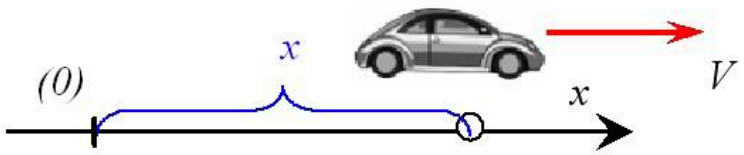
Co-funded by the Erasmus+ Programme of the European Union

TOPIC PLAN		
Partner organization	Belgrade Metropolitan University	
Topic	Game Physics Link: http://opencourses.metropolitan.ac.rs Username: futuremath Password: Futuremath123	
Lesson title	Differential equations of motion	
Learning objectives	Student can understand differential equations of motion: integration of velocities and displacements, according to force driven acceleration. Student can write a code for integration of motion. Student can use differential equations of motion to calculate velocities and displacements according to input force.	**Strategies/Activities** ☐Graphic Organizer ☐Think/Pair/Share ☑Modeling ☑Collaborative learning ☑Discussion questions ☐Project based learning ☑Problem based learning
Aim of the lecture / Description of the practical problem	The aim of this lecture is to learn how to use a differential equations of motion to move game objects in a video game using programing. Practical problem: Isprogramirati kretanje razlicitih tela u video igri. Kretanje projektila primenom kosog hica, pod uticajem gravitacije. Uticaj sile trenja na kretanje loptice u igri Stoni hokej.	
Previous knowledge assumed:	Basics of differential equations. Basics of physics and notion as: movement, force, acceleration, velocity and position. Knowing of vectors. Basics of calculus.	**Assessment for learning** ☑Observations ☑Conversations ☑Work sample ☐Conference ☐Check list ☐Diagnostics

Introduction / Theoretical basics	**BRZINA** Brzina je pređeni put u jedinici vremena. Srednja brzina je količnik pređenog puta i proteklog vremena Neka se objekat kreće duž prave (ili krive) na kojoj smo izabrali tačku x, od koje ćemo meriti pređeno rastojanje, i koordinatni sistem (O,x). Neka se u početnom trenutku t=0 telo nalazi u početnom položaju x = 0, i neka se u proizvoljnom trenutku t telo nalazi na rastojanju x od početnog položaja. Podsetimo se da je: Brzina = pređeni put u jedinici vremena.  Figure 1. Neka je od i-tog do (i+1)-og frejma proteklo vreme △t (tj. jedan frejm traje △t), i neka je telo za to vreme prešlo rastojanje △x . Srednju brzinu kretanja tokom intervala ćemo sračunati kao pređeno rastojanje u jedinici vremena: $$V_{sr} = \frac{\triangle x}{\triangle t}$$ Ako znamo srednju brzinu tokom (i+1)-og frejma, poziciju (position) za prikazivanje na ekranu možemo sračunati kao: $$x_{i+1} = x_i + \triangle x = x_i + V_{sr}\triangle t$$ Ako želimo tačnu brzinu u trenutku t (takozvanu trenutnu brzinu), tada treba smanjivati Δt. Kad Δt → 0 (tj. Δt teži nuli) dobija se da je trenutna brzina u trenutku t: V = dx/dt	**Assessment as learning** ☑Self-assessment ☐Peer-assessment ☐Presentation ☐Graphic Organizer ☐Homework **Assessment of learning** ☑Test ☑Quiz ☐Presentation ☐Project ☐Published work

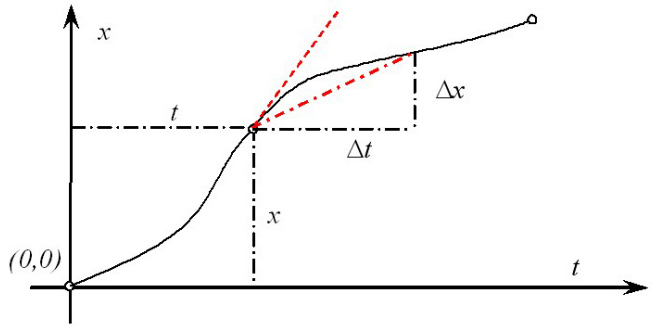


Figure 2. Trenutna brzina je predjeni put u beskonačno kratkom vremenu dt

tj. trenutna brzina je jednaka izvodu pređenog rastojanja po vremenu.
Ako nacrtamo grafik pozicije x u funkciji pređenog vremena t, tada je trenutna brzina jednaka
nagibu tangente na dobijenu krivu u tački (t,x) , dok je srednja brzina jednaka nagibu sečice
kroz datu tačku (Slika-3).
Važi takođe da je pozicija integral trenutne brzine po vremenu.

PRIMENA BRZINE U IGRAMA

Brzina i trajanje frejma određuju promenu pozicije (update) tokom frejma
Za igre je puno interesantnija sledeća intepretacija. Izaberimo interval posmatranja Δt (odnosno trajanje frejma) dovoljno mali tako da se brzina veoma V malo menja u toku Δt.
Neka je Vi brzina u i-tom frejmu, pređeno rastojanje u i-tom frejmu je jednako ViΔt. Da bi se dobili ukupni put i vreme nakon n frejmova, treba sumirati priraštaje u svakom frejmu:
$x = \Sigma_i V_i \Delta t$, $t = \Sigma_i \Delta t$, $i = 1,2, ..., n$
U igri ili simulaciji, integraljenje (sumiranje) se odvija iterativno tako da se u svakom frejmu sračunava tekuća pozicija:

```
// Inicijalizacija
float x = 0, t = 0,
V = V_initial, dt = 0.033;
while(!game_over)
{
    t = t + dt ;
    V = get_velocity ( t) ;
    x = x + V*dt ;
}
```

Figure 3.

Srednje ubrzanje je količnik promene brzine i proteklog vremena
Neka je u frejmu i (odnosno trenutku ti) brzina tela jednaka V, a neka je u frejmu (i+1) tj.
trenutku ti + △t brzina tela jednaka V + △V , Slika-2.
Srednje ubrzanje tokom intervala △t je jednako:
Ako znamo srednje ubrzanje tokom (i+1)-og frejma, brzinu za prikazivanje na ekranu možemo
sračunati kao:
Poziciju tokom (i+1)-og frejma možemo ažurirati stavljajući da je srednja brzina jednaka:

$$V_{sr} = (V_i + V_{i+1})/2 = V_i + a_{sr}\triangle t/2$$

tako da je:

$$x_{i+1} = x_i + \triangle x = x_i + V_{sr}\triangle t$$

TRENUTNO UBRZANJE

Trenutno ubrzanje je jednako izvodu brzine po vremenu
Ako želimo tačno ubrzanje u trenutku t (takozvano trenutno ubrzanje), tada treba smanjivati
Δt.
Kad Δt → 0 (tj. kada teži nuli Δt)dobija se da je trenutno ubrzanje u trenutku t
a = dV / dt

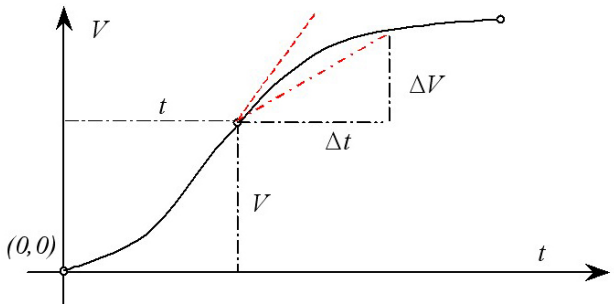


Figure 4.

tj.
Trenutno ubrzanje je jednako izvodu brzine po vremenu.
Obrnuto,
Brzina je integral trenutnog ubrzanja po vremenu.
Ako nacrtamo grafik brzine V u funkciji vremena t, trenutno ubrzanje je jednako nagibu
tangente na dobijenu krivu u tački (t,V) , dok je srednje ubrzanje jednaka nagibu sečice kroz
datu tačku (Slika-3).

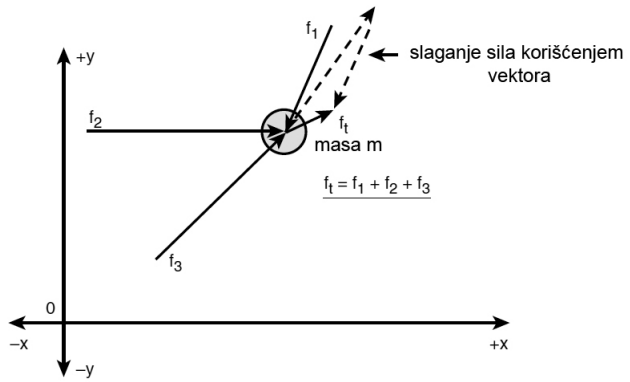
SILA I DRUGI NJUTNOV ZAKON

Da bi telo promenilo to stanje kretanja (tj. ubrzalo ili usporilo), potrebno je da se deluje na njega iz okoline, a ta akcija se naziva sila. Jedinica za silu je Njutn
Svako telo koje se kreće pravolinijski i konstantnom brzinom teži da zadrži to stanje kretanja.
Ovo je poznato kao zakon inercije. Da bi telo promenilo to stanje kretanja (tj. ubrzalo ili usporilo), potrebno je da se deluje na njega iz okoline, a ta akcija se naziva sila.
Poznati II Njutnov zakon kaže da je:
F = m a
gde je F sila, m masa tela, a a ubrzanje.
Ako na telo deluje više sila F1 , F2 , ... Fn , tada se njihova suma naziva rezultanta.
Pri dejstvu više sila II Njutnov zakon glasi:
m a = Σi Fi

<u>Drugi Njutnov zakon u 2D prostoru</u>

Pogledajmo npr. Sliku-3 na kojoj su naznačene 3 sile koje deluju

na materijalnu tačku mase m u 2D ravni. Rezultujuća sila koja deluje na tačku je jednostavno suma svih sila koje deluju na tu tačku.



Formulacija II Njutnovog zakona ostaje ista kao i u jednodimenzionalnom slučaju:
Ubrzanje tela je proporcionalno sumi sila koje deluju na njega m $a = \Sigma_i F_i$

Diferencijalne jednačine kretanja

Napišimo sledeći set relacija koje smo već objasnili u prethodnom tekstu:

$$\frac{dV_x}{dt} = (\sum_i^n F_{ix})/m \qquad \frac{dx}{dt} = V_x$$

$$\frac{dV_y}{dt} = (\sum_i^n F_{iy})/m \qquad \frac{dy}{dt} = V_y$$

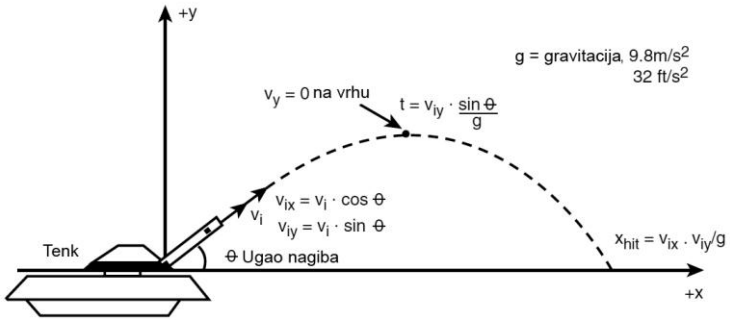
U i igri i simulaciji računamo u svakom frejmu veličine x, y, Vx, Vy koje se nalaze pod znakom izvoda u gornjim jednačinama, koje se stoga nazivaju diferencijalne jednačine.
Postupak ažuriranja pozicija i brzina koji smo izložili je u stvari numeričko rešavanje diferencijalnih jednačina i to jednom od najprostijih metoda – Ojlerovom metodom.

Petlja za ažuriranje frejma može sada biti napisana na sledeći način:

```
class SpaceShip
{
private:
    float  m_Position;
    float  m_Velocity;
    float  m_fMass;
public:
    …
};

void SpaceShip::Update(float TimeElapsedSinceLastUpdate,
                       float ForceOnShip)
{
    float acceleration = ForceOnShip / m_fMass;
    m_Velocity  += acceleration * TimeElapsedSinceLastUpdate;
    m_vPosition += m_Velocity   * TimeElapsedSinceLastUpdate;
}
```

Primena II Njutnovog zakona na kretanje projektila

Razmotrimo sada primenu izložene materije na problem koji se često sreće u igrama [1]. Na Slici-4 je prikazana postavka problema.



Imamo zemljino tle, koje ćemo označiti sa y = O i tenk koji je smešten u tački:

x = 0, y = 0.

Cev topa je nagnuta prema horizontu pod uglom ϴ. Masa projektila je m a početna brzina Vi (skalarna veličina). Traži se putanja projektila, odnosno pozicija u svakom frejmu. Problem rešavamo tako što najpre razložimo vektor brzine na komponente u pravcu x i y ose.

Imamo:

Vix = Vi* cos ϴ

Viy = Vi* sin ϴ

Sada treba na trenutak zaboraviti kretanje u pravcu ose x, i skoncentrisati se na kretanje u pravcu ose y. Jednačina kretanja

u pravcu ose z ima sledeći oblik:

$$\frac{\mathrm{d}V_x}{\mathrm{d}t} = (-mg)/m = -g$$

Pri padu će projektilu trebati isto vreme (i postići će vertikalnu brzinu V iy u trenutku udara), pa je vreme leta projektila jednako 2*t u .

Kako je u horizontalnom pravcu brzina konstantna (jer smo zanemarili otpor vazduha) to je pozicija projektila na x osi u svakom trenutku t jednaka

X(t) = V ix * t

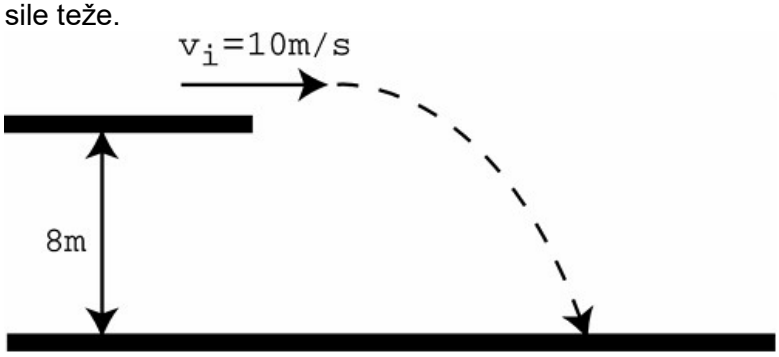
Domet, ili tačka u kojoj će pasti projektil je (stavljajući t = 2*t u) jednaka:

D = V ix *2*t u = V i * cos Θ * 2 * V i * sin Θ / g

= V i2 * (sin 2Θ) / g

U igri cilj ne mora da bude na istoj visini kao i tenk, tj. na y = 0. Sledeći listing prikazuje kako se mogu izvršiti potrebna sračunavanja

```
/// ---------    Inputs
float x_pos     = 0,   // starting point of projectile
y_pos   = SCREEN_BOTTOM, // bottom of screen
y_velocity = 0,   // initial y velocity
x_velocity = 0,   // constant x velocity
gravity = 1,   // do want to fall too fast
velocity   = INITIAL_VEL,  // whatever
angle   = INITIAL_ANGLE; // whatever, must be in radians
// compute velocities in x,y
x_velocity = velocity*cos(angle);
y_velocity = velocity*sin(angle);
// do projectile loop until object hits
// bottom of screen at SCREEN_BOTTOM
while(y_pos < SCREEN_BOTTOM)
{
    // update position
    x_pos+=x_velocity;
    y_pos+=y_velocity;
    // update velocity
    y_velocity+=gravity;
}       // end while
```

Action	Discussion with students about horizontalni hitac, kosi hitac, Programiranju ravnomernog I ubrzanog kretanja, i kretanja pod dejstvom sila. Ustvari cemo pricati o tome kako je moguce realne fizicke pojavi isprogramirati u kodu. **Primer 1: Horizontalni hitac** Prema Slici-4, objekt je bačen sa platforme horizontalnom brzinom i kreće se pod dejstvom sile teže.  Slika 5.1.4 Horizontalni hitac [2] U pravcu xorizontalne ose nema ubrzanja i brzina je konstantna Vx = Vi, dok je u pravcu vertikalne ose ubrzanje konstantno (i negativno, jer se obično osa y usmerava nagore), pa se vertikalna brzina menja s vremenom i jednaka je: Vy = - 9.81 t. Vektor brzine u nekom trenutku vremena t je jednak: U pravcu xorizontalne ose ubrzanje je jednako nuli ax = 0 (i stoga je brzina je konstantna) dok je u pravcu vertikalne ose ubrzanje konstantno ay = -g = -9.81 m/s2 Vektor ubrzanja u nekom trenutku vremena t je jednak: $\mathbf{a} = \begin{bmatrix} \mathbf{0} \\ -g \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ -9.81 \end{bmatrix}$ U slučaju da je platforma visoka 8 metara, a početna brzina 10 m/s, postavlja se pitanje koliko daleko ce otići objekat dok ne udari u zemlju? Rešenje: Polazimo od jednačine za 1D kretanje u pravcu ose y, pri čemu je početna brzina u	

pravcu ose y jednaka 0. Pređeni put u pravcu ose y je:
Dy = Vy t + 1/2 ay t2
–8m = (0m/s)t + 1/2 (–9.8m/s2)t2
Rešavanjem prethodne jednačine dobijamo da je t = 1.28s. Sada imamo dovoljno informacija
da odredimo dužinu puta u pravcu ose x:
Vx = Dx/t => Dx = Vx * t = 10 m/s * 1.28 s
Dx = 12.8m
Stoga, objekat će se udaljiti 12.8 m od ivice platforme.

Primer 2: Programiranje kretanja u 2D
Ono sto nas ovde zanima je šta to znači da isprogramiramo kretanje korišćenjem vektora.

Predjeni put je brzina tela koje se kreće pomnožena sa proteklim vremenom:

```
location.add(velocity * dT);
```

<u>Klasa Mover</u>
Kreiramo klasu Mover pomoću koje opusujemo stvari koje mogu da se
kreću po ekranu
Objekat tipa Mover ima dva podatka: location i velocity, koji su oba tipa PVector.

```
class Mover
{
public:
    PVector location;
    PVector velocity;
```

<u>Ubrzano kretanje</u>
Za interesantnije kretanje, kretanje koje je bliže
realnom svetu koji se dešava oko nas, neophodno je da dodamo još jedan PVector zu našu
klasu —acceleration. U kodu, to bi izgledalo ovako:

```
void update() {
        // Our motion algorithm is now two lines of  code!
        velocity.add(acceleration * dt);
        location.add(velocity * dt);
    }
```

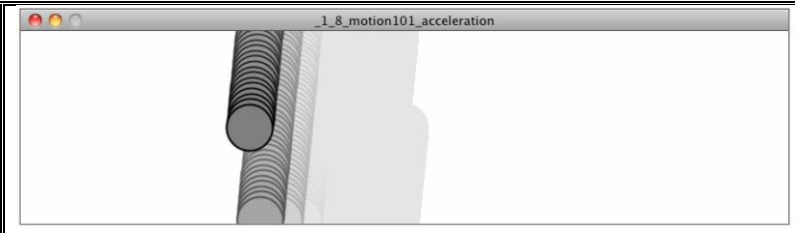
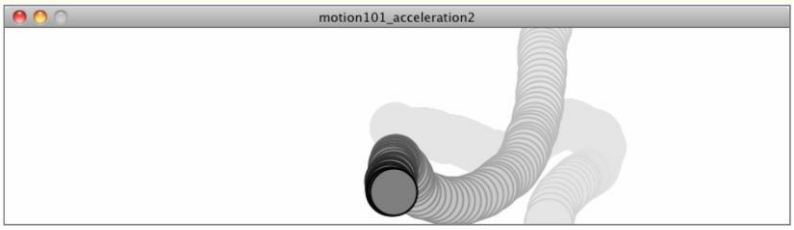


Figure 1. Primer konstantnog ubrzanja.

Potpuno slučajno ubrzanje

U ovom slučaju, umesto inicijalizacije vrednosti ubrzanja u konstruktoru objekta, mi želimo da izaberemo novo ubrzanje u svakom sledećem ciklusu, tj svaki put kada pozovemo funkciju update().

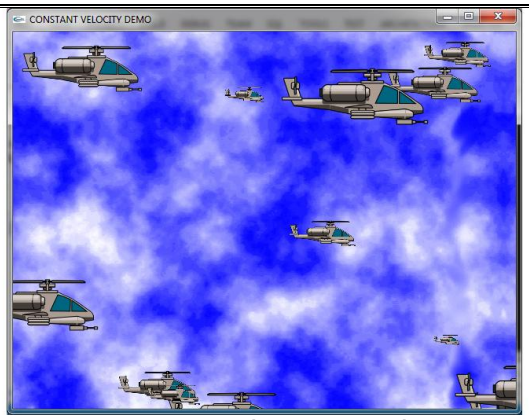
```
// L11 - Primer 3: Kretanje 101 (brzina i proizvoljno ubrzanje)

    void update()
    {
        // The random2D() function will give us a PVector of length 1
                // pointing in a random direction.
        acceleration.random2D();
        velocity.add(acceleration * dt);
        velocity.limit(topspeed);
        location.add(velocity * dt);
    }
```



Demo: Chopper:

U okviru sekcije Deljivi materijali se nalazi L11-ChopDemo.zip koji pokazuje flotu helikoptera
koji se kreću konstantnom brzinom, i koji je realizovan u GLUT / OpenGL / C++ okruzenju, Slika-7.
Projekat kao osnovu koristi projekat RoboRacer2D, koristi SOIL biblioteku za učitavanje slike, i FMOD za generisanje zvuka. Promenljive u igri su:

```
//Game variables
Sprite* chopers[NUM_CHOPPERS];
Sprite* background[NUM_BACKGR];
```



Pozadina se pomera s desno na levo. Da bi slika bila kontinualna uvodimo dve pozadine (kada je jedna pozadina delimično van ekrana, tu crninu prekriva druga pozadina koja se takođe kreće s leva na desno). Kada jedna pozadina izađe potpuno sa ekrana pojavljuje se na drugoj strani ekrana. Slično važi i za helihoptere. Oni se kreću sa desna na levo, kada pređu desnu granicu ekrana pojavjuju se na leveoj strani ekrana.

Demo: Cannon (10 min)

Kod opisan na predavanjima omogućama generisanje tenka, i usmeravanja njegovog topa, a zatim i ispaljivanje projektila. Kompletan projekat je u fajlu L12-Cannon.zip.
Lista projekatila se cuva u nizu:

```
PROJECTILE missiles[NUM_PROJECTILES];
```

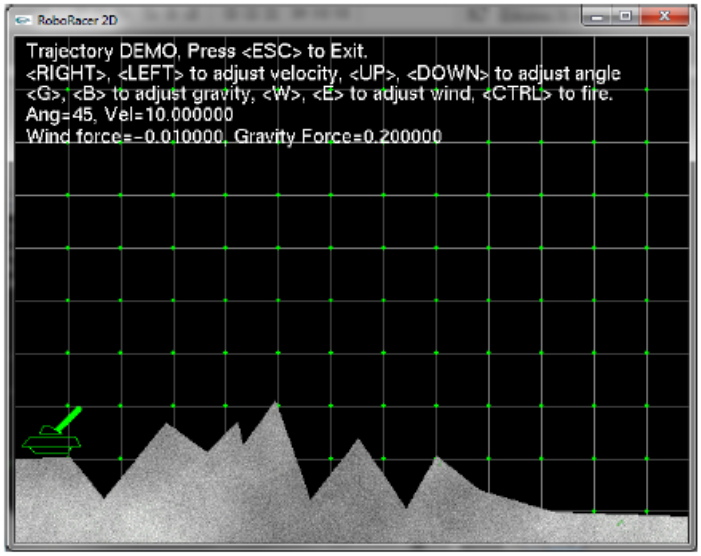


Figure5 Izgled ekrana igre iz primera [2]

U ekranu igre su prikazane kontrole koje omogucavaju promenu ugla, topa, povecanje i smanjenje sile gravitacije i otpora vazduha, kao i ispaljivanje projektila.

Programiranje sila u 2D

Vratimo se na našu klasu Mover i koja ima podatke location, velocity, i acceleration. Naš naredni cilj je da primenimo silu na posmatrani objekat, u obliku:

```
mover.applyForce(wind);
```

Ili

```
mover.applyForce(gravity);
```

gde su vetar (wind) i gravitacija (gravity) podaci tipa PVector. Vršimo slaganje sila, tako da funkcija applyForce()

```
void applyForce(Pvector force) {
        // Newton's second law (with force accumulation
        force.div(mass);
        acceleration.add(force);
    }
```

pri čemu je sa mass označena masa objekta.

Materials / equipment / digital tools / software	*The materials for learning* are given as a part of references of the end from this topic plan; *Equipment*: classroom, board, chalk; *Digital tools*: laptop, projector;	
Consolidation	• The teacher's discussion with the students through appropriate questions; • Independent solving of simple tasks by the students under the supervision of the teacher; • Given of examples by the teacher for introducing a new concept in a cooperation and a discussion with the students; • Assignment of homework by the teacher with a time limit until the next class.	

Reflections and next steps	
Activities that worked	**Parts to be revisited**
After the class, the teacher according to his personal perceptions regarding the success of the class fills in this part.	Through the success of the homework done by the students, questions and discussion at the beginning of the next class, the teacher comes to the conclusion which parts of this class should be revised.

References

1. Miljan Milosevic, Elektronski materijali predavanja za učenje iz predmeta CS232 Programiranje 2D igara, Metropolitan Univerzitet, 2021. godina, Beograd
2. Daniel Shiffman, The Nature of Code: Simulating Natural Systems with Processing , 2014.
3. AndrIrish Lamothe, Tricks of the Windows Game Programming Gurus, Sams Poublishing, 2002.