Main.cpp

#include <iostream>

#include "Game.h"

#include "GameComponent.h"

#include "DrawableGameComponent.h"

void initialising(){

cout << "Initialising Game" << endl;

}

void terminating(){

cout << "Terminating Game" << endl;

}

int main() {

Game \*newGame = new Game(2);

newGame->SetInitialise(initialising);

newGame->SetTerminate(terminating);

newGame->Add(new GameComponent());

newGame->Add(new DrawableGameComponent(0, 0));

newGame->Run();

}

Game.h

#ifndef TASK1\_GAME\_H

#define TASK1\_GAME\_H

#include <ctime>

#include "GameComponent.h"

typedef void(\*FP)();

class Game {

private:

int componentCount;

GameComponent\*\* components;

FP initialise;

FP terminate;

const int TICKS\_1000MS = 1;

// ^ the sleep() function takes seconds as a parameter, so using this as milliseconds

// didn't make much sense. I still assigned it the value of one for consistency of the assignment, but it

// represents one second as a whole

public:

Game(int maxComponents);

~Game();

void Add(GameComponent\*);

void Run();

void SetInitialise(FP init);

void SetTerminate(FP term);

};

#endif //TASK1\_GAME\_H

Game.cpp

#include <unistd.h>

#include "Game.h"

Game::Game(int maxComponents) {

components = new GameComponent\*[maxComponents];

componentCount = 0;

}

Game::~Game() {

}

void Game::Add(GameComponent\* g) {

components[componentCount] = g;

componentCount++;

}

void Game::Run() {

initialise();

for (int j = 0; j < 5; ++j) {

sleep(TICKS\_1000MS); // 1 second

for (int i = 0; i < componentCount; ++i) {

time\_t rawtime;

struct tm \*timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

this->components[i]->Update(timeinfo);

}

}

terminate();

}

void Game::SetInitialise(FP init) {

Game::initialise = init;

}

void Game::SetTerminate(FP term) {

Game::terminate = term;

}

GameComponent.h

#ifndef TASK1\_GAMECOMPONENT\_H

#define TASK1\_GAMECOMPONENT\_H

#include <ctime>

#include "iostream"

using namespace std;

class GameComponent {

protected:

int id;

static int instances;

public:

GameComponent();

~GameComponent();

virtual void Update(const tm\*);

};

#endif //TASK1\_GAMECOMPONENT\_H

GameComponent.cpp

#include "GameComponent.h"

int GameComponent::instances = 0;

GameComponent::GameComponent(){

instances++;

id = instances;

}

GameComponent::~GameComponent() {

}

void GameComponent::Update(const tm\* eventTime) {

cout << "ID: " << this->id << " Time: " << asctime(eventTime);

}

DrawableGameComponent.h

#ifndef TASK1\_DRAWABLEGAMECOMPONENT\_H

#define TASK1\_DRAWABLEGAMECOMPONENT\_H

#include "GameComponent.h"

class DrawableGameComponent: public GameComponent {

private:

void ChangeDirection();

void Draw();

int x;

int y;

public:

enum Direction {Left, Right, Up, Down};

Direction direction;

DrawableGameComponent(int x, int y);

~DrawableGameComponent();

void Update(const tm\* eventTime);

const int SCREEN\_HEIGHT = 20;

const int SCREEN\_WIDTH = 80;

};

#endif //TASK1\_DRAWABLEGAMECOMPONENT\_H

DrawableGameComponent.cpp

#include "DrawableGameComponent.h"

string EnumToString (DrawableGameComponent::Direction direction) {

switch (direction) {

case DrawableGameComponent::Right:

return "Right";

case DrawableGameComponent::Left:

return "Left";

case DrawableGameComponent::Up:

return "Up";

case DrawableGameComponent::Down:

return "Down";

}

}

DrawableGameComponent::DrawableGameComponent(int x, int y): x(x), y(y), GameComponent() {

direction = Direction::Right;

}

DrawableGameComponent::~DrawableGameComponent() {

}

void DrawableGameComponent::ChangeDirection() {

Direction oldDirection = direction;

while (direction == oldDirection ) {

srand(time(0));

direction = Direction(rand() % 4);

}

}

void DrawableGameComponent::Draw() {

cout << "Direction: " << EnumToString(direction) << " x: " << x << " y: " << y << endl;

}

void DrawableGameComponent::Update(const tm \*eventTime) {

cout << "ID: " << this->id << " Time: " << asctime(eventTime);

switch (direction) {

case Right:

x < SCREEN\_WIDTH ? x++ : x=SCREEN\_WIDTH;

break;

case Left:

x > 0 ? x-- : x=0;

break;

case Up:

y < SCREEN\_HEIGHT ? y++ : y=SCREEN\_HEIGHT;

break;

case Down:

y > 0 ? y-- : y=0;

break;

}

Draw();

ChangeDirection();

}