Primjena matematike u izradi računalnih igara

Golubić, Kristian

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Kristian Golubić

PRIMJENA MATEMATIKE U IZRADI RAČUNALNIH IGARA

Diplomski rad

Voditelj rada:
doc. dr. sc. Maja Starčević

Zagreb, 2014.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

U radu se opisuje nekoliko različitih tipova problema s kojima se programer može susresti prilikom izrade igre, a u kojima mu može pomoći matematika.

Najprije je potrebno objekte smjestiti na zaslon monitora. Prilikom takvog smještanja koriste se točke kako bismo prikazali položaj objekta, a točke smještamo u odabrani koordinatni sustav. Na zaslonu računala bit će prikazana projekcija trodimenzionalnog objekta na dvodimenzionalnu ravninu. Memorijski je skupo stvarati iznova objekte u različitim položajima. Zato se koriste transformacije objekata. Prilikom transformacije objekata dobivaju se nove točke koje određuju objekt. U radu su objašnjene transformacije skaliranja, translacije, rotacije, zrcaljenja i smicanja te su objašnjena njihova svojstva kao i primjena tih transformacija u igrama.

Računalne igre su većinom interaktivne, odnosno objekti u računalnim igrama utječu jedni na druge. Potreban je alat koji će prepoznati dodiruju li se dva objekta, odnosno trebaju li utjecati jedan na drugoga. Ograničenja koja nama dopuštaju da utječemo na druge objekte u realnom svijetu, u virtualnom svijetu moramo sami stvoriti. Kako su objekti složenog oblika, traženje presjeka složen je problem. U radu su opisana dva načina smještanja složenih objekata u jednostavnije i testiranje presjeka jednostavnijih objekata. Dva objekta koja se najčešće koriste su sfere i takozvani AABB objekti. Prilikom određivanja postoji li presjek između dva objekta koristi se jednostavna analitička geometrija prostora i ravnine. Spomenuti su i drugi jednostavniji objekti kojima je moguće lako analitički odrediti presjeke.

Kada igramo neku igru i odredimo mjesto na koje želimo da nam se objekt pomakne, često se objekt najkraćim putem pomiče od trenutne pozicije do pozicije na koju ga želimo dovesti. Kako objekti u igraćem svijetu ne mogu sami razmišljati, moramo reći računalu kojim putem objekt treba ići. U radu je opisan problem najkraćeg puta kao i osnovne implementacije grafova u računalu. Detaljno je opisan Dijkstrin algoritam za rješavanje problema najkraćeg puta, a spomenuti su i još neki algoritmi koji se koriste pri rješavanju tog problema.

Strategija u igri vrlo je važna prilikom njezine izrade. Bitno je izraditi dovoljno zahtjevnu igru u kojoj neće svatko pobijediti, ali i dovoljno laku koja neće biti nepobjediva. U FPS i RTS igrama problem zahtjevnosti igara ne postoji, već se on javlja samo kod igara

na ploči i kartaških igara. Jedna od poznatih zahtjevnijih igara tog tipa je šah. Čovjek je kroz povijest pokušavao stvoriti stroj koji će biti nepobjediv u šahu. U radu je opisan minimax algoritam na primjeru igre križić-kružić koji računalu služi kako bi imao strategiju u savršeno informiranoj deterministčkoj igri za dva igrača. Opisana je i optimizacija minimax algoritma, alfa-beta podrezivanje, a spomenuti su i algoritmi za igre za više igrača koji nisu potpuno istraženi.

U mnogim igrama objekte pomičemo, na primjer pritiskom na strelicu prema gore. Time želimo primijeniti konstantnu translaciju prema gore, jednu po okviru, sve dok tipku ne pustimo. Također, u mnogim igrama objekte rotiramo pritiskom na tipku desno. Model je dobar u brzim igrama u kojima translacija koju postižemo tipkovnicom treba odmah utjecati na objekt. No, ako promatramo igru u kojoj to ne želimo jer se objekt u realnosti ne zaustavi odmah (na primjer, podmornica), već usporava kako bi se zaustavio, moramo stvoriti model koji će to omogućiti. U tome nam može pomoći fizika. U radu je objašnjeno kako odrediti položaj objekta koji se giba konstantnom brzinom te konstantnom akceleracijom. Također, opisan je problem kosog hica i lopte koja odskače.

Poglavlje 1

Transformacije

1.1 Koordinatni sustav

Euklid je u Elementima definirao točku kao ono što nema dijelova. Točka se može opisati kao poprečni presjek pravca ili sjecište dvaju pravaca. U računalnim igrama koristimo točke kako bi prikazali položaj objekata i kao osnovu za izgradnju tih objekata. Zasluga za otkriće Kartezijevog koordinatnog sustava kakav mi danas poznajemo pripisuje se francuskom matematičaru Reneu Descartesu. U Kartezijevom koordinatnom sustavu mjerimo položaj točke u odnosu na fiksnu točku, ishodište.

U $\mathbb{R}^2$ definiramo dva okomita brojevna pravca koji prolaze ishodištem, odnosno $x-$os i $y-$os. Položaj točke P određuje uređeni par (x, y), gdje je x udaljenost točke od $y-$osi, a y udaljenost točke od $x-$osi. Uređeni par (x, y) nazivamo koordinatom točke P. Dobivamo koordinatnu ravninu određenu osima x i y, odnosno xy koordinatnu ravninu.

U $\mathbb{R}^3$ definiramo tri međusobno okomita brojevna pravca koji prolaze ishodištem, odnosno $x-$os, $y-$os i $z-$os. Prepoznajemo odgovarajuće koordinatne ravnine xy (određena osima x i y), yz (određena osima y i z) i xz (određena osima x i y). Položaj točke P određuje uređena trojka (x, y, z), gdje je x udaljenost točke od yz ravnine, y udaljenost točke od xz ravnine te z udaljenost točke od xy ravnine. Uređenu trojku (x, y, z) nazivamo koordinatom točke P.

Pretpostavimo da želimo prikazati trodimenzionalni objekt na zaslonu računala. Najprije odabiremo xyz koordinatni sustav u $\mathbb{R}^3$ i to tako da je ishodište odabranog koordinatnog sustava središte video zaslona te da se xy ravnina odabranog koordinatnog sustava podudara s ravninom zaslona. Očito, na zaslonu računala bit će prikazana samo projekcija trodimenzionalnog objekta na dvodimenzionalnu xy ravninu.

1.2 Koordinatna matrica

Pretpostavimo da želimo stvoriti svijet u kojem se računalna igra odvija. Mogli bismo izgraditi sve geometrijske oblike za svaku sliku i za svaku lokaciju igre. Međutim, ukoliko se u slikama igre pojavljuju duplicirani objekti, memorijski je efikasnije izgraditi jednu kopiju za svaki pojedini objekt. Tada za svako pojavljivanje određenog objekta odredimo samo položaj i orijentaciju te prepustimo računalu smještanje objekta na sliku, odnosno zaslon. Drugi razlog za izgradnju jedne kopije svakog pojedinog objekta je taj da se objekti u igri uobičajeno kreću pa se njihovo postavljanje na određenu fiksnu poziciju ne čini praktično. Kako su objekti sastavljeni od niza točaka smještenih u Kartezijev koordinatni sustav, svaku od točaka koja određuje objekt želimo premjestiti na određeni način.

Neka u xyz koordinatnom sustavu, točke $P_1, P_2, \ldots, P_n$ određuju objekt. Koordinata točke P_i je uređena trojka (x_i, y_i, z_i), gdje je i prirodan broj između 1 i n. Konstruiramo matricu A tipa $3 \times n$. Stupci matrice A su koordinate točaka koje određuju objekt, dakle n je broj točaka koje određuju objekt. Matricu A definiranu s

$$A = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \\ y_1 & y_2 & \ldots & y_n \\ z_1 & z_2 & \ldots & z_n \end{bmatrix}$$

nazivamo koordinatnom matricom. Na ovaj je način koordinatnom matricom jedinstveno određen objekt.

Koordinate $(x_1, y_1, z_1), (x_2, y_2, z_2), ..., (x_n, y_n, z_n)$ se zajedno sa specificiranjem koji parovi točaka su spojeni dužinama spremaju u memoriju grafičkog prikaza.

Kada objekt transformiramo, a uvidjeli smo potrebu za stvaranjem jedinstvenog objekta i njegovom transformacijom unutar igre, dobivamo nove točke koje određuju objekt. Točke koje određuju transformirani objekt sada tvore novu koordinatnu matricu A' koja odgovara novom pogledu na objekt, odnosno transformiranom objektu. Dužine koje spajaju različite točke, i time potpuno određuju objekt, prilikom transformacije objekta također se mijenjaju te sada spajaju transformirane točke te određuju transformirani objekt.

1.3 Skaliranje

Prva transformacija koju možemo promatrati je skaliranje. Ono rasteže ili steže objekt tako da svaku koordinatu koja ga određuje množi odgovarajućim koeficijentom. Kako je svaka točka koja određuje objekt određena s tri koordinate, skalirati možemo s tri različita koeficijenta. Ukoliko su ti koeficijenti međusobno jednaki, govorimo o proporcionalnom, odnosno uniformnom skaliranju, a inače govorimo o neproporcionalnom, odnosno neuniformnom skaliranju. Iako koeficijent skaliranja može biti bilo koji realan broj, u računalnoj se

grafici pri izradi igara koristi skaliranje pozitivnim realnim brojevima [1]. Na primjer, ukoliko množimo y koordinatu točaka $P_i = (x_i, y_i, z_i)$ koeficijentom 3, dobit ćemo objekt 3 puta viši od početnog objekta. Točke koje određuju objekt će sada biti oblika $P_i = (x_i, 3y_i, z_i)$. Ukoliko množimo x i z koordinatu točaka $P_i = (x_i, y_i, z_i)$ koeficijentom $\frac{1}{2}$ dobit ćemo objekt koji je upola tanji od početnog objekta. Točke koje određuju objekt će sada biti oblika $P_i = \left(\frac{1}{2}x_i, y_i, \frac{1}{2}z_i\right)$.

Općenito, pretpostavimo da želimo skalirati objekt duž osi x, y i z koeficijentima α, β i γ, respektivno. Neka su P_i s koordinatama (x_i, y_i, z_i) točke koje određuju originalni, početni objekt. Skaliranjem originalnog objekta, točke koje određuju novonastali objekt su točke P'_i s koordinatama $(\alpha x_i, \beta y_i, \gamma z_i)$. Skaliranje se može postići množenjem matrica. Neka je S matrica tipa 3×3 na čijoj se dijagonali nalaze koeficijenti skaliranja α, β i γ, odnosno:

$$S = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix}.$$

Točka P_i koja određuje originalni objekt može se prikazati jednostupčastom matricom:

$$P_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}.$$

Transformirana točka P'_i koja određuje novonastali objekt također se može prikazati jednostupčastom matricom:

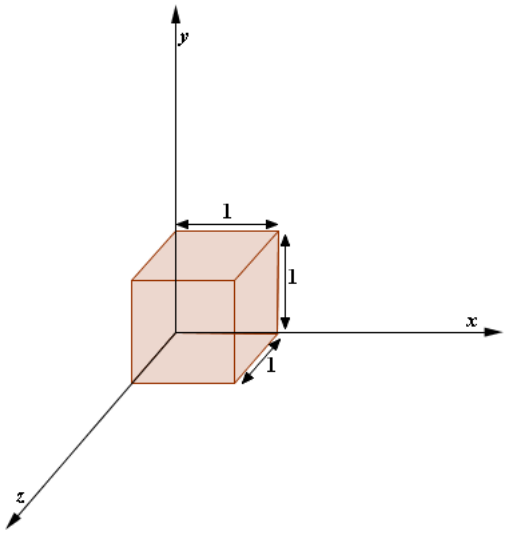
$$P'_i = \begin{bmatrix} x'_i \\ y'_i \\ z'_i \end{bmatrix}.$$

Kako bismo dobili matricu koja prikazuje transformiranu točku P'_i određenu koordinatama $(\alpha x_i, \beta y_i, \gamma z_i)$, očito je potrebno pomnožiti matricu P_i matricom S:

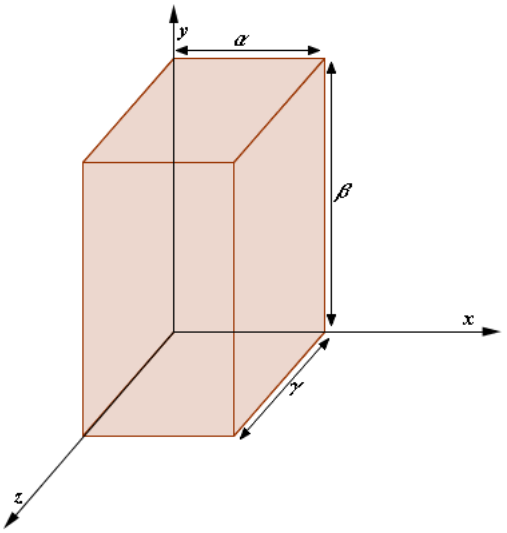
$$P'_i = \begin{bmatrix} x'_i \\ y'_i \\ z'_i \end{bmatrix} = \begin{bmatrix} \alpha x_i \\ \beta y_i \\ \gamma z_i \end{bmatrix} = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}.$$

Primjenjujući koordinatnu matricu A koja u svojim stupcima ima prikazane koordinate svih n točaka originalnog objekta, svih se n točaka može skalirati istodobno. Time dobivamo koordinatnu matricu A' koja odgovara novonastalom objektu:

$$SA = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ z_1 & z_2 & \dots & z_n \end{bmatrix} = \begin{bmatrix} \alpha x_1 & \alpha x_2 & \dots & \alpha x_n \\ \beta y_1 & \beta y_2 & \dots & \beta y_n \\ \gamma z_1 & \gamma z_2 & \dots & \gamma z_n \end{bmatrix} = A'.$$



Slika 1.1: Neskalirani objekt



Slika 1.2: Skalirani objekt

Na primjer, na slici 1.1 se nalazi originalni objekt jedinična kocka. Skaliranjem koeficijentima α, β i γ respektivno dobivamo novi objekt na slici 1.2, odnosno dobivamo kvadar

dimenzija $\alpha \times \beta \times \gamma$.

Skaliranje duž $z-$osi nije vidljivo na zaslonu računala jer se na zaslonu prikazuje projekcija objekta na xy ravninu, no primjenom drugih transformacija može postati vidljivo te se zbog toga također provodi [1].

1.4 Translacija

Sljedeća transformacija koju možemo promatrati je translacija. To je transformacija koja svakoj komponenti točke dodaje određeni pomak. Matematički, neka je u ravnini zadan vektor $\vec{a}$. Taj vektor određuje preslikavanje ravnine koje svakoj točki A te ravnine pridružuje točku A' takvu da je $\overrightarrow{AA'} = \vec{a}$. Vektor $\vec{a}$ zovemo vektorom translacije. Slika dužine $\overline{AB}$ pri translaciji za vektor $\vec{a}$ jest njoj sukladna i paralelna dužina $\overline{A'B'}$. Translacija, nadalje, paralelne pravce preslikava u paralelne pravce, a kut se preslikava u njemu sukladan kut s paralelnim kracima. Mnogokuti se translacijom preslikavaju u sukladne mnogokute. Udaljenost između svih točaka objekta je sačuvana pa takvu transformaciju zovemo rigidnom, odnosno neelastičnom transformacijom [5].

Intuitivno, u računalnoj grafici translacija je transformacija koja prenosi objekt na novu poziciju na ekranu ili ga (djelomično) istiskuje s ekrana. Dakako, transformacija prenosi originalni objekt na novu poziciju ukoliko se primjenjuje na sve točke koje određuju originalni objekt. Time, osim što je objekt na novoj poziciji, ništa drugo se s objektom ne događa, odnosno veličina i oblik originalnog objekta nisu promijenjeni što vidimo na slici 1.4.

Neka su P_i s koordinatama (x_i, y_i, z_i) točke koje određuju originalni, početni objekt. Želimo promijeniti pogled tako da se svaka točka P_i objekta pomakne u novu točku P'_i s koordinatama $(x'_i, y'_i, z'_i) = (x_i + x_0, y_i + y_0, z_i + z_0)$. Translaciju točke P_i u točku P'_i vidimo i na slici 1.3. Vektor

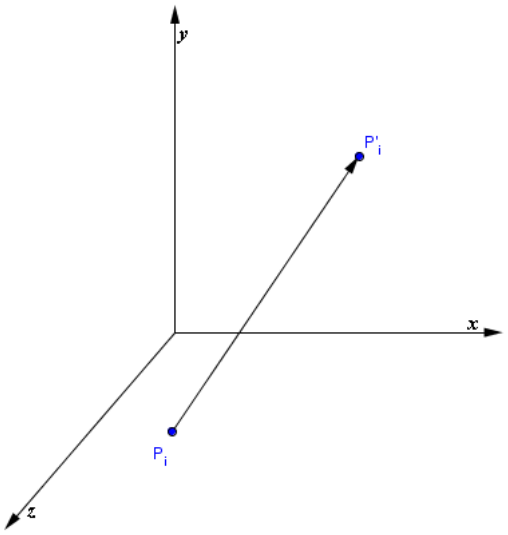
$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

je vektor translacije. Uočavamo da translaciju točke možemo ostvariti zbrajanjem dviju jednostupčastih matrica:

$$\begin{bmatrix} x'_i \\ y'_i \\ z'_i \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}.$$

Neka je

$$A = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \\ z_1 & z_2 & \dots & z_n \end{bmatrix}$$



Slika 1.3: Translacija točke $P_i = (x_i, y_i, z_i)$ za vektor (x_0, y_0, z_0)

koordinatna matrica koja određuje objekt koji translatiramo. Ukoliko definiramo $3 \times n$ matricu T kao matricu:

$$T = \begin{bmatrix} x_0 & x_0 & \dots & x_0 \\ y_0 & y_0 & \dots & y_0 \\ z_0 & z_0 & \dots & z_0 \end{bmatrix},$$

gdje je n broj točaka koje određuju originalni objekt, dobije se koordinatna matrica A' zbrajanjem matrica A i T:

$$A' = A + T.$$

Koordinatna matrica A' sastoji se od novih koordinata n točaka koje određuju objekt koji smo translatirali.

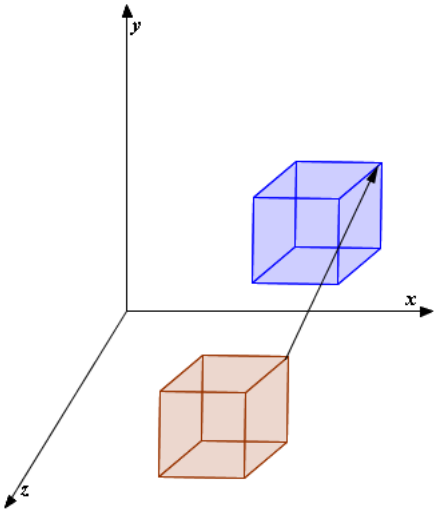
Uočimo da translaciju, osim zbrajanjem matrica možemo ostvariti njihovim množenjem. Neka je P s koordinatama (x, y, z) točka koju translatiramo, a P' s koordinatama (x', y', z') točka dobivena translacijom točke P. Definiramo matricu translacije M:

$$M = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

gdje je vektor (x_0, y_0, z_0) vektor translacije. Tada je

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Ponovno, translacija duž $z-$osi nije vidljiva na zaslonu računala jer se na zaslonu prikazuje projekcija objekta na xy ravninu, no primjenom drugih transformacija može postati vidljiva te se zbog toga također provodi [1].



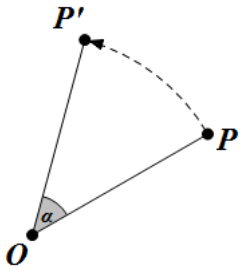
Slika 1.4: Translacija crvenog objekta u plavi

1.5 Rotacija

Glavni dio računalnih igara su objekti koji se rotiraju. Promotrimo najprije rotaciju u ravnini. Neka je O zadana točka ravnine, a α zadani kut. Rotacija oko točke O za kut α je preslikavanje koje točki T ravnine, različitoj od točke O, pridružuje točku T' tako da vrijedi

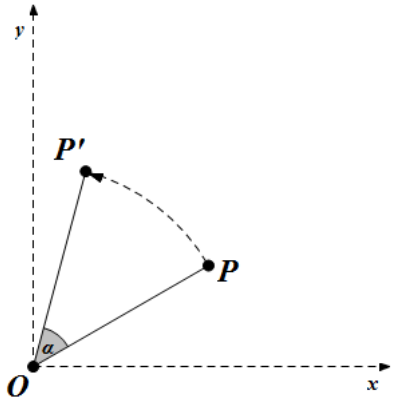
$$|OT| = |OT'| \text{ i } \angle TOT' = \alpha.$$

Kažemo da je točka O središte (centar) rotacije, a kut α kut rotacije. Zadanu točku P rotiramo duž ravninskog luka na stalnoj udaljenosti od čvrste točke, središta rotacije O, za kut α kao što je prikazano na slici 1.5.



Slika 1.5: Rotacija točke u ravnini

Uobičajeno, kao središte rotacije definiramo ishodište koordinatnog sustava (1.6), no ono može biti bilo koja točka ravnine.



Slika 1.6: Središte rotacije u ishodištu xy ravnine

Primijetimo da se kao ni prilikom translacije veličina i oblik rotiranih objekata ne mijenjaju.

Možemo zamisliti da smo rotirali točku T s koordinatama $(r, 0)$ u polarnom koordinatnom sustavu za kut θ te dobili točku P. Podsjetimo se da u polarnom koordinatnom sustavu točka P ravnine ima koordinate (r, θ) gdje je r udaljenost točke P od ishodišta koordinatnog sustava O, a θ kut između polupravca PO i $x-$osi. Neka su koordinate točke P u Kartezijevom koordinatnom sustavu (x, y), odnosno u polarnom $(r\cos\theta, r\sin\theta)$. Želimo rotirati točku P za kut α da bismo dobili točku P'. U polarnom koordinatnom sustavu točka P' ima koordinate $(r, \theta + \alpha)$. Prevođenjem polarnih koordinata točke P' u Kartezijeve dobivamo $P' = (r\cos(\theta + \alpha), r\sin(\theta + \alpha))$. Koristeći trigonometrijske identitete za kosinus i sinus zbroja dobivamo $P' = (r\cos\theta\cos\alpha - r\sin\theta\sin\alpha, r\cos\theta\sin\alpha + r\sin\theta\cos\alpha)$. Znamo da je $P = (r\cos\theta, r\sin\theta) = (x, y)$ iz čega je $P' = (x\cos\alpha - y\sin\alpha, x\sin\alpha + y\cos\alpha)$.

Prilikom rotacije u prostoru postoji pravac na koji rotacija neće imati utjecaj. Taj pravac nazivamo os rotacije. Rotacija objekata u prostoru moguća je oko bilo koje od osi. Promotrimo najjednostavniju rotaciju, onu oko osi okomite na zaslon, odnosno $z-$osi i sa središtem rotacije u ishodištu odabranog koordinatnog sustava. Neka su P_i s koordinatama (x_i, y_i, z_i) točke koje određuju originalni, početni objekt. Na te točke primjenjujemo rotaciju za kut α. Želimo promijeniti pogled tako da se svaka točka P_i objekta pomakne u novu točku P_i' s koordinatama (x_i', y_i', z_i'). Kako rotiramo oko $z-$osi, koordinate z_i neće biti promijenjene. Očito je ova rotacija identična rotaciji u ravnini, odnosno:

$$\begin{aligned} x_i' &= x_i \cos\alpha - y_i \sin\alpha \\ y_i' &= x_i \sin\alpha + y_i \cos\alpha. \\ z_i' &= z_i \end{aligned}$$

Ove se jednadžbe matrično mogu zapisati kao:

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}.$$

Matricu $\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$ zovemo matrica rotacije oko $z-$osi za kut α.

Rotacija oko $x-$osi izgleda kao i rotacija oko $z-$osi ako zamijenimo $x-$os sa $y-$osi, $y-$os sa $z-$osi i $z-$os sa $x-$osi. Neka je $P_i'(x_i', y_i', z_i')$ točka koja nastaje rotacijom točke $P_i(x_i, y_i, z_i)$ oko $x-$osi za kut α. Koordinate točke P_i' možemo dobiti matričnim množenjem:

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}.$$

Matricu $\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$ zovemo matrica rotacije oko $x-$osi za kut α.

Rotacija oko $y-$osi izgleda kao i rotacija oko $z-$osi ako zamijenimo $x-$os sa $z-$osi, $y-$os sa $x-$osi i $z-$os sa $y-$osi. Neka je $P_i'(x_i', y_i', z_i')$ točka koja nastaje rotacijom točke $P_i(x_i, y_i, z_i)$ oko $x-$osi za kut α. Koordinate točke P_i' možemo dobiti matričnim množenjem:

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \end{bmatrix} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}.$$

Matricu $\begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$ zovemo matrica rotacije oko $y-$osi za kut α.

Rotacije oko koordinatnih osi mogu biti kombinirane kako bi se dobio određen prikaz. Na primjer rotiramo objekt najprije oko $x-$osi za kut $30°$, nakon toga oko $y-$osi za kut $-70°$ te na kraju oko $z-$osi za kut $-27°$. Ove tri sukcesivne rotacije možemo zapisati kao jednu transformaciju $P' = RP$, gdje je R produkt triju matrica rotacija:

$$R = \begin{bmatrix} \cos(-27°) & -\sin(-27°) & 0 \\ \sin(-27°) & \cos(-27°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-70°) & 0 & \sin(-70°) \\ 0 & 1 & 0 \\ -\sin(-70°) & 0 & \cos(-70°) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos 30° & -\sin 30° \\ 0 & \sin 30° & \cos 30° \end{bmatrix}.$$

$$= \begin{bmatrix} 0.305 & -0.025 & -0.952 \\ -0.155 & 0.985 & -0.076 \\ 0.940 & 0.171 & 0.296 \end{bmatrix}$$

1.6 Zrcaljenje

Zrcaljenje u $\mathbb{R}^3$ je transformacija koja simetrično preslikava objekt preko ravnine ili preko točke. Neka su P_i s koordinatama (x_i, y_i, z_i) točke koje određuju originalni, početni objekt. Želimo promijeniti objekt tako da se svaka točka P_i objekta preslika u novu točku P'_i s koordinatama (x'_i, y'_i, z'_i). Zrcaljenje, na primjer, s obzirom na yz ravninu postići ćemo na sljedeći način:

$$x'_i = -x_i, y'_i = y_i, z'_i = z_i.$$

Dobivamo efekt zrcala po čemu je transformacija i dobila ime. Na analogan način možemo preslikavati objekt s obzirom na bilo koju koordinatnu ravninu.

Dok se u realnom svijetu susrećemo samo sa zrcaljenjima preko ravnine (zrcalo), u virtualnom se svijetu možemo susreti sa zrcaljenjima preko točke. Neka su P_i s koordinatama (x_i, y_i, z_i) točke koje određuju originalni, početni objekt. Želimo promijeniti objekt tako da se svaka točka P_i objekta zrcali preko ishodišta u novu točku P'_i s koordinatama (x'_i, y'_i, z'_i). To ćemo postići na sljedeći način:

$$x'_i = -x_i, y'_i = -y_i, z'_i = -z_i.$$

Zrcaljenja su simetrične transformacije, odnosno zrcaljenjem zrcaljenog objekta dobivamo početni objekt. Mogli bismo očekivati da pošto u $\mathbb{R}^3$ možemo zrcaliti preko ravnine i točke, možemo zrcaliti i preko pravca. Promotrimo primjer transformacije:

$$x'_i = -x_i, y'_i = -y_i, z'_i = z_i.$$

To bi bila transformacija preko $z-$osi. No, iako je objekt transformiran preko $z-$osi, ova je transformacija ujedno i rotacija oko $z-$osi za 180 stupnjeva [5].

1.7 Smicanje

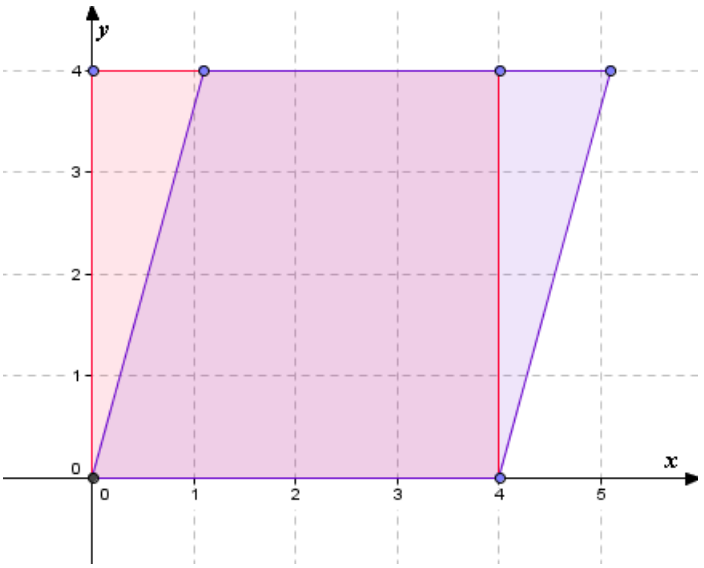
Još jedna od osnovnih transformacija je smicanje (eng. *shear*). Ona u $\mathbb{R}^2$ transformira kvadrat u romb kao što je prikazano na slici 1.7. Djelovanje smicanja sastoji se od transformacije točaka duž koordinatnih osi. Budući da utječe na kutove objekata, ne koristi se često. Smicanje u $\mathbb{R}^2$ je transformacija koja čuva sve točke duž dane osi, dok ostale točke translatira paralelno duž te osi za udaljenost proporcionalnu njihovoj udaljenosti od osi. Na primjer, promotrimo smicanje pravokutnika duž $x-$osi. Koordinata y bilo koje točke (x, y) ostat će nepromijenjena. Koordinata x promijenit će se linearno, ovisno o udaljenost točke od $x-$osi, odnosno o y.

Neka su točke pravokutnika (x_i, y_i). Smicanjem duž x osi za koeficijent a dobivamo točke deformiranog objekta na sljedeći način:

$$x_i' = x_i + ay_i$$
$$y_i' = y_i.$$

Analogno, smicanje duž $y-$osi za koeficijent b dobit ćemo na sljedeći način:

$$x_i' = x_i$$
$$y_i' = y_i + bx_i.$$



Slika 1.7: Djelovanje smicanja duž $x-$osi na kvadrat

Smicanje se istovremeno može sastojati od deformacije duž obje osi u $\mathbb{R}^2$. Tada se koordinate točke (x, y) deformiraju u (x', y') na sljedeći način:

$$x' = x + ay$$
$$y' = y + bx.$$

Smicanje u $\mathbb{R}^3$ duž na primjer x i y osi promijenit će koordinate točke (x, y, z) u (x', y', z') na sljedeći način:

$$x' = x + az$$
$$y' = y + bz$$
$$z' = z$$

gdje su a i b realni brojevi. Analogno dobivamo smicanje duž ostalih koordinatnih osi.

1.8 Primjena transformacija u izradi igre

Primarna uporaba transformacija je manipulacija objektima u igraćem svijetu. Dizajner igre može stvoriti igraći prostor, na primjer neki ured. Stvorit će zidove, prozore, pod i tako dalje kao skup točaka definiran tamo gdje ih mi želimo u igraćem svijetu. Međutim, pretpostavimo da želimo dva identična stola u uredu ili ista stola u različitim prostorima. Dizajner može stvoriti novu verziju stola za svaku lokaciju, ali to je nepotrebno gomilanje memorije potrebne za model. Umjesto toga, možemo imati jednu „glavnu" verziju stola i na njemu provesti niz transformacija kojima ćemo pozicionirati kopiju stola na mjesto na koje želimo.

Kada dizajner igre stvori objekt ili ga mi stvorimo u programu, koordinate točaka koje definiraju objekt definirane su u određenom lokalnom prostoru tog objekta. Središte tog prostora smješta se na poziciju pogodnu za igru, najčešće na dno objekta ili u središte objekta. Smještanje u središte objekta pogodno je ukoliko ćemo često rotirati objekt oko njegovog središta, a na dno ukoliko će objekt biti statičan. Kada konstruiramo igraći svijet, definiramo određen koordinatni sustav, odnosno igraći prostor u kojeg smještamo objekte. Kako bi djelovali na objekte, smještamo njegove lokalne koordinate u igraći svijet i transformiramo ih.

Kada bi u igraćem svijetu koristili lokalne koordinate objekata, objekti bi bili centrirani oko ishodišta igraćeg koordinatnog sustava. Da bismo to izbjegli, djelujemo transformacijama na svaki objekt i smještamo ga na njegovo mjesto te orijentiramo prema igraćem svijetu. Najčešće se koriste translacija, rotacija i skaliranje. Translacija i rotacija odgovaraju dvjema karakteristikama koje želimo kontrolirati: poziciji te orijentaciji. Također, one su rigidne transformacije pa se veličina i oblik objekata ne mijenjaju što je najčešće

željeni efekt. Skaliranje deformira objekt, no često se koristi zbog veličine objekata. Ukoliko dvoje dizajnera izgrađuju igraći svijet i ne dogovore se oko relativne mjere objekata, stol može ispasti veći od sobe ukoliko je smješten u sobu bez transformacije skaliranja. Umjesto da dizajner mijenja model, možemo koristiti skaliranje [5].

Već smo primijetili da primjena transformacija nije komutativna. Promotrimo još jedan primjer. Ukoliko transformiramo točku $(0, 0, 0)$, rotacija oko $z-$osi za neće imati utjecaja pa će rotacija oko $z-$osi za 90 stupnjeva i translacija za vektor (t_x, t_y, t_z) biti ekvivalentna translaciji za vektor (t_x, t_y, t_z). Ukoliko najprije translatiramo točku $(0, 0, 0)$ za vektor (t_x, t_y, t_z) pa je tek onda rotiramo oko $z-$osi za 90 stupnjeva dobit ćemo točku $(-t_y, t_x, t_z)$. Uočavamo da je poredak djelovanja transformacija bitan. Uobičajeno, objekte transformiramo tako da ih prvo skaliramo, nakon toga rotiramo, a na kraju translatiramo.

Osim što objekt iz njegovog lokalnog prostora možemo smjestiti u igraći svijet, objekt možemo smjestiti i u lokalni prostor nekog drugog objekta. Na primjer, pretpostavimo da je dizajner igre napravio posebno ljudsku ruku, a posebno ljudsko tijelo. Tijelo je smješteno u svoj lokalni prostor tako da mu je središte prostora središte tijela. Središte lokalnog prostora ruke neka je rame jer je ono najčešće središte rotacije ruke. Ukoliko smjestimo ruku i tijelo posebno u igraći svijet, ruka će biti unutar tijela, a ne na ramenu. Ideja je najprije transformirati ruku kako bi ona odgovarala tijelu. Dakle, smještamo ruku u u lokalni prostor tijela pa taj lokalni prostor tijela smještamo u igraći svijet. Tijelo i ruku tretiramo kao dva odvojena objekta, svaki sa svojim transformacijama. Provodimo translaciju, rotaciju i skaliranje tijela, a nakon toga translaciju, rotaciju i skaliranje ruke. Naravno, moguća je i veća hijerarhija objekata.

Poglavlje 2

Testiranje presjeka

2.1 Uvod

Bilo da polažemo objekt u igraći svijet ili ga animiramo, bitno nam je kako on u igri djeluje na druge objekte, odnosno bitna nam je interakcija objekata. Pri izradi igre potreban je alat koji će prepoznati kada će dva objekta utjecati jedan na drugog. U ovom poglavlju pitanje je kako otkriti kada se dva geometrijska objekta sijeku. U realnom je svijetu problem interakcije objekata jednostavan. Objekti zbog svojih fizičkih osobina ne prodiru jedan u drugi. U igraćem, virtualnom svijetu ograničenja moramo stvoriti sami. Tek kada osiguramo metode kojima ćemo otkriti preklapaju li se objekti, možemo nastaviti razmišljati kako će objekt reagirati prilikom preklapanja. Očito su objekti u igraćem svijetu različitih oblika pa je traženje presjeka složen problem.

2.2 Presjek objekata

Najdirektniji pristup otkrivanju da li se dva objekta sijeku je direktno očitavanje spojenih točaka objekata. Možemo početi sa spojnicom dviju točaka jednog objekta i vidjeti siječe li se sa spojnicom dviju točaka drugog objekta. Tada bismo nastavili sa sljedećim parom točaka i ponovno provodili test. Iako bi primitivan pristup dao rezultate, osim ako je objekt unutar objekta, potrajao bi i većina vremena u kojem bi se podaci testirali bila bi nepotrebna.

Pretpostavimo da smo okružili svaki objekt jednostavnijim objektom, na primjer sferom. Tada možemo testirati sijeku li se dvije sfere. Ukoliko se one ne sijeku, tada se niti objekti ne sijeku. Ukoliko se sijeku, možemo probati usporediti druge pojednostavljene verzije objekata, na primjer kvadre. Ukoliko se kvadri (kutije) sijeku, tek tada trebamo izvesti test s točkama koje tvore objekt. Objekte kojima okružujemo originalne objekte nazivamo graničnim objektima. U igrama najčešće možemo ignorirati geometriju originalnih

objekata i samo koristiti jednostavne objekte kojima ih okružujemo kako bismo detektirali presjeke. Naime, radnja igre se može odvijati toliko brzo da ne primjećujemo sudaranje objekata, odnosno njihovu preranu interakciju ili je greška mala pa je zanemarujemo.

2.3 Udaljenost objekata

Testovi presjeka često se jednostavnije opisuju u terminima računanja udaljenosti između dva primitivna objekta, kao što su pravac i točka.

Neka su $A = (a_1, a_2, a_3), B = (b_1, b_2, b_3)$ dvije točke u prostoru dane svojim pravokutnim koordinatama. Tada je udaljenost od A do B jednaka

$$d(A, B) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + (b_3 - a_3)^2}.$$

Neka je pravac p dan vektorskom jednadžbom

$$\vec{r} = \vec{r_1} + t\vec{s}$$

gdje je $\vec{r_1}$ radijvektor točke T_1 na pravcu p, a $\vec{s}$ vektor smjera pravca p. Udaljenost točke T_0 s radijvektorom $\vec{r_0}$ od pravca p je

$$d(T_0, p) = \frac{\left|(\vec{r_1} - \vec{r_0}) \times \vec{s}\right|}{\left|\vec{s}\right|}.$$

Udaljenost točke $T_0 = (x_0, y_0, z_0)$ od ravnine $\pi ... Ax + By + Cz + D = 0$ je

$$d(T_0, \pi) = \frac{|Ax_0 + By_0 + Cz_0 + D|}{\sqrt{A^2 + B^2 + C^2}}.$$

2.4 Sfera kao granični objekt

Najjednostavniji granični objekt je sfera. Sfera je skup točaka prostora jednako udaljenih od neke čvrste točke. Prednost sfere nad ostalim graničnim objektima je njezina kompaktna reprezentacija jer ju određuju samo središte i radijus. Jednadžba sfere sa središtem u točki $S = (p, q, r)$ i polumjerom R je $(x-p)^2 + (y-q)^2 + (z-r^2) = R^2$. Također, kada okružujemo originalni objekt, sfera je neovisna o rotaciji objekta što nam omogućuje da se prilikom kretanja objekta moramo brinuti samo o poziciji, odnosno središtu sfere. Ukoliko objekt skaliramo, možemo skalirati radijus analogno.

U idealnom slučaju, želimo najmanju moguću sferu koja opisuje cijeli originalni objekt kojeg u nju smještamo. Ukoliko bi sfera bila premala, mogli bismo zanemariti presjek

dvaju objekata, a ukoliko bi bila prevelika, koristili bismo sporije algoritme za testiranje sijeku li se objekti. Problem koji se nameće je kako odabrati pogodnu sferu.

Jedna od najjednostavnijih metoda je uzeti lokalno ishodište objekta kao središte sfere i izračunati radijus sfere kao najveću udaljenost ishodišta od svih vrhova objekta. Problem s ovim načinom je da smo kao ishodište objekta mogli uzeti bilo koju točku objekta, na primjer dno objekta kako bi ga lakše smjestili u igraći svijet. Tada je, očito, sfera prevelika. Postoje neke efikasnije metode. Primjerice, mogli bismo računati prosjek koordinata svih vrhova objekta da bismo dobili neki centar koji možemo iskoristiti za središte sfere. No, i u tom slučaju dobili bismo preveliku sferu. Također, mogli bismo objekt okružiti kvadrom koja ima bridove paralelne s koordinatnim osima te kvadar smjestiti u sferu. Tako smo pogodno smjestili središte sfere, no njezin je radijus prevelik. Kao kompromis, metoda koja se najčešće koristi je za središte sfere uzeti središte opisanog kvadra, a radijus sfere izračunati kao najveću udaljenost središta sfere od točaka originalnog objekta [5]. Niti ova metoda u svakom slučaju ne daje najbolje rezultate, no iako ne želimo bespotrebno velike sfere, negeneriranje najmanje moguće sfere nije veliki problem.

Presjek dviju sfera

Utvrđivanje da li se dvije sfere sijeku je jednostavno kao i njezina reprezentacija dvama podacima. Potrebno je samo utvrditi da li je udaljenost između središta dviju sfera manja od zbroja njihovih radijusa. Neka su koordinate središta prve sfere $S_1 = (x_1, y_1, z_1)$ i koordinate središta druge sfere $S_2 = (x_2, y_2, z_2)$. Udaljenost tih točaka dana je s

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2},$$

što se izvodi primjenom Pitagorina poučka. Ukoliko je radijus prve sfere r_1, a radijus druge sfere r_2, sfere se sijeku ako i samo ako je

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \leq r_1 + r_2.$$

Pošto je operacija drugog korijena u računalu spora, koristi se relacija

$$(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2 \leq (r_1 + r_2)^2.$$

Presjek sfere i polupravca

Još jedno mjesto gdje se koristi određivanje da li se dva geometrijska objekta sijeku je kada generiramo zraku (polupravac) i određujemo koje objekte u igri će ona sjeći. Očito zraku ne možemo smjestiti u sferu.

Određivanje postoji li presjek pravca i sfere jednostavan je poput određivanja postoji li presjek dviju sfera. Potrebno je odrediti udaljenost između središta sfere i pravca. Ako

je ta udaljenost manja ili jednaka radijusu sfere, tada pravac siječe sferu. Potrebna je dodatna provjera budući da koristimo polupravac. Ukoliko je sfera iza ishodišta polupravca i ishodište je izvan sfere, nema presjeka. Neka je P ishodište, odnosno početna točka polupravca p, S središte sfere i $\vec{t}$ vektor s početnom točkom P i krajnjom točkom S. Ako je kut između vektora smjera polupravca p i vektora $\vec{t}$ veći od 90° i P je izvan sfere, onda je sfera iza ishodišta polupravca, to jest nema presjeka sfere i polupravca [5]. Kut provjeravamo računajući skalarni produkt vektora smjera polupravca p i vektora $\vec{t}$. Taj je skalarni produkt veći od nule za kutove manje od 90 stupnjeva. Ako je sfera iza ishodišta polupravca, može doći do presjeka. Međutim, tada se točka P sigurno nalazi u presjeku pa je dovoljno uz provjeru kuta provjeriti udaljenost točke P od točke S. Ukoliko je udaljenost veća od radijusa sfere, nema presjeka, a ukoliko je manja ili jednaka, postoji presjek.

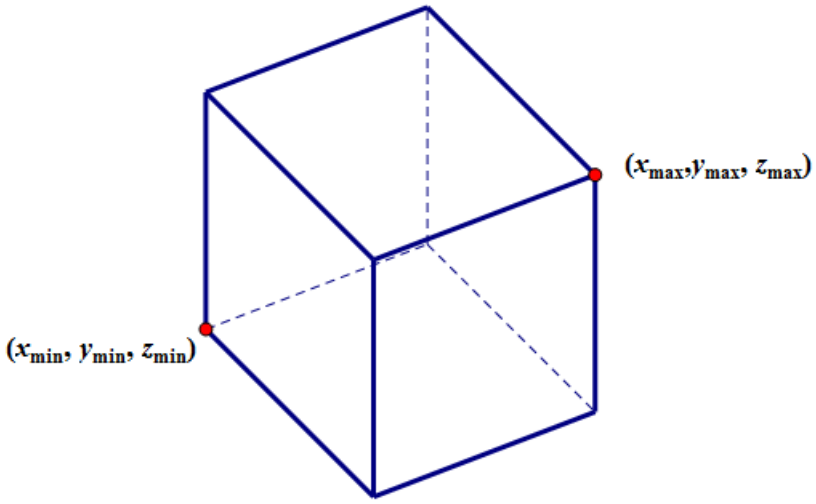
Presjek sfere i ravnine

Ravninu, kao niti polupravac, ne možemo smjestiti u sferu. Potreba za određivanjem postoji li presjek ravnine i objekta je kod kretanja objekta uz, na primjer, zid koji shvaćamo kao ravninu. Provjera da li sfera i ravnina imaju točke presjeka je također jednostavna. Možemo odrediti udaljenost točke i ravnine. Ukoliko je udaljenost središta sfere i ravnine veća od njezinog radijusa, tada sfera i ravnina nemaju zajedničkih točaka. Ukoliko je udaljenost središta sfere i ravnine manja ili jednaka radijusu sfere, tada sfera i ravnina imaju zajedničkih točaka.

2.5 Kvadar s bridovima paralelnima s koordinatnim osima

Sfera je dobar i „jeftin" granični objekt, ali za vrlo mali skup objekata. Na primjer, koristili bismo je prilikom izrade igre s biljarskim kuglama. Za više uglate oblike potreban nam je profinjeniji granični objekt. Jedan takav je kvadar kao granični objekt. Kao i kod sfere, najviše bismo voljeli najmanji kvadar u koji igraći objekt stane. Najjednostavniji tip kvadra za reprezentaciju u koordinatnom sustavu je onaj kojemu su bridovi paralelni s koordinatnim osima igraćeg svijeta. Zovemo ga AABB objektom (eng. *axis-aligned bounding box*). Prilikom reprezentacije tog objekta dovoljne su nam dvije točke, minimum i maksimum xyz pozicija odnosno točka $T_{min} = (x_{min}, y_{min}, z_{min})$ i točka $T_{max} = (x_{max}, y_{max}, z_{max})$ kao na slici 2.1.

Kada translatiramo objekt, translatiramo dvije točke T_{max} i T_{min}. Također, ako objekt skaliramo, skaliramo i dvije točke ovisno o središtu kvadra. Pošto su bridovi kvadra paralelni s osima, svakom rotacijom objekta moramo ponovno računati minimalnu i maksimalnu točku što kod sfere nismo morali jer rotacija ne djeluje na nju. Kao i u slučaju sfere,



Slika 2.1: AABB objekt

malo je objekata kojima je kvadar s bridovima paralelnima s koordinatnim osima dobar granični objekt. Zaobljenim igraćim objektima, poput ljudi, uglovi su hendikep. Međutim, testovi presjeka su jednostavni i brzi pa se često koriste.

Nameće se pitanje, kako odrediti T_{max} i T_{min}. Objekt kojeg ćemo smjestiti u kvadar najprije smještamo u igraći svijet. Nakon toga, T_{max} i T_{min} postavljamo na prvu točku koja određuje objekt. Počevši od druge točke, uspoređujemo xyz vrijednosti svake točke s T_{max} i T_{min}. Ukoliko je jedna od koordinata manja od koordinate trenutnog minimuma, postavljamo vrijednost koordinate trenutnog minimuma na koordinatu točke objekta. Analogni postupak provodimo i za traženje koordinata točke T_{max} [5].

Presjek dvaju AABB objekata

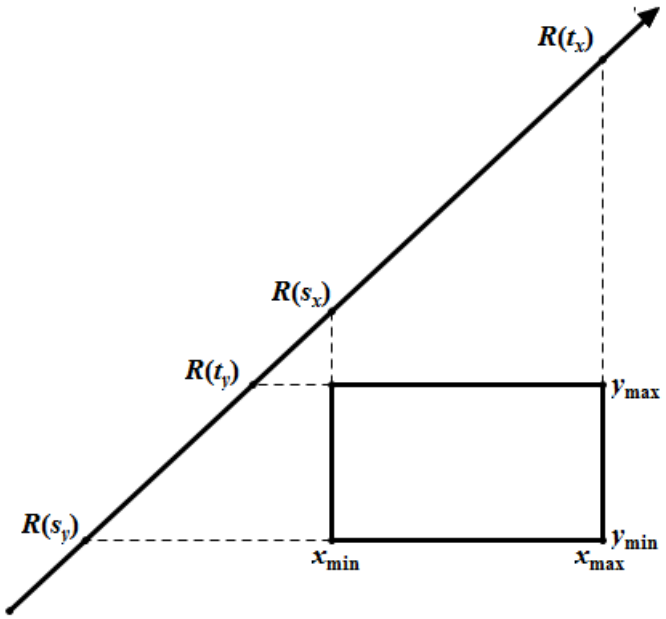
Kako bismo našli presjeke, odnosno odredili postoji li presjek između dva AABB objekta, provjeravamo kvadar u svakom od koordinatnih smjerova. Ukoliko možemo naći ravninu koja odvaja dva kvadra u bilo kojem koordinatnom smjeru, tada se dva kvadra ne sijeku. U suprotnom, oni se sijeku. Dakle, dva se AABB objekta sijeku ako i samo ako se njihove projekcije na x, y i z os sijeku.

Pronađimo, na primjer, ravninu koja odvaja dva AABB objekta u smjeru $x-$osi. Budući da su bridovi kvadra paralelni s koordinatnim osima, svodimo problem na jednodimenzionalni problem, odnosno na problem na brojevnom pravcu. Minimalne i maksimalne vrijednosti dvaju kvadara su ekstremne točke intervala na brojevnom pravcu. Ukoliko su intervali odvojeni, odnosno njihov presjek je prazan skup, kvadri su odvojeni duž $x-$osi.

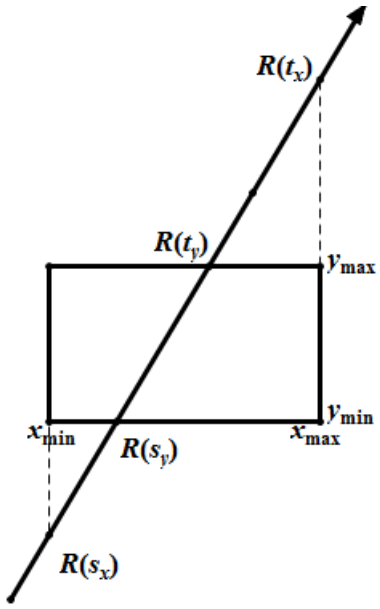
Tome je tako ukoliko je maksimalna x koordinata jednog intervala manja od minimalne x koordinate drugog intervala. Dva se AABB objekta sijeku ukoliko u smjerovima svih koordinatnih osi ne pronađemo ravninu koja ih odvaja.

Presjek AABB objekta i polupravca

Kod sfere smo uočili da nam je potreban alat koji određuje postoji li presjek objekta i polupravca. U ovom je slučaju objekt smješten u kvadar pa nam je potreban alat koji određuje postoji li presjek kvadra i polupravca. Neka je R polupravac za koji određujemo postoji li presjek između njega i kvadra. Neka je točka $R(s_x)$ presjek polupravca i minimalne x ravnine kvadra te neka je $R(t_x)$ presjek polupravca i maksimalne x ravnine kvadra. Također, neka je $R(s_y)$ presjek polupravca i minimalne y ravnine kvadra te neka je $R(t_y)$ presjek polupravca i maksimalne y ravnine kvadra. Ako interval $[s_x, t_x]$ i interval $[s_y, t_y]$ nemaju zajedničkih točaka, polupravac ne siječe kvadar. Ta je situacija prikazana na slici 2.2. Ukoliko imaju zajedničkih točaka kao što je to slučaj na slici 2.3, tada provodimo test duž preostalih ravnina. Ukoliko u oba od preostalih slučajeva postoji presjek, zraka siječe kvadar.



Slika 2.2: Polupravac ne siječe kvadar



Slika 2.3: Polupravac siječe kvadar

Neka je AABB objekt određen dvjema točkama T_{min} i T_{max}. Neka su koordinate tih točaka: $T_{min} = (x_{min}, y_{min}, z_{min})$ i $T_{max} = (x_{max}, y_{max}, z_{max})$. Neka je polupravac zadan početnom točkom $P = (P_x, P_y, P_z)$ i vektorom smjera $v = (v_x, v_y, v_z)$. Tražimo parametre za koje polupravac s početnom točkom P i vektorom smjera v siječe minimalnu i maksimalnu ravninu. Na početku, na primjer u smjeru x, odnosno na ravnini yz, računamo presjek polupravca s ravninom $x = x_{min}$ i ravninom $x = x_{max}$. Kako bismo to odredili, rješavamo jednadžbe:

$$P_x + s_x v_x = x_{min}$$
$$P_x + t_x v_x = x_{max}$$

po s_x i t_x pa dobivamo:

$$s_x = \frac{x_{min} - P_x}{v_x}$$
$$t_x = \frac{x_{max} - P_x}{v_x}.$$

S obzirom da promatramo interval $[s_x, t_x]$, želimo osigurati da je $s_x < t_x$. Ako je $\frac{1}{v_x} < 0$, onda je $s_x > t_x$ pa ćemo zamijeniti x_{min} i x_{max}. Pošto promatramo polupravac, parametri s_x i t_x su iz intervala $[0, \infty\rangle$. Na početku inicijaliziramo maksimalan interval pa postavimo

$s_{max} = 0$, $t_{min} = \infty$. Izračunali smo s_x i t_x pa ukoliko je $s_x > s_{max}$ postavljamo s_{max} na s_x. Također ukoliko je $t_x < t_{min}$ postavljamo t_{min} na t_x. Znamo da polupravac neće sjeći kvadar ako je $s_{max} > t_{min}$. Sada analogno računamo s_y i t_y. Ukoliko je $s_y > s_{max}$, postavljamo s_{max} na s_y te ukoliko je $t_y < t_{min}$, postavljamo t_{min} na t_y. Uviđamo da se polupravac i kvadar ne sijeku ako je $s_{max} > t_{min}$. Još je preostalo na analogan način provjeriti xy i xz ravnine [5].

Primijetimo da v_x može biti 0. U tom je slučaju polupravac paralelan s minimalnom i maksimalnom ravninom. Potrebno je provjeriti pripada li P_x segmentu između x_{min} i x_{max}. Ukoliko ne pripada, ne postoji presjek polupravca i kvadra.

Presjek AABB objekta i ravnine

Kao niti polupravac, niti ravninu ne možemo smjestiti u AABB objekt. Već smo kod sfere uočili potrebu za pronalaženjem informacije postoji li presjek između ravnine i objekta. Najprimitivniji način za provjeru postoji li presjek kvadra i ravnine je da provjerimo da li ijedan od bridova kvadra siječe ravninu. To ćemo postići tako da računamo da li se dva vrha kvadra koji određuju promatrani brid nalaze s iste strane ravnine. Pošto kvadar ima 12 bridova, to zahtjeva 24 provjere. Jedno od poboljšanja je da provjerimo samo nasuprotne vrhove kvadra, odnosno 4 prostorne dijagonale.

2.6 Još neki ograničavajući objekti

Kapsula

Sfera kao granični objekt i AABB objekt ovise o igraćem svijetu. Sfera je simetrična duž svih osi, a AABB objekt ima bridove uvijek paralelne s koordinatnim osima. Za većinu objekata ova dva granična objekta ne pružaju dobru aproksimaciju koliko god mi dobro ograničili objekte sferom ili AABB objektom. Primjer objekta koji ne ovisi o igraćem svijetu, odnosno osima igraćeg svijeta je kapsula. Kapsula je valjak koji na oba kraja, odnosno na obje osnovice ima hemisferu, polovinu sfere. Kapsulu, za razliku od sfere i AABB objekta, prikazujemo u svijetu objekta kojeg njome okružujemo. Prvi korak u računu pogodne kapsule je okružiti igraći objekt AABB objektom. Sljedeći je korak pronaći najdulji brid AABB objekta. Duljina dužine koja će biti visina valjka je duljina tog najduljeg brida. Visina valjka prolazi središtima strana AABB objekta, onih koje su okomite na najdulji brid. Još je potrebno izračunati polumjer valjka, odnosno pripadne hemisfere. Za polumjer možemo uzeti duljinu sljedećeg najduljeg brida AABB objekta. Primijetimo da nam je za prikaz kapsule potreban samo polumjer polusfere i dvije točke koje određuju visinu valjka.

Računanje postoji li presjek dvije kapsule je slično računanju postoji li presjek dvije sfere. Računamo udaljenost između dviju dužina, odnosno visina dviju kapsula, a nakon

toga tu udaljenost uspoređujemo s polumjerima. Ukoliko je udaljenost manja od zbroja dvaju polumjera, kapsule se sijeku.

Kvadri orijentirani prema objektu

Kvadre s bridovima paralelnima s koordinatnim osima, odnosno AABB objekte je jednostavno stvoriti i lako se očitava postoje li presjeci, no igraći objekt koji nije poravnat s koordinatnim osima ne okružuju dobro. Točniji pristup bio bi kreirati kvadar koji dobro okružuje objekt u njegovom lokalnom koordinatnom sustavu, a nakon toga rotirati i translatirati taj kvadar zajedno s objektom u igraćem koordinatnom sustavu. Takvi kvadri, nazivaju se kvadri orijentirani prema objektu, odnosno OBB (eng. *object-oriented-boxes*). Prednost OBB objekata je da ne moramo računati njegove vrhove svaki puta kada se objekt u igraćem svijetu pomakne, već trebamo transformirati postojeći objekt. Mana OBB objekata je komplicirano testiranje postoje li presjeci između dvaju OBB objekta. To je zbog činjenice da orijentacije objekata jedan prema drugom mogu biti proizvoljne. Više o drugim ograničavajućim objektima može se pročitati u [5].

2.7 Hijerarhija ograničavanja objekata

U slučaju da se igraći objekti dosta razlikuju od pripadnih graničnih objekata, veća je vjerojatnost da će testovi presjeka prikazati presjek iako do njega nije došlo. Jedno od rješenja problema je koristiti skup graničnih objekata kako bismo dobili bolju aproksimaciju površine objekta. Početni objekt možemo rastaviti na više objekata te ga ograničiti s nekoliko graničnih objekata. Kako bismo provjerili postoje li presjeci, provodimo test za svaki od graničnih objekata. Ubrzati proces bismo mogli tako da zadržimo i originalni granični objekt i koristimo ga kao grubi test postoji li presjek. Ukoliko ne postoji, provjeravamo dalje. Također, ukoliko koristimo naprimjer kapsule kao granične objekte, možemo kapsule okružiti sferom i provesti brži test presjeka, a tek nakon toga provoditi test s kapsulama. To nam daje hijerarhiju ograničavanja objekata. Najprije uspoređujemo sfere kao granične objekte, ukoliko se one sijeku, objekt rastavimo na nekoliko dijelova i provjerimo sijeku li se sfere koje ograničavaju objekt s drugim objektom. Tek nakon toga provodimo test s kapsulama.

2.8 Dinamični objekti

Do sada su svi testovi koje smo opisali pretpostavljali da se objekti ne pomiču što očito nije tako. Na primjer, u jednom trenutku imamo dva objekta koji se kreću jedan prema drugome. U sljedećem trenutku željeli bismo da se objekti sudare ili minimalno presi-

jecaju. Međutim, ukoliko je razmak između ta dva trenutka prevelik, moguće je da se objekti mimoiđu. Ukoliko koristimo statične testove za svaki trenutak, propustit ćemo sudar objekata. Jedna od metoda koja se može koristiti kako bismo izbjegli problem je da pretpostavimo dva puta kojima će se objekti kretati te tim putem generiramo dva granična objekta. Tada tražimo postoji li presjek između generiranih graničnih objekata. Jednostavan primjer je ukoliko su granični objekti sfere, njihovim pomakom duž dužine nastaje kapsula. Ukoliko se te kapsule sijeku znamo da ćemo između promatranih trenutaka negdje imati presjek objekata.

Oko ovog se problema moramo brinuti samo kada su brzine objekata velike u odnosu na promatrane trenutke prikazane na ekranu.

2.9 Optimizacija

Očito je broj računanja činjenice postoji li presjek objekata u igrama velik. Za svaki objekt potrebno je istražiti siječe li se s drugim objektom. Generiranjem dvije *for* petlje nastaje kvadratna složenost. Pitanje je kako smanjiti broj testiranja presjeka objekata. Očito je ukoliko se ne sijeku objekt 1 i objekt 5, da tada ne trebamo provjeravati sijeku li se objekt 5 i objekt 1, u suprotnom poretku. Također, postoje brojni objekti za koje provjeravamo sijeku li se, a oni se uopće ne miču.

Većina pristupa koristi neku vrstu podjele prostora. Najjednostavnija podjela je duž $x-$osi nizom jednako udaljenih ravnina. Time dobivamo skup ploča koje određuju ravnine duž $x-$osi te neke namještene granice u y i z smjeru. Za svaku ploču spremamo skup objekata koji je presijecaju. Kako bismo testirali postoji li presjek između objekata, određujemo koje ploče objekt siječe te testiramo presjeke s objektima iz tih ploča. Mana ovog pristupa je da se u par ploča mogu nalaziti svi objekti, dok ostale ploče mogu biti prazne ili gotovo prazne što nas dovodi do početnog problema.

Drugi pristup je algoritam brojenja parova i pojednostavljivanja (eng. *sweep and prune*). Pristup je sličan prethodnom. No, umjesto pravilne mreže ploča, koriste se ekstremi objekata kako bi se dobila mreža. Postupak je sličan kao i kod provjere kolizije dvaju AABB objekata. Za svaki objekt projiciramo minimum i maksimum omeđujućih volumena na sve osi. Dva se objekta čiji se intervali ne presijecaju ne sijeku. Svaki minimum i maksimum povezujemo s originalnim objektom, spremamo u listu te uzlazno sortiramo minimume i maksimume. Prolazimo kroz listu od početka do kraja. Kada u listi naiđemo na min_i, dodajemo objekt i u listu aktivnih objekata. Kada naiđemo na max_i, uklanjamo objekt i iz liste aktivnih objekata. Test presjeka za objekt i kada je dodan u listu aktivnih objekata provodi se samo između objekta i i objekata u toj listi.

Poglavlje 3

Grafovi

3.1 Uvod

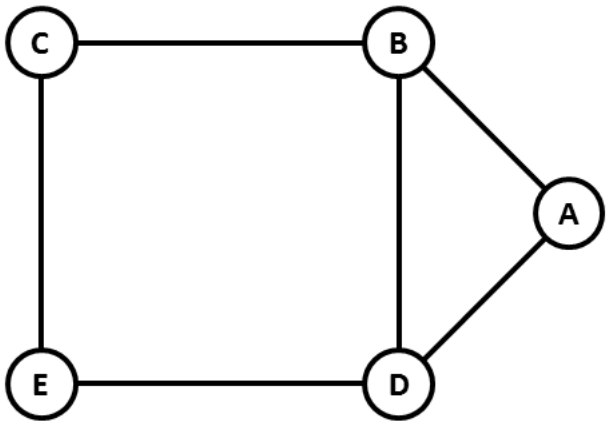
Kada kliknemo na neki pomični objekt u igri i na neku poziciju u igraćem svijetu, često nam se objekt šeće od trenutne pozicije do pozicije na koju smo kliknuli. Pitamo se kako je to ostvareno. Računalo samo ne može razmišljati, nego mu moramo zadati niz preciznih uputa što raditi u svakom koraku šetnje objekta. Dakle, ne možemo objektu reći da nađe najkraći put ili put kojim će najbrže doći, već mu moramo zadati upute kako doći do odredišta. Ovaj problem naziva se problem nalaženja puta. U rješavanju problema potrebni su nam grafovi.

3.2 Grafovi

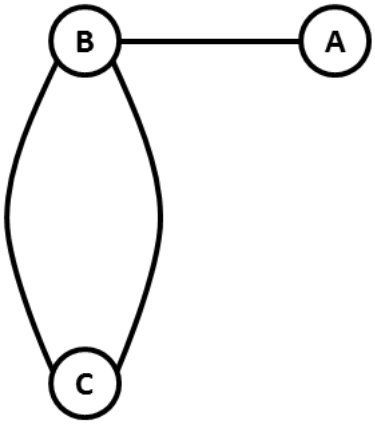
Graf je uređeni par skupova (V, E) gdje je V neprazan skup vrhova, a E skup 2-podskupova od V, koje zovemo bridovi. Svaki brid $e \in E$ spaja dva vrha $u, v \in V$ koji se zovu krajevi od e. Kažemo da su dva vrha susjedna ukoliko postoji brid koji ih spaja. Formalno, vrhovi u i v su susjedni ukoliko postoji $e = \{u, v\} \in E$. Na primjer, na grafu sa slike 3.1 vrhovi A i B su susjedni vrhovi, dok vrhovi A i E nisu susjedni. Brid čiji se krajevi podudaraju zove se petlja, a ako dva ili više bridova povezuje isti par vrhova, zovu se višestruki bridovi. Ako graf sadrži višestruke bridove, zove se multigraf. Primjer multigrafa prikazan je na slici 3.2.

Primjer grafa je mapa gradova. Svaka cesta je brid grafa koji spaja dva grada, odnosno dva vrha. Dva najjednostavnija primjera grafova su potpuni graf, u kojemu je svaki par vrhova brid, i nul graf koji nema bridova.

Definiciju grafa možemo proširiti ukoliko dopustimo usmjerene bridove. To su bridovi koji imaju orijentaciju tako da idu od jednog vrha prema drugome. Usmjerene bridove reprezentiramo uređenim parovima, a ne 2-podskupovima. Graf koji ima usmjerene bridove



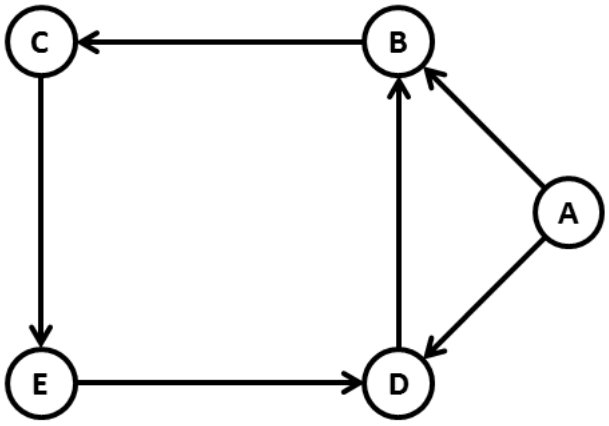
Slika 3.1: Neusmjereni graf



Slika 3.2: Multigraf

nazivamo usmjereni graf ili digraf. Na slici 3.3 prikazan je jedan digraf. Primjer korištenja digrafa je jednosmjerna ulica. Ako graf nije multigraf ili digraf, zovemo ga jednostavan graf.

Dodatno, možemo promatrati sljedeću situaciju. Većina ljudi želi iz grada A stići u grad C, a samo nekolicina u grad B. Obje ceste su jednako široke, što će značiti da je na cesti između grada A i grada C gužva pa je potrebno 30 minuta kako bi se stiglo do grada C. Iz grada A u grad B brže dolazimo jer nema gužve, na primjer za 5 minuta. Uočimo da smo bridovima pridružili brojeve, odnosno težinu. Težinska funkcija na skupu E je funkcija s E u $\mathbb{R}$. Bridno-težinski graf je graf s težinskom funkcijom na skupu bridova. Slika 3.5 prikazuje bridno-težinski graf.



Slika 3.3: Digraf ili usmjereni graf

Niz $(v_0, e_1, v_1, e_2, v_2, ..., e_n, v_n)$ gdje je e_i brid $\{v_{i-1}, v_i\}$ za $i = 1, ..., n$ nazivamo šetnja u (neusmjerenom) grafu. Kažemo da je to šetnja od v_0 do v_n. Duljina šetnje je broj bridova u nizu, odnosno broj vrhova u nizu manje jedan. Kažemo da je šetnja zatvorena ukoliko je $v_n = v_0$. U jednostavnom grafu bridovi šetnje su potpuno određeni vrhovima pa često govorimo o šetnji $(v_0, v_1, ..., v_n)$, gdje se podrazumijeva da su vrhovi v_{i-1} i v_i susjedni. Razlikujemo nekoliko specijalnih vrsta šetnji. Staza je šetnja u kojoj su svi bridovi različiti. Put je šetnja u kojoj su svi vrhovi različiti, osim eventualno prvog i zadnjeg vrha. Zatvoreni put nazivamo ciklus. Za graf kažemo da je povezan ako postoji put između svaka dva različita vrha. Stablo je povezan graf bez ciklusa.

3.3 Implementacija grafova

Kako bismo ideju grafa mogli koristiti prilikom izrade računalne igre ili prilikom bilo koje algoritamske obrade, potrebno ju je jednostavno i praktično prikazati u jeziku razumljivom računalu. Osim da graf zauzima malo memorije, željeli bismo da algoritmi koje ćemo provoditi na grafu budu efikasni. Dvije najčešće metode za implementaciju grafova su matrica susjedstva i lista susjedstva [3].

Matrica susjedstva

Najčešće broj vrhova grafa G označavamo s $v(G)$. Matrica susjedstva za graf G je kvadratna $v(G) \times v(G)$ matrica. Redak i matrice susjedstva sadrži podatke o vrhu v_i grafa G. U $i-$tom retku i $j-$tom stupcu naznačeno je ukoliko postoji brid između vrha v_i i

vrha v_j. Matrica susjedstva dakle zahtjeva jedan bit za svaku od pozicija u matrici. Primijetimo, ovakva reprezentacija pogodna je za jednostavne usmjerene grafove. Ukoliko promatramo bridno-težinski graf, na svaku poziciju matrice umjesto bita smještamo broj, odnosno težinu svakog od postojećih bridova.

Lista susjedstva

Drugi uobičajeni način implementiranja grafa je lista susjedstva. Koristi se polje vezanih lista. Polje sadrži $v(G)$ elemenata, odnosno onoliko elemenata koliko u grafu G postoji vrhova. Na poziciji i u polju je pokazivač na vezanu listu bridova kojem je početni vrh v_i. Vezana lista predstavlja bridove s vrhovima susjednim vrhu v_i. Memorija potrebna za listu susjedstva ovisi o broju bridova i broju vrhova grafa.

Usporedba implementacija

I matrica susjedstva i lista susjedstva mogu biti korištene za usmjerene i neusmjerene grafove. Svaki brid neusmjerenog grafa prikazan je s dva usmjerena brida.

Koja implementacija zauzima manje memorije ovisi o broju bridova grafa. Lista susjedstva sprema informacije samo za one bridove koji se zaista pojavljuju u grafu, dok je kod matrice susjedstva potreban prostor za svaki mogući brid, neovisno postoji li on ili ne. Međutim, za matricu susjedstva nije potrebno polje pokazivača na vezane liste što može biti dodatan trošak memorije. Što je graf gušći, odnosno što više bridova ima, matrica susjedstva je isplativija.

Provjera jesu li dva brida susjedna matricom susjedstva očito je jednostavnija i brža. Prilikom izvršavanja algoritama na grafovima, obično je nespretnije raditi s matricom, odnosno rad s matricom susjedstva je sporiji nego li je to rad s listom susjedstva. U algoritmima nad grafovima, često posjećujemo svakog susjeda svakog vrha. Ukoliko graf implementiramo listom susjedstva, u njoj su spremljeni samo susjedi. Kod matrice susjedstva najprije moramo provjeriti postoji li brid između dva vrha.

3.4 Postavljanje vrhova i bridova grafa

Sada kada smo implementirali ideju grafa, potrebno je pronaći najkraći put između dviju točaka grafa. No, prije toga promotrimo što uopće predstavljaju vrhovi grafa. Svaka ključna točka igraćeg svijeta bit će vrh grafa i svaki put između dviju točaka bit će njegov brid. Možemo zamisliti da je svaka točka igraćeg svijeta vrh grafa, no to bi bilo nepraktično jer bi graf imao stotine tisuća vrhova, a traženje najkraćeg puta trajalo bi jako dugo. Umjesto toga, definiraju se ključne točke igraćeg svijeta. Na primjer, na otvorenom

prostoru bez prepreka nije nam potreban vrh u svakoj točki zbog toga što objekt može prolaziti ravno po prostoru. Uobičajeno, vrhovi grafa potrebni su samo oko prepreka. Nakon što se definiraju vrhovi grafa potrebno je definirati bridove. Naravno, vrhove smijemo spajati bridovima samo ako se bridovi između tih vrhova ne sijeku s nekom od prepreka na putu.

Kada je graf stvoren, računalo treba obaviti sljedeće korake kako bi se objekt mogao kretati. Najprije određuje koji je vrh grafa najbliži objektu kako bi se mogao kretati pravocrtno. Taj vrh je početni vrh. Nakon toga, određuje koji vrh je najbliži cilju. Taj vrh je krajnji vrh. Sada je potrebno odrediti najkraći put koji spaja početni vrh i krajnji vrh grafa.

Naravno, možemo koristiti i bridno-težinski graf. Na primjer, ukoliko je neki put između dva vrha duži ili je na njemu neka savladiva prepreka koja će nas zadržati neko vrijeme, bridovima možemo dodati vrijednosti.

3.5 Najkraći put u netežinskom grafu

Za netežinski graf problem najkraćeg puta je jednostavan. Uobičajeno se koristi *Breadth - first search algoritam* (BFS algoritam). Kod tih se algoritama zadatak rješava razlaganjem na manje podzadatke, podzadaci na još manje podzadatke i tako dalje. Prilikom korištenja BFS algoritama koristi se red. Red je specijalna vrsta liste na kojoj su dozvoljene operacije ubacivanja elemenata na jedan kraj (začelje) liste, uklanjanje elemenata s drugog kraja (čela) liste te pregled sadržaja elemenata na početku liste.

Sve vrhove označavamo bijelom bojom. Početni vrh označavamo crnom bojom, postavljamo njegovu udaljenost na 0 i stavljamo ga u red. Nakon toga dok red nije prazan uklanjamo element u iz reda. Svaki bijeli vrh susjedan vrhu u označavamo sivom bojom, stavljamo ga u red te njegovu udaljenost od početnog vrha postavljamo na udaljenost vrha u od početnog vrha uvećanu za 1 te naznačujemo tko mu je prethodnik. Pri završetku algoritma dobili smo udaljenost svakog vrha od početnog.

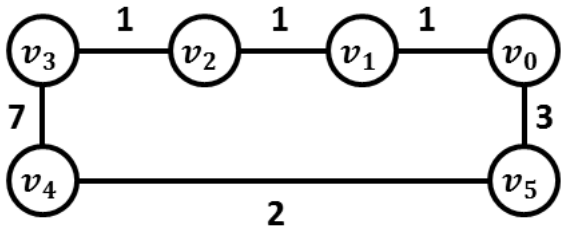
3.6 Najkraći put u težinskom grafu

Na mapi, cesta koja povezuje dva grada označena je udaljenošću ta dva grada ukoliko putujemo tom cestom. Možemo modelirati mrežu cesta pomoću usmjerenog bridno-težinskog grafa. Brojevi kojima modeliramo duljinu puta u nekim drugim primjerima mogu biti težina ili trošak. Tražimo duljinu najkraćeg puta između dva grada. Odnosno, tražimo takav put između vrhova grafa da je zbroj težina bridova na tom putu najmanji moguć. Uočimo da iako dva vrha mogu biti spojena cestom, možda postoji put između drugih vrhova koji je jeftiniji. Jedan je način pronalaska najkraćeg puta naći sve moguće putove

između vrhova, izračunati zbroj težina svakog od puta te izabrati najmanji. Nažalost, ovaj najintuitivniji pristup vodi do sporog rješenja.

Neka su dani vrhovi S i T grafa G. Tražimo najkraći put od vrha S do vrha T. Za dani vrh S naći ćemo najkraći put od S do svakog drugog vrha grafa G. Iako želimo samo najkraći put između vrhova S i T, do sada nije poznat bolji algoritam koji će pronaći najkraći put do jednog vrha od algoritma koji će pronaći najkraći put do svih vrhova.

Ideja za rješavanje problema mogla bi biti da obrađujemo vrhove grafa unaprijed zadanim poretkom. Označimo vrhove grafa s $v_0, ..., v_n$, gdje je $S = v_0$. Kada obrađujemo vrh v_1, određujemo duljinu brida koji spaja v_0 i v_1. Kada obrađujemo vrh v_2, uzimamo kraću od udaljenosti bridova od v_0 do v_2 i zbroja udaljenosti bridova v_0 do v_1 i v_1 do v_2. Općenito, kada obrađujemo vrh v_i, uzimamo u obzir zbroj udaljenosti od v_0 do v_j i udaljenosti od v_j do v_i, gdje je $j < i$, te ga uspoređujemo s težinom mogućeg brida između v_0 i v_i. Međutim, najkraća udaljenost od v_0 do v_i može prolaziti kroz vrh v_j gdje je $j > i$. Takav put neće biti u razmatranju ovakvim pristupom. Na primjer, tako nećemo naći najkraću udaljenost između vrhova v_0 i v_4 na grafu sa slike 3.4.



Slika 3.4: Najkraća udaljenost na grafu

Problem ćemo ukloniti tako da obrađujemo vrhove grafa u poretku udaljenosti od S. Pretpostavimo da smo obradili prvih $i-1$ vrhova najbližih vrhu S. Neka oni određuju skup R. Želimo obraditi i-ti vrh, nazovimo ga X. Najkraća udaljenost od S do X je:

$$d(S, X) = \min_{U \in R}(d(S, U) + w(U, X)).$$

Dakle, najkraća udaljenost između vrha S i vrha X je minimalna vrijednost zbroja udaljenosti između S i U te udaljenosti između vrha U i vrha X, gdje je U neki vrh u R. Drugim riječima, do nekog se vrha najkraćim putem dolazi ili direktno od početnog vrha, ili preko nekog drugog vrha za koji je već određen najkraći put.

Ovaj je algoritam poznat kao Dijkstrin algoritam, a otkrio ga je i objavio Edsger W. Dijkstra, nizozemski matematičar i informatičar [3]. Opišimo kako Dijkstrin algoritam radi. U prvom koraku se najmanja udaljenost do svakog vrha postavlja na beskonačno. Svi se

prethodnici vrhova postavljaju na nedefiniranu vrijednost. Udaljenost početnog čvora postavlja se na nulu. Neka je Q skup neobrađenih vrhova. Kako niti jedan vrh nije obrađen, svi vrhovi nalaze se u Q. Drugi je korak pretražiti cijeli graf i za svaki vrh odrediti udaljenost. Određuje se vrh u s najmanjom udaljenošću, to je na početku očito početni vrh (ostali su na beskonačno). Vrh u briše se iz skupa te se odrađuje treći korak. Treći je korak da se za svaki susjedni vrh vrha u računa udaljenost i to ako se taj vrh nalazi u skupu neobrađenih vrhova. Ukoliko novo akumulirana udaljenost ima manju vrijednost od postojeće udaljenosti, udaljenost se izjednačava s novo akumuliranom udaljenosti, a vrh u postaje prethodni čvor. Sada ponovno tražimo vrh s najmanjom udaljenosti te računamo akumuliranu udaljenost za sve susjedne čvorove koji se nalaze u Q. Postupak se nastavlja dok Q ne postane prazan skup.

Promotrimo graf sa slike 3.5. Želimo pronaći najkraći put od vrha S do vrha T. U prioritetnom redu Q na početku se nalaze svi vrhovi, odnosno $Q = \{S, A, B, C, D, T\}$. Udaljenost do S postavlja se na 0, a ostale udaljenosti na beskonačno. Brišemo vrh s najmanjom udaljenošću iz reda Q, a to je očito S. U redu Q sada su vrhovi: $Q = \{A, B, C, D, T\}$. Za svaki susjedni vrh od S u redu Q računamo udaljenosti i vrh koji vodi do njega. Dobivamo $(A, 18, \{S\})$ te (C, 15, $\{S\}$).

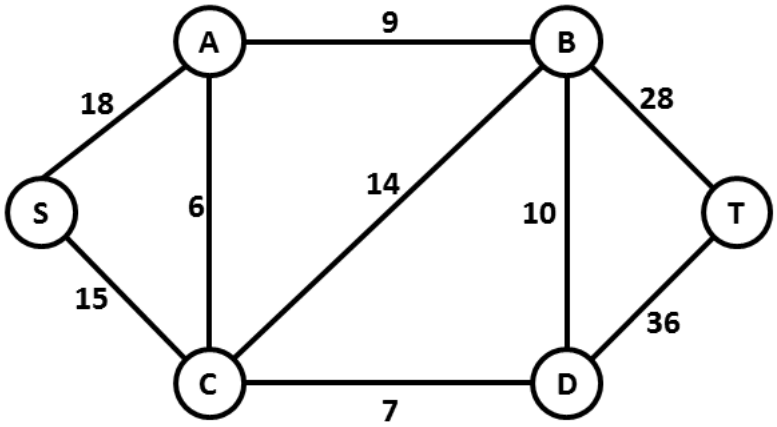
Tražimo vrh u Q s najmanjom udaljenosti. To je vrh C. Brišemo ga iz reda Q pa je stanje reda $Q = \{A, B, D, T\}$. Za svaki susjedni vrh od C računamo udaljenosti i vrh koji vodi do njega. Ukoliko je udaljenost manja od prethodne udaljenosti, postavljamo novu udaljenost na akumuliranu, a mijenjamo i put koji vodi do vrha. Dobivamo: $(A, 18, \{S\})$, $(B, 29, \{S, C\})$, $(D, 22, \{S, C\})$.

Tražimo vrh u Q s najmanjom udaljenosti. To je vrh A. Brišemo ga iz reda Q pa je stanje reda $Q = \{B, D, T\}$. Za svaki susjedni vrh od A računamo udaljenosti i vrh koji vodi do njega. Analogno akumuliramo udaljenosti i put kao i u prošlom slučaju. Dobivamo: $(B, 27, \{S, A\})$. Ostale vrijednosti se ne mijenjaju.

Tražimo vrh u Q s najmanjom udaljenosti. To je vrh D. Brišemo ga iz reda Q pa je stanje reda $Q = \{B, T\}$. Za svaki susjedni vrh od D računamo udaljenosti i vrh koji vodi do njega. Analogno akumuliramo udaljenosti i put kao i u prošlom slučaju. Dobivamo: $(B, 27, \{S, A\})$, $(T, 58, \{S, C, D\})$.

Tražimo vrh u Q s najmanjom udaljenosti. To je vrh B. Brišemo ga iz reda Q pa je stanje reda $Q = \{T\}$. Za svaki susjedni vrh od D računamo udaljenosti i vrh koji vodi do njega. Analogno akumuliramo udaljenosti i put kao i u prošlom slučaju. Dobivamo: $(T, 55, \{S, A, B\})$.

Tražimo vrh u Q s najmanjom udaljenosti. To je jedini preostali vrh, izbacujemo ga iz reda pa red ostaje prazan i pronašli smo najkraći put. On ima težinu 55 i vodi od vrha S preko vrhova A i B do vrha T.



Slika 3.5: Bridno - težinski graf

3.7 Ostali algoritmi za najkraći put

Prilikom traženja najkraćeg puta Dijkstrinim algoritmom težine bridova ne smiju biti negativni brojevi. U slučaju da jesu, koristi se algoritam Bellman-Ford. Njegove su performanse sporije od Dijkstrinog algoritma, no ponekad je neki problem nemoguće riješiti bez korištenja negativnih težina bridova.

Poopćenje Dijkstrinog algoritma koje pruža osjetna poboljšanja i ubrzanja je algoritam A*. On obično smanjuje broj vrhova grafa koje treba ispitati. Smanjenje broja vrhova zasnovano je na korištenju funkcije koja procjenjuje donju granicu udaljenosti od završnog vrha. Algoritam A* daje dobre rezultate samo ako je graf poznat u potpunosti. No, ukoliko se u grafu mijenjaju težine, koristi se algoritam D*.

Dijkstrin će nam algoritam riješiti problem najkraće udaljenosti između dva vrha. Ponekad može biti potrebno odrediti najkraću udaljenost između svih parova vrhova. Jedan od načina je izvršiti Dijkstrin algoritam za svaki par vrhova, no složenost tog načina je očito prevelika. Za rješenje problema koristi se algoritam Floyd-Warshall.

Poglavlje 4

Strategije u igrama

4.1 Uvod

Umjetna inteligencija danas ima primjenu u mnogim područjima. Neki od najpoznatijih primjera korištenja umjetne inteligencije su računalno jezikoslovlje, robotika, računalni vid, ali i računalne igre.

Igre razlikujemo prema broju igrača, i to kao igre za jednog igrača (sudoku, pasijans), igre za dva igrača (križić-kružić, šah, igra dame) te igre za više igrača (belot). Prema ishodu razlikujemo rezultat igre kao poraz, neriješeno ili pobjeda (šah, križić-kružić, igra dame) te rezultat igre kao određeni broj bodova (belot). Prema informiranosti razlikujemo savršeno i nesavršeno informirane igre. Savršeno informirane igre su one u kojima protivnik ima potpunu dostupnost informacija o stanju protivnika (šah, križić-kružić, igra dame), dok su nesavršeno informirane sve ostale. Deterministčke igre su igre u kojima ishod ne ovisi o sreći, već o umijeću igrača. Primjeri determinističkih igara su ponovno šah, križić kružić i igra dame. Stohastičke igre su one u kojima je sreća jedan od faktora odlučivanja o pobjedniku.

4.2 Napredan nivo u igrama

Savršeno informirane deterministčke igre za dva igrača

1769. godine Wolfgang von Kempelen stvorio je mehanički stroj za kojeg se činilo da igra šah protiv čovjeka. No, stroj je bio optička varka jer se unutar njega skrivao čovjek. 1912. godine Leonardo Torres y Quevedo napravio je stroj koji je mogao igrati beskonačnu igru topa i kralja protiv kralja. Claude Shannon je 1950. godine predložio šahovski uređaj, dok je Alan Turing 1951. godine napisao program za kompletnu partiju šaha. Šah je kao igra dominirao prilikom istraživanja kako čovjeku stvoriti nenadmašivog protivnika.

Programi za šah koji se danas koriste oslanjaju se na minimax algoritam te alfa beta podrezivanje. Isti algoritam koriste i programi za igru dame (eng. *checkers*) te križić-kružić. Sve tri igre imaju slična svojstva. Igre su za dva igrača, savršeno su informirane te su deterministične.

Ostale igre za dva igrača

Kako u mnogim kartaškim igrama ne znamo točne karte koje protivnik drži, moramo odrediti vjerojatnost da drži neku kombinaciju karata (ruku). Tada na temelju vjerojatnosti stvaramo ruku koju protivnik drži i dalje nastavljamo s algoritmom koji koristimo i kod savršeno informiranih igara. Navedena ideja je preteča Monte Carlo simulacije. Umjesto generiranja jedne ruke, generiraju se skupovi ruku koji reprezentiraju ruku za koju očekujemo da protivnik drži. Za vrijeme igre ažuriramo model na temelju protivnikovih poteza. Algoritmi za stohastičke igre i nesavršeno informirane igre su u razvoju i predmet su brojnih istraživanja za poboljšanje.

Igre za više igrača

Tri su najraširenije vrste igara za više igrača koje zahtijevaju računalo kao protivnika. Prva vrsta su *first-person shooter* (FPS) igre. To je vrsta računalne igre u kojoj igrač ima pogled iz perspektive glavnog lika. Primarni element tih igara je borba te najčešće uključuju vatreno oružje. FPS igre uobičajeno su brzog tempa pa od računala zahtijevaju brze reflekse i preciznost, ali oni ne smiju biti takvi da čovjek koji igra protiv računala uvijek izgubi. Najpopularnije FPS igre su igre iz serijala *Doom*, *Half-life* i *Call of Duty*. Zajedničko im je da igrač započinje igru kao glavni protagonist te mu je cilj preživjeti, odnosno ubiti protivnike na putu.

Druga vrste igara su *real-time strategy* (RTS) igre. To su igre u kojima igrač upravlja nekom jedinicom i razvija ju na određeni način kako bi porazio protivnika. Uobičajeno je u takvim igrama da je igrač pozicioniran u igraćem svijetu te mora graditi specifične strukture ili vojsku za napredovanje u igri. Skupljanje resursa u svrhu gradnje vojske ili struktura glavni je fokus RTS igara. Najpopularnije RTS igre su igre iz serijala *Warcraft* i *Age of Empires*.

Treća vrsta igara su kartaške igre i igre na ploči. To su igre koje se uobičajeno igraju i bez računala uz pomoć karata ili igraće ploče. Računalo u igrama služi kao zamjena za igrača. Postoje brojne kartaške igre i igre na ploči za više igrača. Neke od njih su poker, *Uno*, *Monopoly* i kineska igra dame.

Programeri FPS i RTS igara koriste različite metode kako bi osigurali pravednu igru jer nemaju problem napisati program koji će pobijediti čovjeka. Kod kartaških igara i igara na ploči problem je kako stvoriti jakog protivnika [4].

Algoritmi koji se koriste u igrama za dva igrača ne funkcioniraju kod igara za više igrača ili su neefikasni. Nažalost, ti algoritmi trenutno nisu predmet proučavanja i poboljšavanja. Ne postoji niti jedna deterministička igra za više igrača za koju je napisan program koji će pobijediti svakog čovjeka, odnosno koji će biti dostojan protivnik najboljim igračima.

4.3 Križić-kružić i minimax algoritam

Cilj je napisati program koji će biti nepobjediv u igri. Da bismo razmišljali o pobjedi u nekoj igri, potrebna nam je strategija. Strategija obično podrazumijeva da igrač zna sva moguća stanja igre kao i reakciju na određeno stanje. Ovo je izvedivo na malim igrama, ali ne i na velikim igrama, odnosno igrama s puno mogućih stanja. Strategija u igri je statična funkcija promjene i pravila odlučivanja. Statična funkcija promjene dodjeljuje vrijednost svakom listu u igraćem stablu, a pravilo odlučivanja kazuje kako se te vrijednosti šire duž stabla. Na primjer u šahu, jednostavna statična funkcija promjene će vratiti razliku u materijalnim vrijednostima figura između dva igrača ako je svakoj igraćoj figuri dodijeljena određena vrijednost (pijunu 1, kraljici 9). Dok je funkcija vrlo zavisna o području igre, pravila odlučivanja su primjenjiva na svim igrama. Prilikom izrade programa ključno je odabrati pravilo odlučivanja, a tek nakon toga pogodnu funkciju. Standardno pravilo odlučivanja u igrama za dva igrača je minimax.

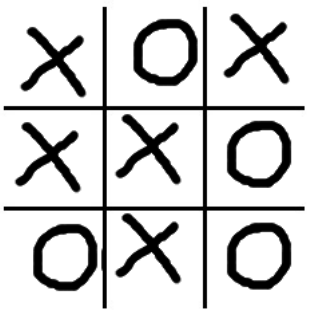
Promotrimo primjer savršeno informirane, determinističke igre za dva igrača. Križić-kružić je igra koja se igra na praznom 3×3 polju. Jedan igrač postavlja križiće, a drugi kružiće. Počevši od igrača s križićem, igrači naizmjenično odabiru prazna polja i unutar njih postavljaju svoj znak. Igrač pobjeđuje kada ostvari tri svoja uzastopna znaka u nekom retku, stupcu ili na dijagonali. Ukoliko to ne uspije ni jedan igrač, igra završava bez pobjednika.

Kako bi program bio nepobjediv, potreban je algoritam koji će odrediti sve moguće poteze računala i nekom metrikom odabrati najbolji mogući potez. Već smo rekli da se za ovaj problem koristi minimax algoritam.

Savršen program, odnosno program na naprednom nivou bi u svakom krugu igre ili pobijedio ili odigrao neriješeno. Dapače, ukoliko bi program igrao protiv nepobjedivog igrača, igra bi uvijek završila neriješeno. Igra križić-kružić dakle može završiti u tri stanja: pobjeda, neriješeno i poraz. Dodijelimo svakom od stanja numeričku vrijednost. Ukoliko program pobjedi, dobiva 1 bod, ukoliko odigra neriješeno, nitko ne dobiva bod, odnosno oboje dobivaju 0 bodova. Ukoliko igrač protiv kojeg program igra pobjedi, program gubi bod, odnosno dobiva -1 bod jer je bod dobio drugi igrač.

Primijetimo, problem križić-kružić igre definirali smo s četiri komponente. To su početno stanje, funkcija prijelaza, provjera završetka te funkcija korisnosti. Početno stanje, odnosno stanje iz kojeg igra započinje je prazna 3×3 ploča. Funkcija prijelaza koja za

dano stanje vraća skup svih stanja dosezivih iz početnog stanja je definirana akcijom postavi kružić ili križić u prazno polje. Da li je igra završena, provjeravamo tako da provjerimo postoje li tri kružića u nizu, tri križića u nizu ili su sva polja popunjena. Skup svih mogućih stanja može se prikazati stablom. Prvi nivo predstavljat će 9 mogućih stanja, a nakon toga iz svakog će od mogućih stanja na drugom nivou biti prikazano sljedećih 8 mogućih stanja. Funkciju korisnosti koja daje numeričku vrijednost završnim stanjima definirali smo tako da za pobjedu vraća 1, za poraz -1, a za neriješeno 0. Na primjer, završnom stanju na slici 4.1 dodijelili smo vrijednost 0.

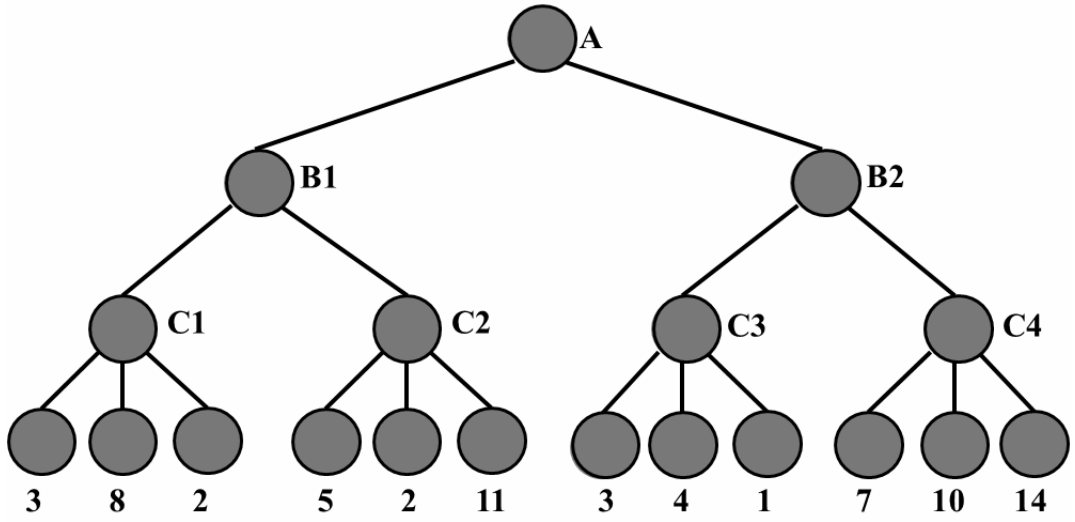


Slika 4.1: Završno stanje 0

Minimax je algoritam koji koristi činjenicu da dva igrača teže suprotnim ciljevima te tako predviđa koje će se buduće stanje u igri doseći. Algoritam optimizira šanse za pobjedu. Protivnik algoritma će pokušati minimizirati koju god vrijednost algoritam pokušava maksimizirati. Na primjeru križić-kružića protivnik će pokušati doseći -1, a program 1. Od tuda dolazi i naziv algoritma minimax. Program bi trebao napraviti potez koji protivniku ostavlja mogućnost napraviti najmanju moguću štetu.

U idealnom slučaju, kad bi računalo imalo beskonačno mnogo vremena za obradu i beskonačnu memoriju, istražilo bi svaki mogući ishod iz trenutnog stanja kao i puteve koji vode do tog stanja i svakom od ishoda dodijelilo bi vrijednost. Nakon toga, krenuvši od kraja (vrha koji prezentira završno stanje), računalo određuje koje je stanje najbolje za protivnika. Tada pretpostavlja da će protivnik odigrati optimalno, odnosno najbolje za sebe. Kako računalo pretpostavlja potez koji će protivnik odigrati, ima predodžbu o tome koje će biti završno stanje igre. Isti postupak računalo može provesti i za stanje prije završnog i tako sve do trenutnog stanja. Na kraju, svaka će od mogućnosti koje računalo ima biti određena nekim brojem, a računalo će odabrati onu s najvećom vrijednošću. Promotrimo primjer na slici 4.2. Računalo ima izbor na nivou A i nivou C, a protivnik na nivou B. Računalo pokušava maksimizirati rezultat pa će na nivou C odabrati onaj čvor s najvećom

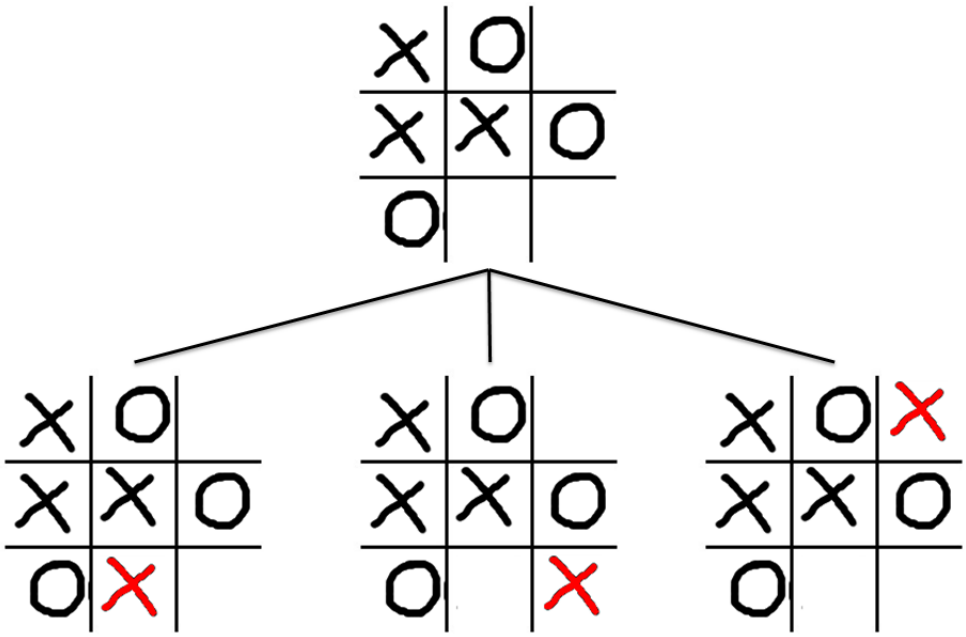
vrijednošću. Dakle vrijednosti čvorova na nivou C su redom: C1 ima vrijednost 8, C2 ima vrijednost 11, C3 ima vrijednost 4, C4 ima vrijednost 14. Kada protivnik izabire čvor na nivou B, on pokušava odabrati put koji vodi do grane C s najmanjom završnom vrijednošću. Dakle, vrijednosti svakog od B čvorova bit će minimum vrijednosti na C čvorovima koji od njih slijede. Vrijednost B1 čvora je 8, a vrijednost B2 čvora 4. Sada smo na početnom odabiru. Računalo sada zna što će se dogoditi ukoliko odaberemo B1 ili B2 čvor. I kakav god kraj igre bio, računalo odabire čvor s najvećom vrijednošću od ove dvije. Ukoliko se protivnik ne ponaša optimalno, odnosno ukoliko ne igra najbolje moguće poteze po sebe, rezultat čvorova se ponovno računa, a rezultat je dobar kao prethodni ili bolji po računalo.



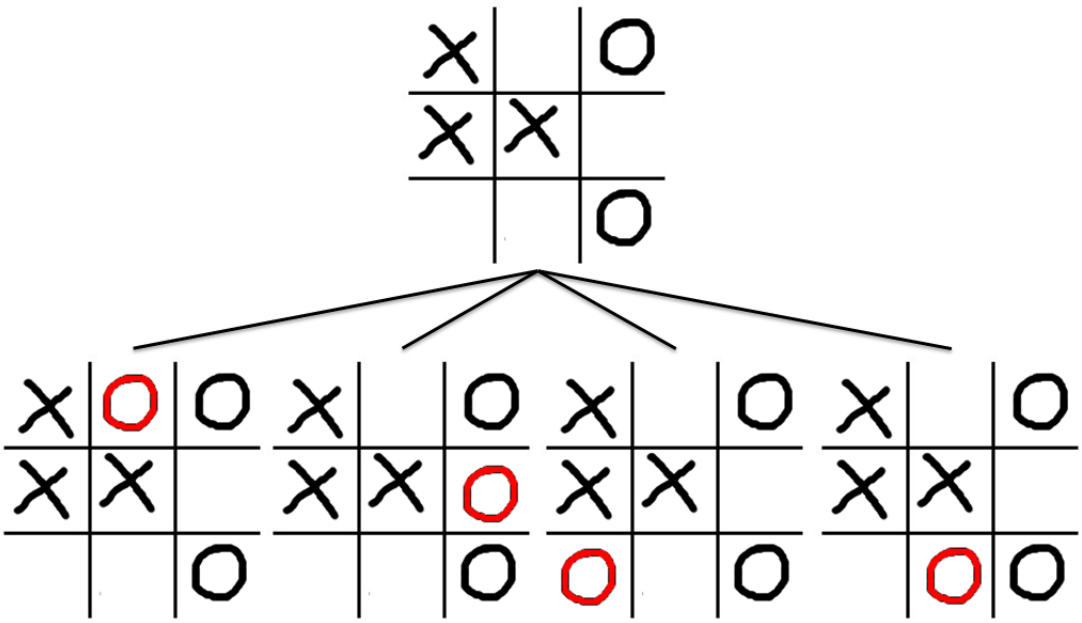
Slika 4.2: Minimax stablo

Promotrimo primjer na slici 4.3. Red je na igraču koji postavlja križiće. Tri su moguća poteza, od čega je jedan potez pobjednički. Kako jedno od stanja omogućuje igraču s križićem da pobjedi, a njegov je red, stanje na gornjem nivou jednako je dobro za igrača kao i stanje na donjem nivou. Dakle, njegova je vrijednost 1. Vrijednosti međustanja u kojima je križić sljedeći na potezu postavljamo na maksimalnu vrijednost sljedećih mogućih stanja.

Pitamo se što ako je kružić na potezu. Promotrimo primjer na slici 4.4. Kružić je na potezu pa su moguća četiri prikazana stanja. Jedno od stanja prikazuje pobjedu kružića. Iz perspektive križića, to stanje je poraz, odnosno ima vrijednost -1. Istu vrijednost pridodajemo i prethodnom stanju u kojem je kružić na potezu jer pretpostavljamo da će kružić iskoristiti mogućnost pobjede. Dakle, vrijednost međustanja gdje je kružić na potezu postavljamo na minimalnu vrijednost mogućih sljedećih stanja.



Slika 4.3: Stablo stanja - na potezu križić



Slika 4.4: Stablo stanja - na potezu kružić

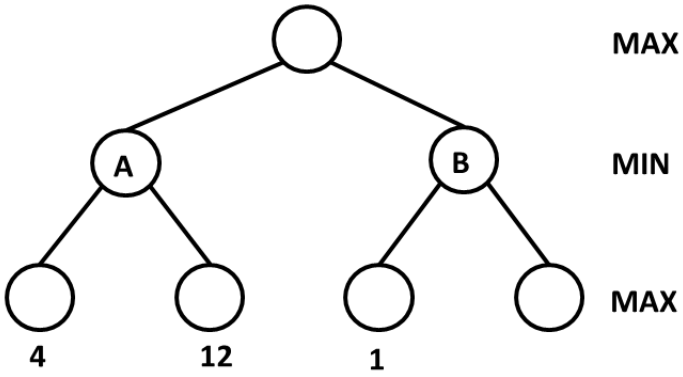
Primijetimo da će minimax algoritam kod nekih drugih igara žrtvovati velike pobjede, odnosno one s velikim vrijednostima završnih čvorova zbog toga što pretpostavlja da će protivnik predvidjeti sve poteze koji do njega vode. Iscrpno korištenje minimax algoritma je nepraktično. Računalo može izračunati sve moguće poteze u igri križić-kružić, ali to mu neće pomoći ukoliko igra s protivnikom koji igra optimalno. Čak i u šahu, kada bi računalo i moglo odrediti sva moguća stanja igre, sama informacije o tome vodit će do neriješenog rezultata protiv optimalnog protivnika. Minimax je dobar algoritam za one igre u kojima je pobjeda sigurna ukoliko je igrač prvi ili drugi na potezu, odnosno u nepravednim igrama. Međutim, minimax je koristan algoritam jer iako računalo ne može odrediti sva moguća stanja do kraja igre od njena početka, može odrediti koji potez vjerojatnije vodi do materijalne prednosti.

4.4 Alfa-beta podrezivanje

Uočili smo da minimax algoritam ima eksponencijalnu vremensku složenost zbog toga što obilazi sve čvorove. Jedno od poboljšanja algoritma minimax je alfa-beta podrezivanje. Iako eksponent nije moguće eliminirati u potpunosti, moguće ga je prepoloviti. Alfa-beta podrezivanjem odredit ćemo najbolji mogući potez koji će odrediti i minimax algoritam, ali bez pregledavanja svakog čvora u stablu. Alfa-beta podrezivanje može biti primijenjeno na stablu bilo koje visine, a često odbacuje cijela podstabla prilikom pretraživanja. Promatramo čvor n negdje u stablu u koje se može doći. Ukoliko postoji stanje m koje je roditelj ili je iznad čvora n, a bolji je izbor od n, do čvora n nećemo doći. Kada imamo dovoljno podataka o n da bismo došli do ovog zaključka, možemo ga podrezati. Alfa-beta podrezivanje dobilo je ime po parametrima α i β. Parametar α je vrijednost najboljeg izbora koji smo pronašli duž puta za MAX, a β vrijednost najboljeg izbora koji smo pronašli duž puta za MIN.

Promotrimo primjer na slici 4.5. Vrijednost stanja A je 4 jer protivnik želi minimizirati vrijednost završnog stanja. Također, prva pronađena vrijednost podstanja od B je 1. Budući da je na potezu igrač koji želi minimizirati vrijednost završnog stanja, znamo da vrijednost podstanja od B mora biti manja ili jednaka 1. Također, znamo da je vrijednost stanja A 4, a A i B imaju istog roditelja na nivou u kojem igrač maksimizira vrijednost. To znači da put kroz stanje B neće biti izabran, jer je 4 bolji izbor od 1 ili manje na koji vodi to stanje. Dakle, podstanja nakon B ne moramo promatrati.

Primijetimo da je alfa-beta podrezivanje jako ovisno o redoslijedu pregledavanja stanja, odnosno čvorova stabla. Bez obzira na podrezivanje, i dalje u složenijim igrama imamo velika stabla za pregledavanje. Za rješenje tog problema koristi se već spomenuta evaluacijska funkcija. Ona vraća procjenu očekivane korisnosti izbora stanja. Na isti način i ljudi nesavjesno igraju igru, a ne pregledavaju sva moguća stanja.



Slika 4.5: Alfa-beta podrezivanje

4.5 Algoritmi za više igrača

Paranoičan algoritam odlučivanja je najjednostavniji algoritam odlučivanja u igrama za više igrača. Algoritam pretpostavlja da u igri za n igrača njih $n-1$ igra protiv jednog čime problem više igrača svodi na problem dva igrača. No, on ne daje uvijek potpuno točne rezultate. Rezultat igrača prikazuje se pomoću $n-$torke, gdje je na $i-$tom mjestu rezultat $i-$tog igrača. Kako bi igru preveli u igru za dvoje igrača, zamjenjujemo $n-$torku razlikom između rezultata prvog igrača i zbroja rezultata ostalih igrača. Dalje nastavljamo kao i kod minimax algoritma.

Algoritam $\max^n$ je generalizacija algoritma minimax. Za igru za dva igrača $\max^n$ se svodi na minimax algoritam. U igri za n igrača svako je stanje reprezentirano $n-$torkom, gdje je $i-$ti element $n-$torke rezultat $i-$tog igrača. U stanjima prethodnicima vrijednost $\max^n$ stanja u kojem je igrač i na potezu je $\max^n$ vrijednost djeteta tog stanja, za koje je $i-$ti element maksimalan. Više o algoritmima za više igrača opisano je u [4].

Poglavlje 5

Fizika u igrama

5.1 Uvod

Kada udarimo neki objekt u realnom svijetu koji želimo srušiti, on se ne sruši odmah na pod niti vječno pada. Očekujemo da padne u ovisnosti gdje smo ga udarili i ako je moguće, odbije se od podloge na koju pada. Dakle, želimo igraći svijet u kojem će se objekti ponašati kao u realnom svijetu.

Grafika i zvuk su najviše napredovali u evoluciji računalnih igara. Današnje igre imaju idealnu grafiku koja teško da može uznapredovati, realan zvuk i razvijenu umjetnu inteligenciju koja pobjeđuje i najbolje igrače. No, kada se objekti u igri ne ponašaju kao što to rade u realnom svijetu, igra nema smisla. To je razlog zbog čega je fizika dio računalnih igara od samih početaka. Iako se sve do nedavno nije toliko brinulo o realnosti u igrama, danas su fizičke zakonitosti uz matematiku jedan od ključnih alata prilikom izrade dobre i realne računalne igre. U jednom od prethodnih poglavlja spomenuto je da je potrebno odrediti kada se dva objekta presijecaju. No, na svako presijecanje očekujemo neku reakciju objekata kao što je to u realnom svijetu. Objekt ili neće moći proći drugi objekt ili će se odbiti od njega ili će se oba objekta odbiti jedan od drugoga.

Prva primjena fizike u računalnim igrama vezana je uz igru *Pong* nastalu 1972. godine. Od igre *Half-life* 2 koja je izašla 2004. godine fizika se koristi u kompliciranije svrhe. Na primjer, gravitacija se može koristiti za računanje putanje artiljerije, a sustavi čestica se mogu koristiti za simulaciju vatre, eksplozija i dima [2].

Programiranje realističnih fizičkih modela očito nije lagan zadatak zbog čega programeri igara koriste softvere za stvaranje fizikalnih modela (eng. *physics engine*).

5.2 Kretanje stalnim ubrzanjem

Opišimo jednostavan fizikalan problem koji se svodi na primjenu analitike u prostoru. Promotrimo kretanje objekta kao funkciju P koja vraća položaj objekta za svaki trenutak t. Kako $P(t)$ predstavlja položaj objekta u trenutku t, reprezentira se pomoću koordinata, odnosno za slučaj dvodimenzionalnog igraćeg svijeta $P(t) = (x(t), y(t))$, a u slučaju trodimenzionalog igraćeg svijeta $P(t) = (x(t), y(t), z(t))$. Ukoliko objekt konstantno translatiramo u nekom smjeru, imamo jednoliko gibanje. Brzinu objekta kojom se on giba možemo odrediti kao omjer prijeđenog puta i proteklog vremena. Ovo gibanje možemo opisati linearnom funkcijom danom pravilom pridruživanja:

$$P(t) = P_0 + tv_0,$$

gdje P_0 predstavlja početni položaj objekta, v_0 je vektor kojim se objekt kreće, a t je količina vremena u kojem se objekt giba. Vektor v_0 je, kao i $P(t)$, reprezentiran pomoću koordinatnog zapisa. Derivacijom funkcije P po varijabli t dobivamo:

$$P'(t) = v_0.$$

Derivaciju funkcije položaja zovemo brzina. No, u realnom se svijetu objekti ne kreću konstantnom brzinom već ubrzavaju, odnosno usporavaju. Primijetimo da ukoliko još jednom deriviramo deriviranu funkciju $P'(t)$, dobit ćemo nulu. Pretpostavimo da je druga derivacija različita od nule. Time mijenjamo funkciju brzine u

$$v(t) = v_0 + ta. \tag{5.1}$$

Derivacija funkcije brzine a naziva se akceleracija. Ovo gibanje je jednoliko ubrzano gibanje. Objekt u jednakim vremenskim intervalima dobiva jednaki prirast brzine. Prepravimo i funkciju položaja objekta. Integriranjem (5.1) dobivamo funkciju položaja u ovisnosti o vremenu:

$$P(t) = P_0 + tv_0 + \frac{1}{2}t^2a.$$

Okvir (eng. *frame*) je slika koja se prikazuje na zaslonu računala. Očito slika u igrama nije statična, već se okviri, to jest slike na računalu brzo izmjenjuju. Time dobivamo dojam kretanja objekata. Jedno od najčešćih mjerila koje se koristi u mjerenju grafičke performanse igre je broj okvira po sekundi (eng. *frames per second*). Današnje igre uobičajeno postižu od 30 do 60 okvira po sekundi. Dakle, nije potrebno konstantno mjeriti položaj objekta u igri, već je potrebno odrediti položaj u trenucima u kojima se objekt prikazuje na zaslonu. Uočimo, za igraći okvir i, položaj P_{i+1} i brzina v_{i+1} objekta u sljedećem igraćem okviru $i + 1$ dana je s:

$$P_{i+1}(t) = P_i + tv_i + \frac{1}{2}t^2a_i \tag{5.2}$$

$$v_{i+1}(t) = v_i + ta_i, \tag{5.3}$$

gdje je t duljina trajanja igraćeg okvira i. Za danu početnu poziciju, brzinu i akceleraciju objekta koja je konstantna duž intervala kojeg promatramo možemo izračunati položaj objekta u svakom trenutku tog intervala.

5.3 Sila i masa

Postavlja se problem kako izračunati vrijednost akceleracije. To se radi pomoću fizikalne veličine sile. Sila uzrokuje promjenu brzine tijela, usporava ga ili ga ubrzava. Silom ruke djelujemo na loptu kako bismo je bacili i uzrokovali njezino gibanje po zraku. Ta sila uzrokuje akceleraciju proporcionalnu masi objekta na koju djelujemo. Proporcionalna veza mase i akceleracije dana je drugim Newtonovim zakonom:

$$\vec{F} = m\vec{a}. \tag{5.4}$$

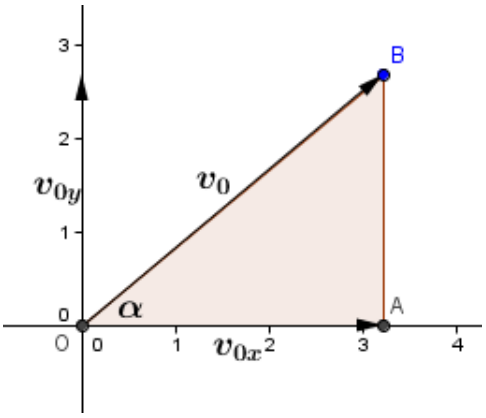
Ovdje pretpostavljamo da se masa tijelu ne mijenja tijekom promjene brzine. Primjer gdje to nije tako je gibanje rakete. Njoj se masa prilikom ubrzavanja značajno smanjuje jer velika količina goriva izgara u kratkom vremenu. Uobičajeno, na objekt u jednom trenutku djeluje više sila. Prilikom bacanja loptice, najprije mi djelujemo silom kako bismo je bacili, nakon toga na nju djeluje sila teža, odnosno gravitacija te uz to na nju djeluje otpor zraka i vjetar. Kako su sile vektori, možemo zbrajati sile i stvoriti jedinstvenu silu koja predstavlja njihov učinak na objekt. Tada skaliramo dobiveni vektor s $\frac{1}{m}$ da bismo dobili akceleraciju.

Opća formulacija temeljnog zakona gibanja, bez ograničenja da se masa tijelu ne mijenja tijekom promjene brzine bila bi da je brzina promjene količine gibanja objekta jednaka sili koja djeluje na objekt. Količina gibanja je fizikalna veličina koja određuje stanje gibanja tijela, odnosno to je umnožak mase tijela i brzine tijela. To je sklonost objekta da ostane u trenutnom stanju gibanja. Očito je za teže tijelo ili tijelo veće brzine potrebna veća sila kako bismo promijenili njegovu brzinu. To možemo objasniti na sljedeći način: iako je kamenčić u stanju mirovanja lakše pomaknuti nego stijenu to ne mora biti tako ukoliko je kamenčić ispaljen velikom brzinom. Prilikom određivanja reakcije objekta na sudar vrlo je važan zakon očuvanja količine gibanja: Količina gibanja izoliranog sustava je konstantna, odnosno promjena količine gibanja u vremenu u izoliranom sustavu jednaka je nuli. Izoliran sustav je onaj u kojem u obzir uzimamo samo sile unutar tog sustava, odnosno sile između na primjer objekata u igraćem svijetu.

5.4 Kosi hitac

Pretpostavimo da imamo projektil s početnom brzinom v_0 i položajem P_0. Budući da kretanje objekta ne ovisi o njegovoj težini, dok god projektil značajno ne promjeni položaj u

visinu, njegov let ima konstantu negativnu akceleraciju. Ta je akceleracija ovisna o gravitaciji pa ukoliko ignoriramo otpor zraka, brzina i položaj objekta mijenjat će se u ovisnosti o toj akceleraciji. Za gravitaciju u Zemljinom polju uobičajeno se uzima konstanta $g = 9.8m/s^2$. Kako promatramo položaj projektila koji se kreće uvis, riječ je o vertikalnom hicu i dolazi do promjene samo u smjeru $y-$osi ukoliko je riječ o dvodimenzionalnom igraćem svijetu. Ponovno možemo računati položaj i brzinu objekta kao kod (5.2) i (5.3), gdje je $a_i = (0, -g)$, a $v_i = (0, v)$.



Slika 5.1: Kosi hitac

Projektil ne mora putovati vertikalno s obzirom na igraći svijet. Primjer takvog gibanja je ispaljivanje kugle iz topa. Vektori akceleracije i brzine nisu kolinearni. Kretanje tijela koje je bačeno s početnom brzinom pod nekim kutom naziva se kosi hitac. Top ispaljuje kuglu pod točnim kutem i brzinom. Kut kojeg određuju vektori brzine i vektor paralelan s površinom iz koje se hitac događa naziva se nagibni kut. Duljina između ravnine iz koje je ispaljen hitac do najvišeg vrha do kojeg objekt doseže naziva se visina hica, a udaljenost od početne točke iz koje se ispaljuje hitac do točke do koje objekt doseže naziva se domet hica. Da bismo mogli odrediti položaj tijela u nekom trenutku t moramo izvesti jednadžbu kosog hica.

Neka je O, odnosno ishodište, točka iz koje se događa hitac i v_0 vektor početne brzine. Neka je P funkcija položaja objekta mase m u trenutku t. $P(t) = (x(t), y(t))$ je položaj tog objekta u trenutku t. Kada ne bi djelovala sila teža, tijelo bi se jednoliko gibalo duž vektora v_0. Razlažemo vektor v_0 na dvije komponente, jednu horizontalnu te jednu vertikalnu. Iz pravokutnog trokuta OAB sa slike 5.1 dobivamo:

$$v_{0x} = v_0 \cos\alpha = x'(0) \tag{5.5}$$

$$v_{0y} = v_0 \sin\alpha = y'(0). \tag{5.6}$$

U vertikalnom smjeru prema dolje djeluje sila teža, odnosno:

$$F = (0, -mg).$$

Prema drugom Newtonovom zakonu (5.4) vrijedi:

$$F = mP''(t),$$

gdje je $P''(t)$ druga derivacija funkcije položaja objekta u trenutku t. Dakle, vrijedi:

$$m(x''(t), y''(t)) = (0, -mg)$$

iz čega je:

$$\begin{aligned} x''(t) &= 0 \\ y''(t) &= -g. \end{aligned}$$

Integriranjem dobivamo:

$$\begin{aligned} x(t) &= At + B \\ y(t) &= -g\frac{t^2}{2} + Ct + D. \end{aligned}$$

Zbog toga što je početni položaj objekta u ishodištu, odnosno $x(0) = 0$ te zbog (5.5) vrijedi da je:

$$x(t) = v_0 t \cos\alpha.$$

Također, zbog toga što je $y(0) = 0$ te zbog (5.6) vrijedi da je:

$$y(t) = v_0 t \sin\alpha - \frac{1}{2}gt^2.$$

Konačno dobivamo:

$$P(t) = \left(v_0 t \cos\alpha, v_0 t \sin\alpha - \frac{1}{2}gt^2\right).$$

Primijetimo da je do sada bilo riječi samo o stalnoj akceleraciji. No, moguće je da akceleracija tijekom nekog puta ne bude stalna. Tada računanje položaja objekta nije jednostavno kao što je to kod jednoliko ubrzanog gibanja. Prilikom izvođenja izraza dolazimo do problema rješavanja diferencijalnih jednadžbi koje ne moraju biti rješive analitičkim metodama. Umjesto toga koriste se numeričke metode integracije.

5.5 Lopta koja odskače

Zamislimo da držimo loptu u zraku te je nakon toga pustimo da padne na tlo. Pošto ne djelujemo silom na nju, na nju djeluje samo gravitacija. Već je rečeno da je akceleracija promjena brzine u vremenu. Kada pustimo loptu, njezina je početna brzina nula, no njezina akceleracija nije nula, već ukoliko je puštamo na Zemlji otprilike 9.81 m/s^2. To znači da će njezina brzina rasti. Također, kako bismo na zaslonu dobili lijepi prikaz lopte, želimo da između dva okvira prođe što je moguće manje vremena. U igrama broj okvira po sekundi varira. Dakle, prema (5.3), brzina za igraći okvir $i + 1$ dana je s:

$$v_{i+1}(t) = v_i + t \cdot (0, -9.81),$$

gdje je t vrijeme između dva okvira. Položaj lopte za igraći okvir $i + 1$ je prema jednadžbi (5.2):

$$P_{i+1}(t) = P_i + tv_i + \frac{t^2}{2}(0, -9.81).$$

Kada lopta padne na tlo, ona bi trebala odskočiti. Najprije trebamo prepoznati dodir lopte i tla, nakon čega moramo uspostaviti pravilnu reakciju. Znamo iz iskustva da će lopta nakon prvog odskoka skočiti na manju visinu nego li je visina s koje smo je spustili. No, promotrimo najprije savršeno elastični sudar tla i lopte. Elastični sudar je sudar u kojem je energija u sudaru očuvana. Nakon što program prepozna sudar, odnosno udarac lopte u tlo, loptica očito mijenja brzinu, odnosno vektor brzine. U slučaju savršeno elastičnog sudara vektor brzine samo će promijeniti predznak prema zakonu o očuvanju energije.

Koeficijent restitucije je mjera elastičnosti sudara. On je omjer brzine prije sudara i brzine poslije sudara. Ukoliko je koeficijent restitucije jednak 1, lopta će od tla odskočiti istom brzinom kojom je i doskočila. Primijetimo, to je slučaj savršeno elastičnog sudara. Ukoliko je koeficijent restitucije 0 lopta će ostati na tlu. Neka je v_f brzina prilikom sudara i v_i brzina nakon sudara. Tada je koeficijent restitucije:

$$C = \frac{v_f}{v_i}.$$

Koeficijent restitucije pronalazi se pokusom. Kada se odbije od podloge brzina lopte bit će dakle

$$v_f = Cv_i.$$

Bibliografija

[1] Howard Anton, *Elementary linear algebra*, John Wiley & Sons, 2010.

[2] John Patrick Flynt i Danny Kodicek, *Mathematics and Physics for Programmers*, Course Technology PTR, 2012.

[3] C.A. Shaffer, *Data Structures & Algorithm Analysis in C++*, Dover Books on Computer Science Series, Dover Publications, 2011, ISBN 9780486485829, `http://books.google.hr/books?id=0HvFCe1Ez2UC`.

[4] Nathan Reed Sturtevant, *Multi-player games: Algorithms and approaches*, Disertacija, Citeseer, 2003.

[5] James M Van Verth i Lars M Bishop, *Essential mathematics for games and interactive applications: a programmer's Guide*, CRC Press, 2008.

Sažetak

U radu su opisani različiti tipovi problema s kojima se programer može susresti prilikom izrade igre. To je najprije reprezentacija objekata na računalu te prevođenje objekta iz trodimenzionalnog svijeta u dvodimenzionalni. Objašnjen je problem prepoznavanja sudara dvaju objekata. U radu je detaljno opisano rješenje tog problema pomoću smještanja objekata u sferu ili AABB objekt, a spomenuta su još neka moguća rješenja. Opisan je problem pronalaženja najkraćeg puta kao i njegovo rješenje Dijkstrinim algoritmom te različitim implementacijama grafa. Naglašena je važnost strategije u igri te je opisan minimax algoritam za strategiju igranja savršeno informiranih determinističkih igara za dva igrača, kao i njegova optimizacija. Također, u radu je spomenuta važnost fizikalnih zakonitosti i njihovih primjena prilikom izrade igre.

Summary

This paper describes various problems which programmers face during the process of creating a video game. The first problem is representation of objects in a computer and turning objects from the three-dimensional to the two-dimensional space. The problem of collision detection of two objects is also explained. The paper describes solutions of that problem using a sphere or an AABB object as surrounding objects. In the paper are mentioned some other solutions, too. It describes the shortest path problem and the solution using Dijkstra's algorithm and various implementations of graphs. This paper emphasizes the importance of strategy in the game and describes minimax algorithm for deterministic games where two players are perfectly informed. There is also mentioned importance of physical laws and their use during the process of creating a video game.

Životopis

Moje ime je Kristian Golubić. Rođen sam 9.2.1991. u Zagrebu. Pohađao sam Osnovnu školu Tituša Brezovačkog (1997.-2005.). 2005. godine upisao sam Gimnaziju Lucijana Vranjanina gdje sam maturirao 2009. godine na temu Ustav i ustavne promjene Republike Hrvatske. Tri godine kasnije, 2012. završavam Preddiplomski sveučilišni studij matematike (smjer: nastavnički) i upisujem Diplomski sveučilišni studij matematike i informatike, smjer: nastavnički na Prirodoslovno - matematičkom fakultetu u Zagrebu.