračunal detekcijo in odboje za konkretno podani geometrijski obliki.

```
+ (void) collisionBetween:(id)item1 and:(id)item2 {
        id<IParticleCollider> item1Particle = [item1
           conformsToProtocol:@protocol(IParticleCollider)] ? item1 : nil;

        id<IAxisAlignedHalfPlaneCollider> item1AAHalfPlaneCollider = [item1
           conformsToProtocol:@protocol(IAxisAlignedHalfPlaneCollider)] ? item1 : nil;

        ... // We do this for all supported protocols and repeat the same for the other item.

        if (item1Particle && item2Particle) {
                [ParticleParticleCollision collisionBetween:item1Particle and:item2Particle];
        }
        else if (item1Particle && item2AAHalfPlaneCollider)  {
                [ParticleAxisAlignedHalfPlaneCollision collisionBetween:item1Particle and:item2AAHalfPlaneCollider];
        }
        else if (item2Particle && item1AAHalfPlaneCollider)  {
                [ParticleAxisAlignedHalfPlaneCollision collisionBetween:item2Particle and:item1AAHalfPlaneCollider];
        }
        else ... // Repeat for all supported protocols.
}
```

Deli algoritma

Kot smo rekli, se delo s trki najprej deli na detekcijo, ki pove samo ali je do trka prišlo (geometrijski del, vrne binarno vrednost YES/NO) in na računanje odboja (fizikalni del, spremeni pozicije in hitrosti samih objektov). Na tak način tudi organiziramo vsakega od naših razredov za delo z dvema konkretnima oblikama predmetov, recimo za razred *ParticleParticleCollision*:

```
+ (BOOL) detectCollisionBetween:(id<IParticleCollider>)particle1 and:(id<IParticleCollider>)particle2;

+ (void) resolveCollisionBetween:(id<IParticleCollider>)particle1 and:(id<IParticleCollider>)particle2;
```

Poleg tega si pripravimo še metodo, ki jo kličemo v večini primerov, ko hočemo izvesti trk v celoti in ne le njeno detekcijo. Ta metoda torej pokliče resolve v primeru, da detect vrne YES.

```
+ (void) collisionBetween:(id<IParticleCollider>)particle1 and:(id<IParticleCollider>)particle2 {
        if ([ParticleParticleCollision detectCollisionBetween:particle1 and:particle2]) {
           [ParticleParticleCollision resolveCollisionBetween:particle1 and:particle2];
        }
}
```

Naloga: Detekcija trkov

S postavljeno arhitekturo fizikalnega pogona nam preostane le še jedro: implementacija algoritmov za detekcijo prekrivanja dveh geometričnih oblik. Kot rečeno moramo za vsak različen par oblik spisati svoj algoritem; velja pa tudi, da recimo algoritem za detekcijo prekrivanja kroga in konveksnega večkotnika že sam po sebi vsebuje detekcijo krog-pravokotnik in krog-polravnina. Pravokotnik je namreč konveksni večkotnik, detekcija pa ne more delovati brez tega, da bi vsako od stranic večkotnika obravnavali kot polravnino.

Je pa vsekakor res, da, če kompleksnejših oblik ne potrebujemo, so algoritmi med bolj specifičnimi oblikami, recimo detekcija krog-krog ali med dvema z osmi poravnanima pravokotnikoma, veliko bolj preprosti kot recimo prekrivanje dveh poljubnih konveksnih večkotnikov (namig: separating axis-theorem).

Reakcija na trk

Relaksacija

Tudi reakcijo na trk razdelimo na dva dela. Najprej je tu relaksacija oziroma razrešitev stanja prekrivanja. Igralčev lik, ki je zašel v steno, v tem koraku premaknemo nazaj do točke, ko se stene ravno dotika. Podobno ravnamo z vsemi pari objektov. Odvisno od geometrije izračunamo, za koliko jih moramo razmakniti, ter to ustrezno storimo. Pri tem pazimo, da premaknemo predmeta v sorazmerju z njihovimi masami, saj bi bilo čudno, če bi žoga, ki se z veliko hitrostjo zadane v recimo tovornjak, zaradi tega koraka uspela premakniti več kot tisočkrat težji predmet.

```
+ (void) relaxCollisionBetween:(id)item1 and:(id)item2 by:(Vector2*)relaxDistance {
	// We have to ask, how far we move each item.
	// The default is each half way, but we try to take
	// the mass of the colliders into account, if items have mass.
	float relaxPercentage1 = 0.5f;
	float relaxPercentage2 = 0.5f;

	// Determine mass of the colliders. If an item has no mass it is considered static,
	// so we should move only the other one. If both have mass, we move them reciprocal to their mass.
	// So a heavier item will move a little and a lighter item more.
	id<IMass> itemWithMass1 = [item1 conformsToProtocol:@protocol(IMass)] ? item1 : nil;
	id<IMass> itemWithMass2 = [item2 conformsToProtocol:@protocol(IMass)] ? item2 : nil;

	if (itemWithMass1 && itemWithMass2) {
		float mass1 = itemWithMass1.mass;
		float mass2 = itemWithMass2.mass;
		relaxPercentage1 = mass2 / (mass1 + mass2);
		relaxPercentage2 = mass1 / (mass1 + mass2);
	} else if (itemWithMass1) {
		relaxPercentage1 = 1;
		relaxPercentage2 = 0;
	} else {
		relaxPercentage1 = 0;
		relaxPercentage2 = 1;
	}

	// Now we need to turn the percentages into real distances.
	id<IPosition> itemWithPosition1 = [item1 conformsToProtocol:@protocol(IPosition)] ? ((id<IPosition>)item1) : nil;
	id<IPosition> itemWithPosition2 = [item2 conformsToProtocol:@protocol(IPosition)] ? ((id<IPosition>)item2) : nil;

	if (itemWithPosition1) {
		[itemWithPosition1.position subtract:
			[Vector2 multiply:relaxDistance by:relaxPercentage1]];
	}

	if (itemWithPosition2) {
		[itemWithPosition2.position add:
			[Vector2 multiply:relaxDistance by:relaxPercentage2]];
	}
}
```

Na tem mestu se pri določenih igrah že lahko ustavimo. Da Pacman ne more skozi steno, je to dovolj; fizikalnega odboja nam ni treba računati. Da pa se bo žogica pri Arkanoidu odbila od opeke, potrebujemo še drugi korak.

Odboj

Do zdaj smo se ukvarjali zgolj z geometričnimi problemi prekrivanja in iskanja pravih vektorjev ter razdalj. Šele na tem mestu zares pridemo do fizike.

Pri trku pride do delovanja sil med dvema telesoma, ki se gibata ena proti drugemu. V nekem trenutku se dotakneta in izmenjata energijo, kar vidimo posledično v spremembi vektorjev hitrosti.

Zelo pomembna lastnost pri trkih je, da se v sistemu ohranja skupna gibalna količina vseh udeleženih teles. Spomnimo se, gibalna količina telesa je produkt vektorja hitrosti z maso telesa.

$$\vec{G} = m\vec{v}$$

Torej za trk dveh teles velja

$$\vec{G}_{1zač} + \vec{G}_{2zač} = \vec{G}_{1kon} + \vec{G}_{2kon}$$

Dodatno zaradi izreka o gibalni količini velja, da je sprememba gibalne količine enaka sunku zunanjih sil na to telo. Sunek sile izračunamo kot zmnožek sile in časa, v katerem ta sila deluje.

$$\vec{G}_{kon} = \vec{G}_{zač} + \vec{F}\Delta t$$

Zaradi tretjega Newtonovega zakona, zakona o vzajemnem učinku, vemo tudi, da, če med trkom prvo telo deluje z neko silo na drugega, bo drugo telo delovalo na prvega z nasprotno enako silo

$$\vec{F}_{12} = -\vec{F}_{21}$$

Torej gre v našem trku dveh teles za sistem dveh enačb

$$\vec{G}_{1kon} = \vec{G}_{1zač} + \vec{F}\Delta t$$
$$\vec{G}_{2kon} = \vec{G}_{2zač} - \vec{F}\Delta t$$

Normala trka

Preden se vržemo v izračun sunka sile, se lahko rešimo vektorske notacije, če razdelimo delovanje sil med trkom na dve komponenti. V smeri tangente trka, torej linije, ki ločuje telesi vsako na svojo stran, deluje zgolj sila trenja, zaradi hrapavosti obeh površin. Izmenjava sil trka, o katerih govorimo v tem delu, pa deluje zgolj pravokotno na tangento (v smeri normale trka).

Tako nas zanima zgolj komponenta gibalne količine teles v smeri normale trka in posledično le hitrost telesa v smeri normale. Izračun velikosti te komponente je preprost, saj gre za skalarni produkt vektorja hitrosti z enotskim vektorjem normale.

$$v_{trk} = \vec{v} \cdot \vec{n}_{trk}$$

Od tu naprej se bomo ob skalarni količini hitrosti nanašali na komponento vektorja hitrosti telesa v smeri normale trka.

Prožnost trka

Da rešimo zgornji sistem dveh enačb s tremi neznankami (v_{1kon}, v_{2kon}, F) rabimo dodatno enačbo. Trki se lahko gibljejo med popolnoma prožnimi, pri katerih velja, da se v sistemu ohrani skupna kinetična energija in popolnoma neprožnimi, pri katerih se po trku objekta "zlepita" ter nadaljujeta z gibanjem s popolnoma enako hitrostjo.

Kje med tema dvema ekstremoma se nahaja dejanski izračun trka, pove restitucijski koeficient trka.

$$e = \frac{v_{2kon} - v_{1kon}}{v_{1zač} - v_{2zač}}$$

Koeficient je konstanta za dva tipa teles in ga moramo imeti podanega vnaprej. Vrednost 1 pomeni popolnoma prožen trk, pri 0 gre za neprožen trk.

Zdaj lahko vse enačbe uporabimo in izračunamo sunek sile.

$$F\Delta t = \frac{-(e+1)(v_1 - v_2)}{\frac{1}{m_1} + \frac{1}{m_2}}$$

S to količino lahko končno spremenimo hitrosti objektov v smeri normale trka. Ker je sprememba hitrosti pospešek

$$v_{kon} = v_{zač} + a\Delta t$$

pospešek pa je po drugem Newtonovem zakonu enak

$$a = \frac{F}{m}$$

dobimo končni enačbi za novi hitrosti teles po trku.

$$v_{1kon} = v_{1zač} + \frac{F\Delta t}{m_1}$$
$$v_{2kon} = v_{2zač} + \frac{F\Delta t}{m_2}$$

V kodi pa ustrezni deli izračuna odboja izgledajo takole:

```
float speed1 = [Vector2 dotProductOf:item1.velocity with:collisionNormal];
float speed2 = [Vector2 dotProductOf:item2.velocity with:collisionNormal];
float speedDifference = speed1 - speed2;

float mass1inverse = 1.0f / item1.mass;
float mass2inverse = 1.0f / item2.mass;

float impact = -(cor + 1) * speedDifference / (mass1inverse + mass2inverse);

[item1.velocity add: [Vector2 multiply:collisionNormal by:impact * mass1inverse]];
[item2.velocity subtract: [Vector2 multiply:collisionNormal by:impact * mass2inverse]];
```

Naloga: sprememba premikanja pri odboju

Če želimo fizikalno simulirati odboj po zaznavanju trka, moramo izračunati sunek sile ter glede na njega spremeniti hitrosti premikanja teles.

Da lahko to storimo, moramo ob izračunu vedeti hitrosti in mase teles ter restitucijski koeficient trka. Ta je načeloma definiran za par predmetov, lahko pa si poenostavimo stvari in ga definiramo kar za vsak objekt posebej, skupni koeficient, ki ga upoštevamo pri trku, pa izračunamo recimo kot zmnožek vrednosti obeh teles.

Dodatno lahko v izračunih upoštevamo nepremična telesa, pri katerih predpostavimo, da je njihova hitrost enaka 0, masa pa neskončna (inverz mase je enak 0).