AISandbox – code tutorial

Zmiana zachowania aktorów

# Cel

Celem tutoriala jest zapoznanie użytkownika ze sposobem zmiany zachowania aktorów. Po przeczytaniu dokumentu użytkownik będzie wiedział, jak:

* Utworzyć nowy kontroler
* Sterować ruchem aktora
* Sprawić, by aktor strzelał

Po wykonaniu wszystkich kroków opisanych w dokumencie, uruchomienie programu będzie miało następujące efekty:

* Zostanie wczytany poziom z meshem „2nomansland”
* Zostaną wyspawnowane 2 grupy aktorów z przeciwnych drużyn (niebieska i czerwona) sterowane utworzonym kontrolerem
* Każdy aktor będzie co 3 sekundy skakał, strzelał oraz zmieniał kierunek i szybkość ruchu na losowe

# Utworzenie nowego kontrolera

Kontroler nazwiemy TutorialActorController. Należy utworzyć pliki TutorialActorController.h i TutorialActorController.cpp oraz zdefiniować klasę TutorialActorController dziedziczącą publicznie po klasie IActorController. Następnie należy zdefiniować metody:

* void onCreate()
* void onTakeDamage()
* void onUpdate(float dt)
* void onDebugDraw()

Konstruktor klasy powinien być zdefiniowany jako explicit z parametrem typu ActorAI\*.

Po zdefiniowaniu klasy należy dodać ją do fabryki ActorControllerFactory – w pliku ActorControllerFactory.cpp należy włączyć nagłówek kontrolera oraz dodać wpis do metody ActorControllerFactory::create o następującej treści:

CREATE\_CONTROLLER(Tutorial);

Po dodaniu kontrolera do fabryki możliwe będzie dynamiczne tworzenie aktorów używających kontrolera, podając jego nazwę – „Tutorial” (bez końcówki ActorController).

# Sterowanie aktorem

Ruchem aktora sterują dwie metody klasy Character:

* void setDirection(const mkVec& dir) – jako argument należy podać kierunek aktora w przestrzeni świata. Wektor nie musi być znormalizowany, jego długość nie ma wpływu na szybkość ani kierunek
* void setSpeed(float speed) – jako argument należy podać liczbę 0..1, gdzie 0 oznacza zerową prędkość, zaś 1 maksymalną prędkość aktora

Zatem aby aktor zmieniał kierunek i szybkość ruchu co 3 sekundy, do metody onUpdate kontrolera należy dodać następujący kod:

const float cur\_time = g\_game->getTimeMs();

const float turn\_period = 3000.f;

const float time\_since\_dir\_change = m\_lastTurnTime < 0 ? turn\_period : cur\_time - m\_lastTurnTime;

if (time\_since\_dir\_change >= turn\_period)

{

getAI()->setDirection(getRandomHorizontalDir());

getAI()->setSpeed(randFloat(.5f, 1.f));

m\_lastTurnTime = cur\_time;

}

Należy także dodać pole m\_lastTurnTime i zainicjować je na -1.f; Skok realizowany jest za pomocą metody jump, strzał – shoot. Aby razem ze zmianą prędkość nastąpił skok i strzał:

getAI()->jump();  
getAI()->shoot();

# Definicja mapy

Należy utworzyć plik tutorial.json w katalogu build/data/levels, a następnie dodać następującą treść:

{

"mesh" : "2nomansland.mesh",

"player\_spawn\_pos" : [-25.2942, 1.2370, 1.8834],

"ai" :

{

"can\_jump" : true,

"max\_speed" : 2.8,

"sight\_distance" : 20.0,

"hor\_sight\_angle\_deg" : 45.0,

"shooting\_range" : 8.0,

"spawn\_radius" : 3.0,

"shooting\_dmg" : 25.0,

"ai\_group" :

{

"controller" : "Tutorial",

"conflict\_side" : "BlueTeam",

"prefab" : "actor\_ai\_blue",

"spawn\_around" : [-27.6264, 5.0, 2.3333],

"number" : 5

},

"ai\_group" :

{

"controller" : "Tutorial",

"conflict\_side" : "RedTeam",

"prefab" : "actor\_ai\_red",

"spawn\_around" : [27.4202, 5.0, -0.3792],

"number" : 5

},

},

}

Znaczenie poszczególnych wpisów wyjaśnione jest w dokumentacji projektu. Najważniejszy wpis to „controller" o wartości Tutorial – mówi ona, aby użyć nowoutworzonego kontrolera do sterowania.

# Uruchomienie programu

Aby przetestować utworzony kontroler, należy uruchomić program z argumentem „-level tutorial", np. za pomocą skryptu run\_args.cmd. Katalogiem roboczym programu musi być katalog build. Szczegóły na temat sposobu prowadzenia obserwacji znajdują się w tutorialu dla użytkownika.