# Tema 1 - Duck Hunt

- **Responsabili:** Anca Băluțoiu, Alex Grădinaru, Chris Brandon
- **Lansare:** 31 octombrie 2022
- **Termen de predare:** 13 noiembrie 2022, ora 23:59
- **Regulament:** Regulament General
- **Notă: Orice informație ce nu a fost acoperită în acest document este la latitudinea voastră!**

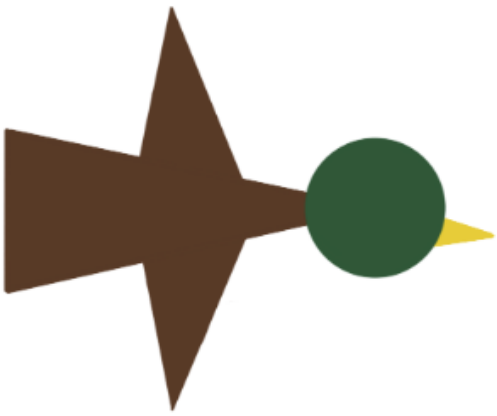
În cadrul temei 1, veți avea de implementat un joc de tipul Duck Hunt. Pentru inspirație, puteți testa jocul original aici: https://www.duck-hunt.org/ [https://www.duck-hunt.org/]. :)

De asemenea, puteti viziona un mic demo construit rapid pe baza frameworkului de laborator care acopera cerintele, avand un aspect vizual minimal :).


## Rațe

### Construcție

Rațele sunt reprezentate de câteva primitive geometrice 2D (minimum 4: corp, aripi și cap), poziționate sugestiv. Puteți vedea o construcție propusă a răței în imaginea de mai jos, formată din 4 triunghiuri și un cerc (câte un triunghi pentru corp, aripi și cioc, și capul reprezentat de cerc).



### Animare

Pentru a construi rața va fi așadar nevoie de un ansamblu de mai multe obiecte definite manual (vedeți modalitatea de definirea a pătratului din laborator) care se vor mișca împreuna, dar vor avea și animații independente. Astfel, rața:

- Va avea o animație de zbor: va da din aripi - puteți aplica de exemplu o rotație simplă pe fiecare aripă
- Va zbura în diverse direcții în mod unitar - tot asamblul obiectelor care compun rața se va mișca sau roti împreună

Pentru a obține aceste efecte, recomandăm folosirea de transformări ierarhice: va puteți construi o matrice de transformare a întregii rațe pe care să o aplicați fiecărei primitive în parte, după ce se aplică eventuale transformări individuale ale acestora.

Exemplu de funcționare a transformărilor ierarhice folosind OpenGL și glm (daca porniți de la laboratorul 3, se poate realiza similar folosind funcțiile definite in Transform2D):

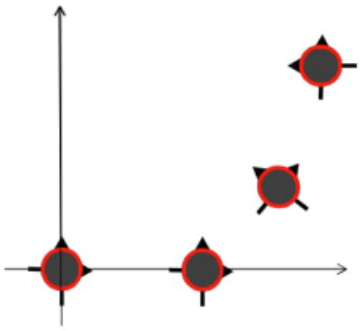
1. Definim o structură ierarhică de transformări (relative la o matrice de modelare părinte)

```
glm::mat4 model = glm::mat4(1.0f);
drawPog(model);  // model = I

model = glm::translate(model,glm::vec3(3,0,0))
drawPog(model);  // model = T1

model = glm::translate(model,glm::vec3(1,1,0))
model = glm::rotate(model,45,glm::vec3(0,0,1))
drawPog(model);  // model = T1.T2.R1

model = glm::translate(model,glm::vec3(2,0,0))
model = glm::rotate(model,45,glm::vec3(0,0,1))
drawPog(model);  // model = T1.T2.R1.T3.R2
```



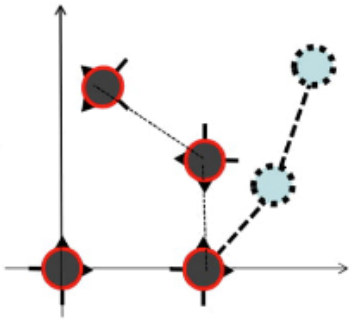
2. Modificând o transformare pe lanțul ierarhic, aceasta se va propaga tuturor copiilor

```
glm::mat4 model = glm::mat4(1.0f);
drawPog(model);  // model = I

model = glm::translate(model,glm::vec3(3,0,0))
drawPog(model);  // model = T1

model = glm::translate(model,glm::vec3(1,1,0))
model = glm::rotate(model,90,glm::vec3(0,0,1))
drawPog(model);  // model = T1.T2.R1

model = glm::translate(model,glm::vec3(2,0,0))
model = glm::rotate(model,45,glm::vec3(0,0,1))
drawPog(model);  // model = T1.T2.R1.T3.R2
```

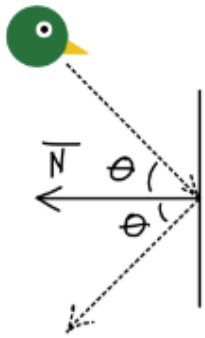


## Afișare

Ratele apar cate una pe rand pe ecran, astfel incat la un moment dat sa nu fie decat o rata prezenta in scena. In momentul in care o rata este impuscata sau evadeaza se va putea afisa urmatoarea.

## Mișcare

Rațele apar în partea de jos a ecranului și își vor începe zborul pe o direcție aleatoare în plan. Continuând pe acea direcție, la un moment dat vor ajunge la marginea ecranului. În această situație, rața trebuie să se “reflecte” și să își continue drumul în direcția aferentă, asemenea unei bile de biliard care lovește marginea mesei. Dacă rața este împușcată, aceasta va cădea pe sol mișcându-se printr-o animație vertical în jos, respectiv dacă evadează va zbura în afara ecranului mișcându-se printr-o animație vertical in sus.



După un număr rezonabil de secunde, în care rața s-a ciocnit de câteva ori de marginea ecranului, aceasta va evada. Numărul de secunde va depinde de viteza cu care se mișcă rața în implementarea voastră, dar propunem să fie în jur de 3-5 secunde pentru a nu sta foarte mult o singură rață pe ecran.

Hint! Pentru a gestiona timpul, cea mai simplă variantă ar fi să țineți cont de numărul total de secunde care au trecut de la apariția ultimei rațe. Din moment ce va exista o singură rață pe ecran la orice moment de timp, această informație poate lua forma unei variabile în care se aduna deltaTimeSeconds la fiecare apel al funcției Update. În momentul în care numărul de secunde active ale rației este depășit în această variabilă, rața va tranziționa către starea de “evadat”. La apariția unei rațe noi, temporizarea reîncepe de la 0 secunde și se poate continua astfel pentru întreaga durata a jocului. De asemenea, puteți folosi și metode de măsurare a timpului prin funcții C++.

Atenție! Dacă unghiul de plecare al rației este perfect vertical sau orizontal, rața se va mișca doar pe axa respectivă. Ar fi bine să se evite unghiuri prea mici față de axele OX și OY pentru a garanta o traiectorie rezonabilă a rațelor.

Așadar, rațele au 3 moduri de mișcare:

1. Activ: rața a apărut pe ecran și se mișcă pe o direcție, aceasta schimbându-se la momentul ciocnirii cu marginea ecranului. Din această stare, rata va tranziționa ori către starea de împușcat, ori către starea de evadat.
2. Împușcat: rața a fost împușcată și se va mișca vertical în jos, până ajunge la marginea de jos a ecranului, moment în care dispare
3. Evadat: rața a evadat și se va mișca vertical in sus, până ajunge la marginea de sus a ecranului, moment în care dispare

Observatie: capul ratei trebuie sa fie indreptat aproximativ spre directia de miscare :) Nu sunt permise rate care merg/zboara cu spatele.

## Vieți

Jucătorul va porni inițial cu 3 vieți. În momentul în care o rață scapă (jucătorul a ratat să nimerească rața cu toate cele 3 gloanțe), acesta pierde o viață. Numărul de vieți rămase vor fi desenate pe ecran, în colțul stânga sus, de exemplu sub forma unor cercuri roșii.

## Gloanțe

La orice moment de timp, jucătorul trebuie să știe câte gloanțe mai are la dispoziție. Pentru asta, în colțul stânga sus (sub numărul de vieți) veți afișa și numărul de gloanțe disponibile în mod similar cu numărul de vieți, de exemplu sub forma unor dreptunghiuri verzi.

## Scor

Pentru fiecare rață împușcată, scorul jucătorului va crește. Acesta va fi și el reprezentat în interfața grafică în colțul dreapta-sus cu ajutorul a două dreptunghiuri:

- Un dreptunghi wireframe, care reprezintă nivelul maxim de scor (puteți considera un nivel maxim de scor de 50 sau orice altă valoare >10)
- Un dreptunghi solid, albastru, în interiorul celui wireframe, care reprezintă scorul curent

# Gameplay

Scopul jocului este ca jucătorul să împuște cât mai multe rațe înainte să rămână fără vieți. Jocul începe cu un număr de vieți disponibile, dintre care se pierde câte una de fiecare dată când o rața reușește să scape fără să fie împușcată.

Rațele apar câte una singură pe ecran, pe rând, fiecare după dispariția (prin evadarea sau împușcarea) celei anterioare. O rață va cădea pe sol când este împușcată și va zbura vertical în sus pentru a simboliza evadarea (sunt descrise deja detalii în secțiunile “Rațe - Afișare” și “Rațe - Mișcare”). Ele se mișcă după regulile descrise la secțiunea “Rațe - Mișcare”. După generarea a câte 5 rațe, viteza de mișcare a acestora va crește și, ca atare, dificultatea jocului va crește.

De exemplu: dacă primele 5 rațe se mișcă la o viteza v0, următoarele 5 ar putea avea viteza v1 = v0 + v0 / 5, următoarele v2 = v0 + 2 * v0 / 5 și așa mai departe, pentru a avea o modificare incrementală a dificultății.

## Trasul cu pușca

Pentru a împușca o rață, jucătorul are la dispoziție 3 gloanțe care se reîncarcă de fiecare dată când apare o nouă rață. De fiecare dată când acesta face clic pe ecran, se va considera ca un glonț a fost tras în acel punct pe ecran și numărul de gloanțe disponibile va scădea cu 1. Dacă punctul de pe ecran (detalii secțiunea “Intersecția glonțului cu rața”) care a fost împușcat intersectează rața, ea va fi considerată împușcată, scorul (detalii secțiunea “Scor”) va crește și rața va cădea pe sol (detalii secțiunea “Rațe - Mișcare”).

## Intersecția glonțului cu rața

Având în vedere că rațele au o formă destul de complexă, ar fi greu de calculat exact ce puncte de pe ecran se intersectează cu ele. Așadar, le puteți aproxima sub forma unui dreptunghi în care este încadrată rața (similar figurii de mai jos). Pentru a calcula coordonatele colțurilor acestui pătrat, puteți folosi câteva noțiuni de geometrie în plan, aplicate în funcție de coordonatele la care se află punctele de pe silueta raței.

Pătratul încadrator este mult mai ușor de calculat dacă toate coordonatele locale ale primitivelor din care este construită rața sunt calculate față de un punct cât mai apropiat de centrul raței (detalii secțiunea “Rața - Construcție”).

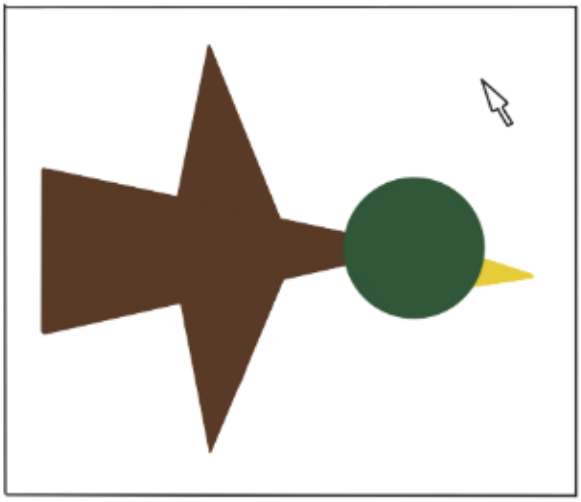
Mai multe detalii despre aproximarea coliziunilor 2D: https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection [https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection] .

Pentru a afla dacă mouse-ul se află în interiorul dreptunghiului încadrator al rației, trebuie făcută o conversie din coordonate de vizualizare către coordonatele logice în care rața se plimbă pe ecran. Dacă va folosiți de scheletul laboratorului 3, aceste 2 spații de coordonate corespund 1:1 întrucât spațiul de desenare este limitat la rezoluția porții de afișare și se poate considera că poziția cursorului mouse-ului este și poziția sa în coordonate logice. Altfel, va trebui aplicată o transformare similară cu transformarea fereastră - poartă (detaliată tot în laboratorul 3), doar ca inversă. Atenție la corecția coordonatei Y.

După ce cursorul și dreptunghiul se află în același spațiu de coordonate, verificarea intersecției se rezumă la verificarea dacă un punct în plan se află într-un dreptunghi aliniat cu axele Ox și Oy. Detaliile acestui calcul sunt lăsate ca exercițiu pentru student.

Inputul de la mouse se poate trata în funcția “OnMouseMove” din framework. Aceasta are 4 parametri: mouseX, mouseY, deltaX, deltaY. Primele 2 se referă la poziția la care se află cursorul în momentul în care se apelează funcția, în pixeli. Numerotarea începe din colțul stânga-sus al ferestrei de vizualizare în (0, 0). Cei 2 parametri din urmă se referă la deplasarea exactă (tot în pixeli) a cursorului de la poziția sa în frame-ul anterior pana la poziția sa în frame-ul în care a fost apelată funcția. De exemplu, daca mouse-ul s-a mișcat de la poziția (1200, 300) la poziția (1220, 294) în intervalul de la ultimul frame până la cel curent, vom avea următoarele valori: mouseX = 1220, mouseY = 294, deltaX = 20, deltaY = 294.

Atenție! Aceste valori sunt întregi, aveți grijă la tipurile de date dacă intenționați să le împărțiți. De asemenea, deltaX și deltaY deja sunt calculate față de frame-ul anterior. Ca atare, nu mai este nevoie să ne legăm de deltaTimeSeconds pentru a avea o mișcare independentă de frame rate.



### Evadarea raței

În momentul în care toate cele 3 gloanțe au fost consumate fără a fi fost împușcată rața, aceasta evadează. De asemenea, dacă rața nu a fost împușcată timp de un număr de secunde, va evada. În ambele situații în care rața evadează, jucătorul va pierde o viață (detalii secțiunea “Vieți”) și nu va primi scorul aferent raței respective.

## Funcționalități obligatorii (150 puncte)

- Mișcare/animații rață (75p total)
  - Desenare/asamblare figură geometrică rață 15p
  - Deplasare activă (deplasare ansamblu rață și animații aripi) 20p
  - Poziționare și direcție inițială 10p
  - Reflexii 10p
  - Împușcat 10p
  - Evadare 10p
- Gameplay (75p total)
  - Apariție rațe, câte una pe ecran 5p
  - Incrementare viteză rațe 5p
  - Temporizare evadare 5p
  - Trasul cu pușca
    - Țintire 20p
    - Coliziuni rață 20p
  - Interfața grafică (funcționalitate și afișare)
    - Vieți 5p
    - Gloanțe 5p
    - Scor 10p

## Exemple de funcționalități bonus

Orice funționalitate suplimentară implementată (care nu este inclusă în cerințele obligatorii) poate fi considerată ca punctaj bonus dacă este suficient de complexă. Funcționlitățile bonus se iau în considerare doar dacă funcționalitățile obligatorii au fost realizate.

- Scor variabil în funcție de viteza rației și multiplier aplicat pentru număr de rațe împușcate consecutiv (killing spree). Este necesară afișarea unui element animat (de ex o formă simplă de romb care se rotește) în momentul atingerii acestui prag.
- După ce apar 5 rațe, se trece la următorul nivel: se afișează pe ecran un text care îi comunică jucătorului această schimbare și următoarele 5 rațe (până la schimbarea din nou a nivelului) se vor mișca mai repede.

- Boss fight, care apare de exemplu o dată la 10 rațe - o rața mai mare care se mișcă mai greu și trebuie împușcată de mai multe ori.
- Fiecare rață are o culoare aleatorie (similar cu jocul original) și își poate schimba direcția brusc/random, chiar dacă nu atinge marginea ecranului (poate face de exemplu o mișcare zig-zag cu temporizare aleatoare).
- Realizarea gazonului de jos în spatele căruia apare rața, astfel încât să nu apară brusc pe ecran de nicăieri.
- Rațe mai detaliate/complexe ca geometrie.
- Crosshair pe mouse și o pușcă/armă animată în funcție de poziția cursorului pe ecran (să fie îndreptată/rotită spre cursor).
- Sistem de tragere animat (gloanțe vizibile, o mică animație în momentul în care se atinge rața - de exemplu sar niște ‘pene’ aproximate în direcții aleatorii).
- Rațe “speciale” care dau jucătorului anumite abilități (o viață extra, un glonț extra, toate rațele se vor mișca foarte încet pentru câteva secunde). Fiecare astfel de rață va avea o culoare specifică power up-ului pe care îl acordă si va exista un indicator de timp care arata cat timp mai dureaza acea abilitate (de ex o bara similara cu cea de scor, care scade in timp si dispare cand nu mai e valabila abilitatea).
- Animație complexă când moare rata.
- Posibilitatea de a avea mai multe rațe pe ecran în același timp și sistem de gestiune a gloanțelor corespunzător/îmbunătățit pentru mai multe rațe.

## Întrebări și răspunsuri

Pentru întrebări vom folosi forumurile de pe moodle. Orice nu este menționat în temă este la latitudinea fiecărui student!

## Notare

Baremul este orientativ. Fiecare asistent are o anumită libertate în evaluarea temelor (de exemplu, să dea punctaj parțial pentru implementarea incompletă a unei funcționalități sau să scadă pentru hard coding). Același lucru este valabil atât pentru funcționalitățile obligatorii, cât și pentru bonusuri.

Tema trebuie încărcată pe moodle. Pentru a fi punctată, tema trebuie prezentată la laborator. Vor exista laboratoare speciale de prezentare a temelor (care vor fi anunțate).

## Indicații suplimentare

Tema va fi implementată în OpenGL și C++. Este indicat să folosiți framework-ul și Visual Studio.

Pentru implementarea temei, în folderul **src/lab_m1** puteți crea un nou folder, de exemplu **Tema1**, cu fișierele **Tema1.cpp** și **Tema1.h** (pentru implementare POO, este indicat să aveți și alte fișiere). Pentru a vedea fișierele nou create în Visual Studio în Solution Explorer, apăsați click dreapta pe filtrul **lab_m1** și selectați **Add→New Filter**. După ce creați un nou filtru, de exemplu **Tema1**, dați click dreapta și selectați **Add→Existing Item**. Astfel adăugați toate fișierele din folderul nou creat. În fișierul **lab_list.h** trebuie adăugată și calea către header-ul temei. De exemplu: **#include “lab_m1/Tema1/Tema1.h”**

## Arhivarea Proiectului

- În mod normal arhiva trebuie să conțină toate resursele necesare compilării și rulării
- Înainte de a face arhiva asigurați-vă că ați curățat proiectul Visual Studio:
  - Click dreapta pe proiect în **Solution Explorer → Clean Solution**
  - Ștergeți folderul **/build/.vs** (dacă nu îl vedeți, **este posibil să fie ascuns**)
- SAU ștergeți complet folderul **/build**
- În cazul în care arhiva tot depășește limita de 50MB (nu ar trebui), puteți să ștergeți și folderul **/deps** sau **/assets** întrucât se pot adăuga la testare. Nu este recomandat să faceți acest lucru întrucât îngreunează mult testarea în cazul în care versiunea curentă a bibliotecilor/resurselor diferă de versiunea utilizată la momentul scrierii temei.