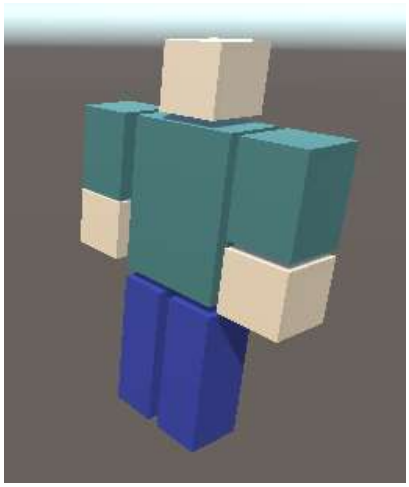
Tema 2 - Survival Maze

În cadrul Temei 2, veți avea de implementat un joc labirint, pe care jucătorul va trebui să îl rezolve. Pentru a îngreuna traseul jucătorului, în labirint se vor găsi inamici.

Jucatorul

Jucatorul va putea controla in varianta Third Person un personaj realizat din cuburi colorate, sub forma unui avatar uman simplificat.



Deplasare

Jucatorul se poate deplasa in scena folosind tastele directionale sau combinatia traditionala WASD. Tot ansamblul avatarului trebuie sa se miste unitar. Animarea, sub orice fel, a personajului este optionala (vedeti bonus).

Atac

In mod implicit, personajul va avea atasata o camera Third Person, dar acesta va avea posibilitatea de atac prin lansarea unor proiectile. Intrarea in modul de atac se poate face pe tasta CTRL sau mouse-dreapta, mod in care camera va fi trecuta in First Person si va putea tinti si lansa proiectile pe directia camerei prin apasarea butonului mouse-stanga sau tasta SPACE.

Proiectilele pot fi obiecte simple (de ex sfere) care au o directie, o viteza si o durata de viata. In momentul in care acestea intalnesc un obstacol (inamic, perete) sau depasesc durata de viata vor disparea din scena.

Labirint

Pentru a implementa labirintul, puteți să îl gândiți sub forma unui grid de dimensiune oarecare. Un perete poate fi reprezentat de un cub. Pentru a genera labirintul, puteți să vă folosiți de o matrice de dimensiunea gridului. Această matrice poate avea 3 valori diferite:

- 0 - va reprezenta un drum liber (nu vom plasa un cub la această poziție)
- 1 - va reprezenta un perete din labirint (vom plasa un cub)
- 2 - un inamic

Jucătorul va fi poziționat inițial la o poziție aleatoare validă (pe o celulă ce reprezintă un drum liber) si se poate deplasa doar pe drumuri libere, in rest fiind blocat de coliziuni.

Update clarificare: Labirintul trebuie sa fie diferit intre doua rulari.

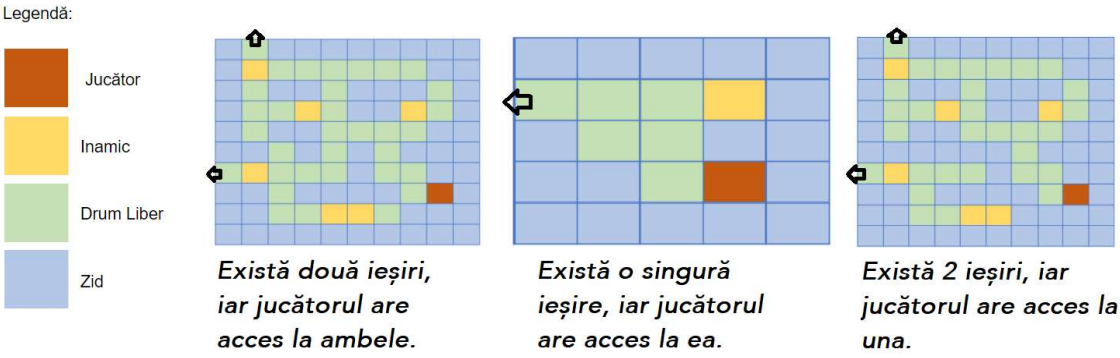
Coliziunile nu trebuie sa fie perfecte, se pot aproxima prin dreptunghiuri sau sfere. Mai multe informatii despre coliziuni si cum se pot implementa in 3D:

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

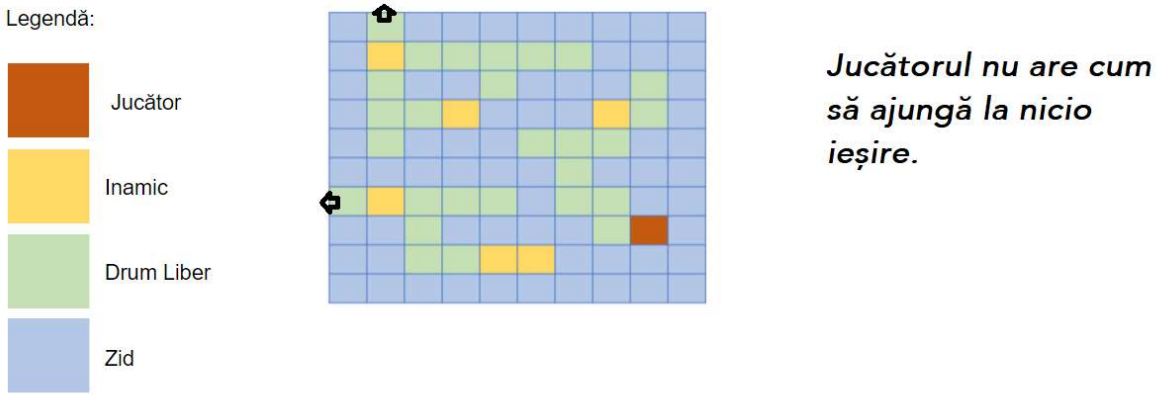
Labirintul puteți să îl generați cu ce dimensiuni doriți și în ce mod doriți cu condiția să existe cel puțin o ieșire din labirint conectată la calea pe care se află jucătorul.

Aici găsiți o serie de algoritmi pentru generarea labirintului: MAZE_GEN_ALGOS [https://github.com/john-science/mazelib/blob/master/docs/MAZE_GEN_ALGOS.md]

Exemplu de configurații valide



Exemplu de configurație invalidă



Inamici

In labirint jucatorul se poate intalni cu inamici. Acestia patruleaza coridorul (se misca in mod regulat in celula lor galbena din grid). Inamicii nu trebuie sa urmareasca jucatorul, dar pozitionarea lor ii poate bloca trecerea. Forma si culoarea inamicilor trebuie sa difere de forma si culoarea jucatorului. Este suficient ca inamicii sa fie reprezentati de forme simple (un singur mesh).

Mișcare inamici

Miscarea inamicilor are loc doar in cadrul patratului galben asociat lor, conform gridului de mai sus. Aveti libertatea sa alegeti orice tip de miscare, cu conditia ca aceasta sa acopere o suprafata cat mai mare din celula. De exemplu, inamicul poate patrula de-a lungul marginilor propriului patrat (ramanand in interiorul celulei). Ca un alt exemplu, inamicul se poate deplasa pe o traiectorie in forma de "8".

Coliziune cu jucătorul

Pentru a ajunge la iesirea din labirint, jucatorul poate fi obligat sa treaca prin celule ce contin inamici. Totusi, daca jucatorul este atins de un inamic, el pierde viata. Atunci cand pierde toata viata disponibila, se poate afisa "Game Over!" la consola si jocul se va opri imediat.

Coliziune cu proiectilul

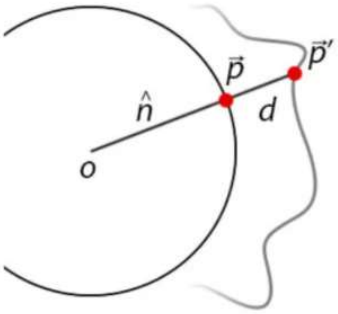
Jucatorul are posibilitatea sa lanseze proiectile spre inamici. Atunci cand acestea lovesc, inamicul respectiv va suferi o animatie de deformare realizata in Vertex Shader (vertex displacement), iar apoi va disparea. Jucatorul va putea astfel sa-si continue drumul. Deformarea pe care inamicii o sufera trebuie sa produca o modificare in geometria obiectului si nu doar o simpla transformare de rotatie/scalare/translatie.

Pentru deformarea primitivelor ce compun inamicii, puteți să aplicați de exemplu o funcție de zgomot pe direcția normalelor vârfurilor (exemplu în imaginea de mai jos). Animația de deformare va înceta dupa cateva secunde.

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 *(nu este necesara colorarea)*



HUD

Pe langa lumea inconjuratoare, trebuie sa desenati si un HUD (heads-up display). Elementele de HUD trebuie sa fie proiectate ortografic. Este necesar sa aveti cel putin 2 elemente care apar intotdeauna pe ecranul jucatorului in aceeasi pozitie, ca elemente de interfata grafica:

- **Viață**: Viata este indicata sub forma unui healthbar prin 2 dreptunghiuri, unul wireframe altul solid (vezi tema 1).
- **Timpul ramas**: Timpul ramas este indicat printr-o a doua bara de stare, in mod similar cu bara de viata.

Implementarea este la latitudinea voastra: puteti folosi un viewport separat, puteti "grupa" elementele de HUD cu jucatorul, sau orice alte variante.

Gameplay/Detalii de implementare

Scopul jocului este ca jucatorul sa iasa cu succes din labirint intr-un timp limita.

Asa cum s-a precizat mai sus, exista celule cu inamici care se deplaseaza incontinuu in interiorul acestor celule. De fiecare data cand jucatorul este atins de un inamic, acesta pierde din viata. Inamicul trebuie sa fie suficient de mare si/sau sa mearga suficient de rapid pentru a nu permite jucatorului sa il ocoleasca (usor), astfel acesta fiind nevoit sa traga in inamic.

Cand jucatorul iese din labirint, ramane fara viata sau fara timp, jocul se termina.

Crearea mediului inconjurator trebuie sa se realizeze in asa fel incat consumul de memorie si timpul de redare sa fie optime. Nu creati incontinuu obiecte care sa reprezinte inamici si proiectile care apar si dispar din spatiul de desenare!!!! O implementare eleganta este sa creati un singur obiect in functia init(), iar in functia Update() sa dati comanda de desenare pentru acel obiect de mai multe ori, de fiecare data la alta pozitie si cu alt factor de scalare.

Toate animatiile trebuie sa fie independente de timpul de procesare al unui cadru.

Exemple de functionalitati bonus

- HUD complex - de ex cu Minimap, Skills etc.
- Animarea avatarului de personaj
- Animarea mai avansata a inamicilor in vertex shader prin colorare si displacement complex
- Pick-up bonuses sau skill system cu mecanica de cooldown sau boost (modificarea proiectilelor, tragerea cu mai multe proiectile odata, health recovery, saritura (bolta) peste inamic cu o tasta etc.)
- Scene complexe
- Labirint care sa isi schimbe configuratia in timpul jocului (se misca, regenereaza configuratia din timp in timp)
- Schimbare culoare inamic in functie de viata ramasa
- Inamici inteligenti care urmaresc jucatorul dupa ce intra in patratul lor
- Boss fight (de ex o celula cu inamici de diverse tipuri, si un inamic mai 'capabil')

Functionalitati obligatorii

Barem orientativ pentru realizarea functionalitatilor (din 200 puncte):

- Harta
  - Generare (podea si obstacole) (30p)
  - Pozitionarea initiala corecta (10p)
- Personajul
  - Constructie (10p)
  - Deplasare (miscare si orientare) (20p)
  - First/third-person camera (15p)
  - Proiectile (desenare si miscare) (15p)
- Inamici
  - Deplasare (miscare) (20p)
  - Animare in shader (30p)
- Coliziuni
  - jucator - perete (10p)
  - jucator - inamic (10p)
  - proiectil - inamic (10p)
- Viata si timp (functionalitate si UI) - 20p

Intrebari si raspunsuri

Pentru intrebari vom folosi forumurile de pe moodle. Orice nu este mentionat in tema este la latitudinea fiecarui student!

Notare

Baremul este orientativ. Fiecare asistent are o anumita libertate in evaluarea temelor (de exemplu, sa dea punctaj partial pentru implementarea incompleta a unei functionalitati sau sa scada pentru hard coding). Acelasi lucru este valabil atat pentru functionalitatile obligatorii, cat si pentru bonusuri.

Tema trebuie incarcata pe moodle. Pentru a fi punctata, tema trebuie prezentata la laborator. Vor exista laboratoare speciale de prezentare a temelor (care vor fi anuntate).

Indicatii suplimentare

Tema va fi implementata in OpenGL si C++. Este indicat sa folositi framework-ul si Visual Studio.

Pentru implementarea temei, in folderul **src/lab_m1** puteti crea un nou folder, de exemplu Tema2, cu fisierele Tema2.cpp si Tema2.h (pentru implementare POO, este indicat sa aveti si alte fisiere). Pentru a vedea fisierele nou create in Visual Studio in Solution Explorer, apasati click dreapta pe filtrul lab_m1 si selectati Add→New Filter. Dupa ce creati un nou filtru, de exemplu Tema2, dati click dreapta si selectati Add→Existing Item. Astfel adaugati toate fisierele din folderul nou creat. In fisierul lab_list.h trebuie adaugata si calea catre header-ul temei. De exemplu: #include "lab_m1/Tema2/Tema2.h"

Arhivarea proiectului

- in mod normal arhiva trebuie sa contina toate resursele necesare compilarii si rularii
- inainte de a face arhiva asigurati-va ca ati curatat proiectul Visual Studio:
  - click dreapta pe proiect in **Solution Explorer → Clean Solution**
  - si stergeti folderul **/build/.vs** (daca nu il vedeti, **este posibil sa fie ascuns**)
- SAU stergeti complet folderul **/build**
- in cazul in care arhiva tot depaseste limita de 50MB (nu ar trebui), puteti sa stergeti si folderul **/deps** sau **/assets** intrucat se pot adauga la testare. Nu este recomandat sa faceti acest lucru intrucat ingreuneaza mult testarea in cazul in care versiunea curenta a librariilor/resurselor difera de versiunea utilizata la momentul scrierii temei.

Deadline tema

17 decembrie ora 23:59

Responsabili tema

- Anca Balutoiu
- Alex Gradinaru
- Florin Iancu
- Philip Dumitru

egc/teme/2021/02.txt · Last modified: 2021/12/08 14:51 by andrei.lambru