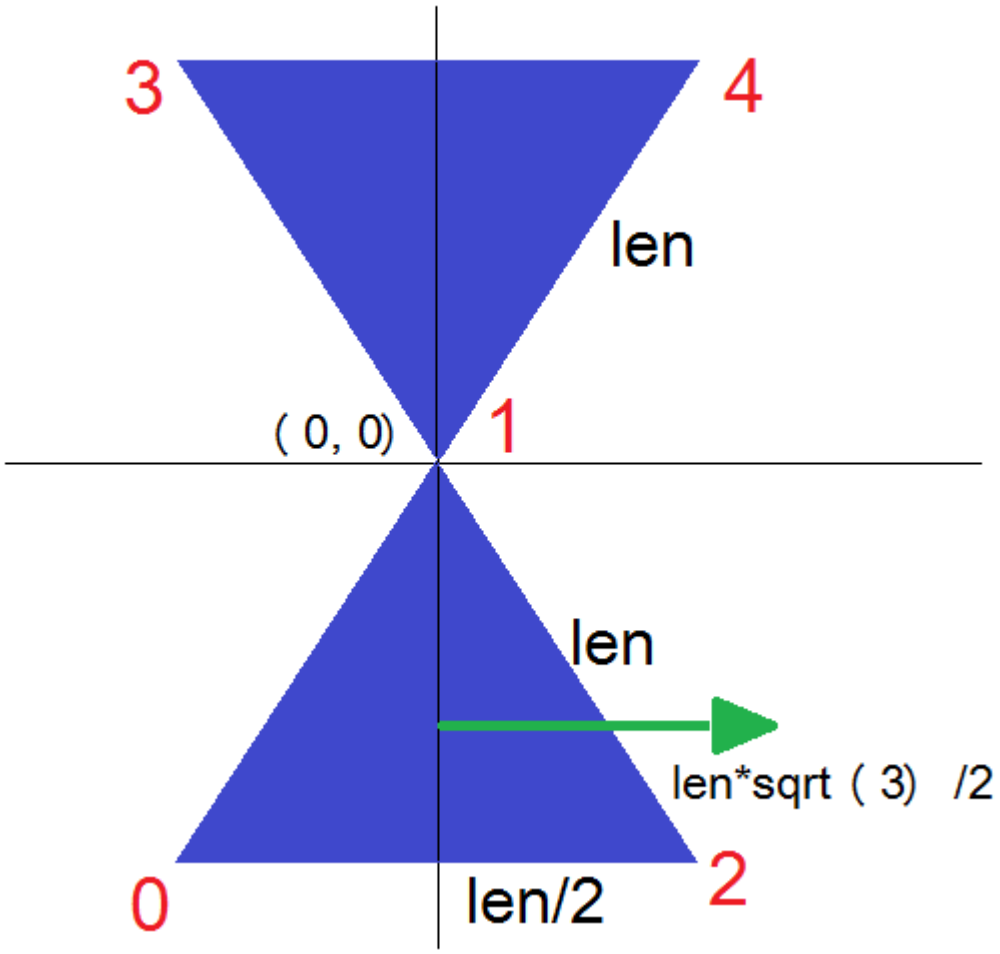
Gherghina Cristian-Catalin 333CC

Tema 1 EGC – War games

Crearea navei

In clasa Object2D am creat o functie noua cu antetul:

```
Mesh* CreateShip(std::string name, glm::vec3 leftBottomCorner, float length, glm::vec3 color, bool fill)
```



Folosing teorema lui Pitagora am calculat care este inaltimea unui triunghi (len * sqrt(3)/2). Nava are nevoie de 2 variable translateX si translateY in care retin mereu pozitia navei raportata la coltul jos stanga. Viteza navei este de 3.5f si initial unghiul cu care este inclinata este 0 (angularStep). In functia Init din Laborator3.cpp creez nava, o adaug hashmap-ului de meshe sub numele “my_ship” si o translatez in mijlocul ecranului), dupa care o randez.

De asemenea in init creez si vectorul de vieti. Am folosit o structura LifeStick definita astfel:

```
typedef struct LifeStick {
	float tx, ty;
	Mesh* mStick;
	string name;
} LifeStick;
```

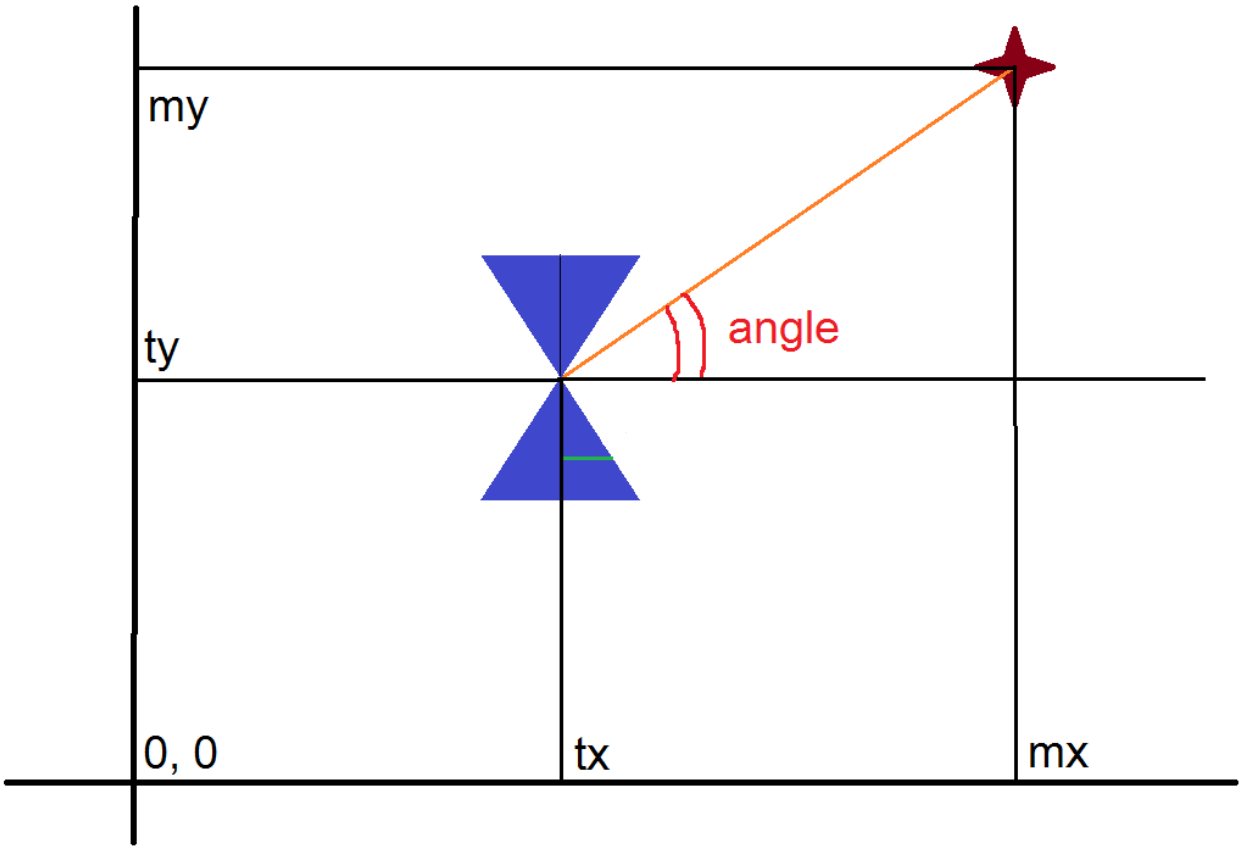
Intr-un vector de 3 elemente retin cele 3 dreptunghiuri care reprezinta numarul de vieti ale jucatorului, deoarece initial jucatorul porneste cu 3 vieti. La fiecare update le randez in coltul dreapta sus.

Miscarea navei pe W,A,S,D

In functia `OnInputUpdate` pur si simplu modific coordonatele navei, adica cresc sau scad tx/ty cu viteza navei.

Rotatia dupa mouse

In functia `OnMouseMove` din Laborator3.cpp actualizez angularStep al navei. Exista 4 cazuri in functie de cele 4 cadrane. Voi explica doar pentru primul cadran. Deoarece eu am punctul de origine in stanga jos, iar mouse-ul il are in coltul stanga sus trebuie modificat mouseY pentru a corespunde aceluiasi sistem. Astfel fac modificarea `actualMouseY = window->GetResolution().y - mouseY;`



Luam triunghiul din imagine. tg(angle) = latura_opusa/latura_alaturata. => tg(angle) = (my-ty)/(mx-tx) => angle = arctg((my-ty)/(mx-tx)). Pentru celelalte cazuri este asemanator doar ca trebuie sa avem grija la ce laturi folosim.

Gloantele

Pentru gloante am folosit strucura definite astfel:

```
typedef struct Bullet {
	Mesh* mBullet;
	float angularStep;
	float tx, ty;
	string name;
} Bullet;
```

In functia `OnMouseBtnPress` creez un bullet nou si ii atribui coordonatele navei, cat si unghiul navei din acel moment. Adaug bulletul in lista de meshe, dar de asemenea in vectorul bulletVector in care retin toate gloantele trase (in momentul in care se face o coliziune sau iese din ecran, voi sterge acele gloante).

Directia glontului este usor de construit folosind o formula clasica, stiind punctul de pornire si panta sau unghiul dreptei, deci este usor de generat urmatorul glont. Formula este urmatoarea: `tx += tx_multiplyer`, unde `tx_multiplyer = bullet_speed * cos(as);` Analog sip e axa OY. Din nou, intervin cele 4 cazuri in functie de cele 4 cadrane din cauza semnelor cos si sin.

Cele 4 cazuri sunt urmatoarele:

```
if (as < M_PI / 2) { tx += tx_multiplyer; ty += ty_multiplyer; }
if (as > M_PI / 2 && as < M_PI) { tx -= -tx_multiplyer; ty += ty_multiplyer; }
if (as > M_PI && as < 3 * M_PI / 2) { tx -= -tx_multiplyer; ty -= -ty_multiplyer; }
if (as > 3 * M_PI / 2 && as < 2 * M_PI) { tx += tx_multiplyer; ty -= -ty_multiplyer; }
```

Dupa ce am calculat care va fi urmatorul punct al glontului, il randez in acea pozitie. Acest lucru se face la fiecare update.

Inamicii

Folosesc urmatoare structura pentru inamici:

```
typedef struct Enemy {
	Mesh* mEnemyShip;
	float angularStep;
	float tx, ty;
	string name;
	int type; // for scaling
	float speed;
	int remaining_lives;
	float scaleX, scaleY;
} Enemy;
```

De asemenea, folosesc un vector de Enemy in care retin fiecare inamic. Inamicii ii generez la fiecare interval, care initial este 2 secunde si care scade pana la 0.5 secunde. Deoarece este nevoie de a ii genera pe marginea cercului de raza constanta pe care am definit-o la inceputul

programului ca fiind `resolution.x / 2 - 50`. Ea se poate modifica usor de la constant la dinamic, dar nu am considerat ca acest lucru este necesar.

Generarea pozitiei inamicului am facut-o astfel. Este nevoie de a fixa intial o axa. Eu am ales sa fixez coordonata pe axa OX. Astfel ex poate lua orice valoare din intervalul [tx-raza, tx+raza]. Acest lucru il generez random. Apoi folosind ecuatia cercului, determin care este coordonata ey pentru enemy. Ecuatia este sqrt($(ex-tx)^2 + (ey-ty)^2$) = $raza^2$. Stiind care este ex, calculcam ey ca fiind ty +/- sqrt(raza2 - $(ex-tx)^2$). Daca este cu + va fi in jumatatea superioara a ecranului, iar cu – in jumatatea inferioara a ecranului. Acest lucru este generat random de un numar care ia valori ori 1 ori 0.

In functia update se randeaza fiecare enemy. De asemenea miscarea si inclinatia inamicului este in functie de cele 4 cadrane. Unghiul de rotatie al fiecarui inamic este necesar sa il calculez la fiecare update, deoarece inamicul trebuie sa urmareasca nava jucatorului. Este asemanator cu calcularea unghiului pentru nava dupa mouse.

Inamicii sunt de 2 feluri, albastri si galbeni. Cei albastri au o viata, cei galbeni au 2.

Coliziunea

Folosesc coliziunea de tip sfera, adica daca distanta dintre centrul glontului si centrul inamicului sau centrul inamicului si centrul navei jucatorului este mai mica decat lungimea unei laturi a navei, deoarece am ales sa folosesc un triunghi echilateral, pentru a fi mai usor de calculat, atunci are loc o coliziune si trebuie sa distrug acele obiecte.

Verific pentru fiecare enemy daca are loc coliziune cu nava jucatorului, daca nu verific daca are coliziune cu vreun glont. In cazul in care are loc o coliziune intre glont si un inamic galben, care are 2 vieti, acesta trebuie scalat la jumatate. Fac acest lucru in 2 update-uri deoarece deltaTimeSeconds este cam 0.16 si este nevoie de o tranzitie de aprox 0.25 secunde. In momentul coliziunii, clonez acel inamic, doar ca ii schimb niste proprietati.

In functia update de asemenea o sa sterg acei inamici si acele gloante care intra in coliziune.

Sfarsitul jocului

In momentul in care jucatorul moare de 3 ori, jocul se termina si are loc o tranzitie de 2 secunde in care ecranul se inroseste. Pentru a face acest lucru, calculez mai intai in cate update-uri trebuie facuta tranzitia. `pieces_for_game_over_anim = (int) (2.0f / deltaTimeSeconds);`

Apoi, pornind de la 0, adun cate o fractiune la indexul R din culoarea ecranului si updatez acest lucru.

```
if (player_nr_lives == 0 && starting_color <= 1.0f) {
	glClearColor(starting_color, 0.0f, 0.0f, 1.0f);
	starting_color += 1.0f / pieces_for_game_over_anim;
	glClear(GL_COLOR_BUFFER_BIT);
}
```

Tema a fost foarte interesanta si mi-a placut. Am pus in aplicare cunostintele de matematica (geometrie) si ce am invatat la laborator. Acord temei o nota de 7 ca dificultate.