# Tema 2 - Skyroads

- **Responsabili:** Anca Bălușoiu, Alex Grădinaru, Florin Iancu, Philip Dumitru
- **Lansare:** 22 noiembrie
- **Termen de predare:** 13 decembrie 2020, ora 23:55
- **Regulament:** Regulament General
- **Notă: Orice informație ce nu a fost acoperită în acest document este la latitudinea voastră!**
- **Mentiune depunctari vacanta**: In perioada 19 decembrie - 10 ianuarie **NU** se vor calcula depunctari pentru teme!

În cadrul temei 2 trebuie să implementați o versiune modificată a jocului Skyroads. Un walkthrough al jocului original îl puteți găsi aici:


## Gameplay

Spre deosebire de jocul original, care este structurat pe mai multe niveluri finite, jocul pe care îl veți implementa va fi un endless runner. Harta se va genera aleator pe minim trei coloane (benzi) de mers astfel încât să nu devină imposibil pentru jucător să treacă peste un anumit spațiu, iar jucătorul se va deplasa doar înainte, având posibilitatea să își schimbe viteza de deplasare și coloana de mers.

În timpul jocului, jucătorul va trebui să țină constant cont de cantitatea de combustibil pe care o mai are. Cantitatea de combustibil va scădea treptat odată cu deplasarea. În momentul în care rămâne fără combustibil, jocul se va termina. Jocul se poate termina, de asemenea, și dacă jucătorul cade de pe o platformă (în cazul acesta va fi necesară o animație prin care jucătorul cade din scenă - de exemplu, personajul va cădea o distanță prestabilită și apoi va dispărea din scenă. Orice altă idee de animație care să simbolizeze sfârșitul jocului și dispariția jucătorului este binevenită).

Vor exista de asemenea platforme care vor aduce beneficii sau dezavantaje jucătorului dacă aterizează pe ele. Un astfel de exemplu este platforma de recăpătare a combustibilului pierdut în timpul jocului.

Puteți alege dacă să folosiți obiecte 3D deja existente sau dacă să le generați voi. 😃

## Construcția Scenei și a Jucătorului

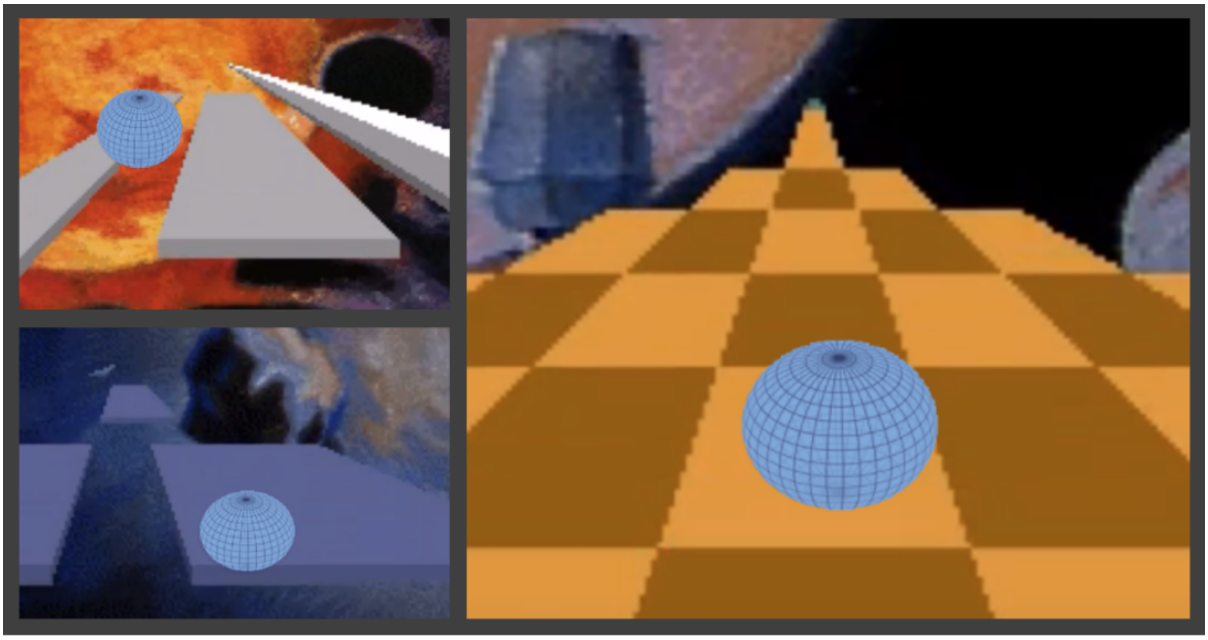
- **Harta**

Se va genera aleator prin plasarea în scenă a mai multor platforme poziționate pe mai multe coloane (minim 3, dar puteți să vă definiți și mai multe). Aceste platforme pot fi reprezentate prin cuburi de culori și dimensiuni diferite, scalate astfel încât să fie subțiri pe y, pe x să aibă dimensiunea unei

coloane, iar pe z să fie o valoare aleatoare. Cuburile nou generate vor fi mereu generate/afișate inițial în scenă la aceeași distanță de jucător pe axa OZ (este la latitudinea voastră cât setați această distanță), iar în timp cuburile care vor ieși din scenă în spatele jucătorului nu vor mai fi redate.

Puteți genera platforme în același timp pentru oricâte coloane. Este la latitudinea voastră câte cuburi redați la un moment dat în scenă. Singura condiție este să nu creați situații când distanța dintre oricare două platforme este prea mare și jucătorul nu poate sări pe o altă platformă.

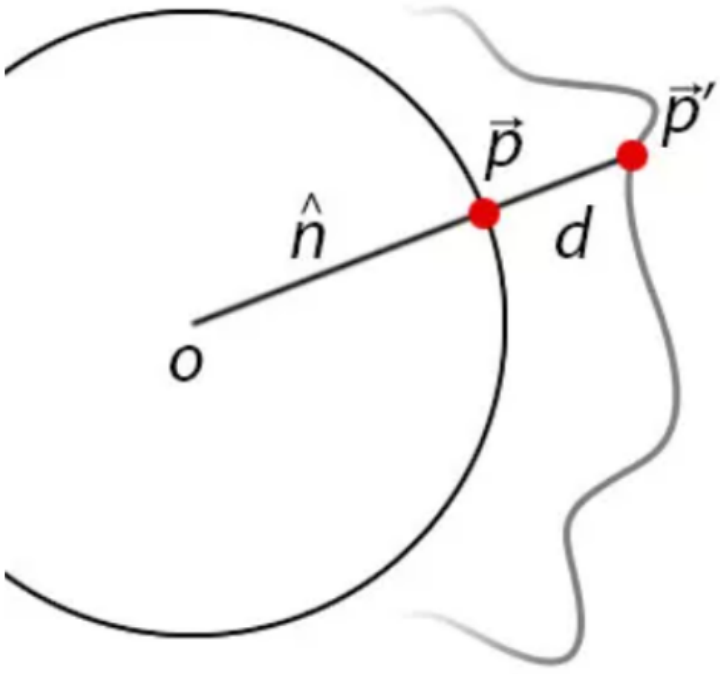
În imaginea de mai jos, puteți vedea câteva exemple de generare de scene. Se poate observa că sunt 2 modalități de a vă defini coloanele: fie le gândiți încă de la început cu o anumită distanță între ele, fie le gândiți ca fiind lipite. Astfel, în prima situație veți avea în permanență spații între platformele laterale, iar în cel de-al doilea caz puteți avea și platforme lipite între ele. Între două platforme de pe aceeași coloană veți avea un spațiu de o dimensiune aleatoare.



Atenție! Imaginea de fundal din exemplele date nu trebuie implementată! 😃

- **Jucătorul**

Acesta va fi reprezentat de o sferă. În momentul în care sfera aterizează pe o platformă specială, care acordă power-ups, aceasta va fi deformată în Vertex Shader. De exemplu puteți să aplicați o funcție de zgomot pe direcția normalelor vârfurilor (exemplu în imaginea de mai jos). Animația de deformare va înceta în momentul în care expiră abilitatea luată de jucător.



Legendă:

- n = normala la suprafață a vârfului
- p = poziția inițială a vârfului
- p’ = poziția finală a vârfului
- d = distanța de deformare (diferită pentru fiecare vârf)

Exemple de deformări:

În momentul implementării, puteți alege să deplasați jucătorul pe axa OZ înspre cuburi, iar cuburile stau pe loc sau deplasați cuburile înspre jucător, iar jucătorul stă pe loc. În funcție de această decizie, trebuie să aveți grijă la poziția inițială a cuburilor în scenă (fie se vor reda inițial la aceeași coordonată z, fie coordonata z va fi diferită în funcție de poziția jucătorului).

## Controlul Jucătorului

Jucătorul va fi controlat prin tastele WASD și SPACE. Folosind A și D, va schimba coloanele de mers, tasta SPACE va fi folosită pentru a sări peste spații, iar W și S pentru a alege viteza de mers. Viteza maximă pe care o poate atinge jucătorul o puteți alege voi. Folosind tasta C, se va oscila între modurile de joc first și third person camera.

## Interfața cu Utilizatorul

Jucătorul trebuie să țină în permanență cont de cantitatea de combustibil pe care o mai are. De aceea este important să poată vedea acest aspect pe ecran. Pentru acest lucru se vor folosi două dreptunghiuri suprapuse afișate într-un colț al ecranului (unul alb/negru în spate și unul colorat în față, pe care îl veți scala pentru a evidenția consumarea combustibilului).

Atenție! Fiind vorba de un element de interfață grafică (deci care nu ține cont de restul lumii), nu se aplică transformările MVP aplicate pe întreaga lume.

## Platforme cu Avantaje/Dezavantaje

În cadrul jocului vor exista două tipuri de platforme: simple și cu efecte. Platformele simple vor fi de culoare albastră. Platformele care vor acorda jucătorului abilități sau dezavantaje vor fi colorate astfel:

- Roșu: jocul se termină instant
- Galben: jucătorul pierde o parte din combustibil
- Portocaliu: jucătorul este blocat un anumit număr de secunde la o viteză foarte mare (tastele W și S nu au niciun efect în această perioadă)
- Verde: jucătorul recuperează o parte din combustibil

Orice platformă pe care jucătorul aterizează își va schimba culoarea în mov.

Mai multe informații despre coliziunile în 3D găsiți aici:

- https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection [https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection]

## Bonusuri Posibile

- Adăugarea unei vieți. Jucătorul poate avea mai multe vieți. Astfel, va avea șansa să continue jocul când a rămas fără combustibil. Va fi nevoie de reprezentarea vieții în interfața cu utilizatorul (asemănător combustibilului) și de platforme de pierdere și recăpătare a vieții.
- Obstacole: În scenă pot apărea și obstacole. Acestea vor fi realizate tot din cuburi redimensionate. Pot exista obstacole vizibile de la început în scenă sau pot exista obstacole care devin vizibile când jucătorul se apropie de ele (capcane). În acest caz, se va crea o animație de scalare a cubului astfel încât să pară că iese din platformă.
- În cazul coliziunii cu un obstacol, sfera va fi animată corespunzător (animație care să simuleze dezumflarea ei sau spargerea în mai multe bucăți)
- Implementarea unei platforme speciale care îi acordă jucătorului invincibilitate pentru o perioadă de timp (poate trece prin obstacole, dar este în continuare nevoit să sară peste spații)
- Animare realistă a sferei atunci când aterizează și sare pe o nouă platformă (de exemplu, puteți considera sfera o minge de cauciuc)
- Număr variabil de coloane (de exemplu, la un moment dat în cadrul aceleași rulări sunt 3 coloane existente, apoi sunt 5, pe urmă sunt iar 3, apoi 4 etc.)
- Creșterea dificultății jocului pe măsură ce trece timpul: viteza minimă mai mare, combustibilul se termină mai repede, mai puține platforme, reducerea numărului de platforme care aduc avantaje și creșterea numărului de platforme care aduc dezavantaje

## Notare (200p)

- Platformele (60p)
  - Generare (20p)
  - Schimbarea culorii după ce a aterizat personajul (10p)
  - Poziționarea inițială corectă (30p)
- Personajul (60p)
  - Deplasare (10p)
  - Animare în shader (20p)
  - First person camera/Third person camera (20p)
  - Animarea personajului atunci când cade de pe platforme (10p)
- Coliziuni (20p)
- Interfața cu utilizatorul (10p)
- Combustibil (20p)
- Power-ups (30p)

- Logică (20p)
- Platforme generate corespunzător (10p)

## Indicații Suplimentare

Tema va fi implementată în OpenGL și C++. Este indicat să folosiți framework-ul și Visual Studio. Pentru implementarea temei, în folderul Source/Laboratoare/ puteți crea un nou folder, de exemplu Tema2, cu fișierele Tema2.cpp și Tema2.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 Laboratoare și selectați Add→New Filter. După ce creați un nou filtru, de exemplu Tema2, dați click dreapta și selectați Add→Existing Item. Astfel adăugați toate fișierele din folderul nou creat. În fișierul LabList.h trebuie adăugată și calea către header-ul temei. De exemplu: #include <Laboratoare/Tema2/Tema2.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 dat clean la proiect
  - Click dreapta pe proiect în **Solution Explorer** → **Clean Solution**, sau
  - Ștergeți folderul **/Visual Studio/obj**
- Ștergeți fișierul **/Visual Studio/Framework_EGC.sdf** (în caz că există)
- Ștergeți fișierul **/Visual Studio/Framework_EGC.VC.db** (în caz că există)
- Ștergeți folderul **/.vs** (în caz că există)
- Ștergeți folderul **/x64** sau **/x86** (în caz că există)
  - Executabilul final este generat în folderul **/x86** sau **/x64** la finalul link-editării în funcție de arhitectura aleasă la compilare (32/64 biți)
- În cazul în care arhiva tot depășește limita de 20MB (nu ar trebui), puteți să ștergeți și folderul **/libs** sau **/Resources** 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.

egc/teme/2020/02.txt · Last modified: 2020/12/15 14:48 by anca.morar