Programming Systems and Environments – Lab 9

Jakub Grzana, 241530

Bouncing balls

I've created program that asks user for number of balls to be used, and length of "court" (path that each ball has to traverse) Balls starts in random order, and move with random speed.

Different speed is achieved by waiting for randomly generated time, given in ms. Ball attempts to move every X ms. Lower X, faster speed.

I've also created "static" class Tools (the fact that C++ doesn't support static classes isn't a problem, just minor inconvenience) with… well, tools useful in this and incoming program.

To generate random numbers, I use MT19937 generator initiated with random state, generating values from uniform integer distribution. Thanks to C++ keywords 'static' and 'thread_local' I don't need to protect generator with mutex.

For display, I used ncurses library, but for better abstraction I've created GraphicalObject class that represents pixel in terminal. I didn't try to make full abstract wrapper tho, too much work.

There's function to print all balls on the screen, and another to move each ball. Every ball is operated by separate thread. Printing is managed by single, created for it thread.

There's minor bug that can potentially happen: if printing thread attempts to read position of ball, while another thread is reading it, race conditions occur and there's chance that printing thread get trash values. However in practice I didn't notice any odd behavior and besides, this wouldn't be a fatal error, thus I ignored it.

C++ has incredibly good multithreading support, much better than pthread (tho it does use pthread on linux internally) I love code that works on multiple platforms… thus I don't like ncurses.

Bibliography:

C++ reference: https://en.cppreference.com/w/
I didn't use anything more. All comes from my head.

CODE:

```
#include <ncurses.h>
#include <thread>
#include <list>
#include <chrono>
#include <iostream>
#include <random>
#include <cstdlib>

class Tools
{
	public: static int RandomInt(const int& min, const int& max) {
		static thread_local std::random_device rd;
		static thread_local std::mt19937 generator(rd());
		std::uniform_int_distribution<int> distribution(min, max);
		return distribution(generator);
	}

	public: class GraphicalObject
	{
		private:
			int x;
			int y;
			char letter;
		public:
			int GetPosX() const { return this->x; }
			int GetPosY() const { return this->y; }
			char GetLetter() const { return this->letter; }
			void SetPosX(int pos) { this->x = pos; }
			void SetPosY(int pos) { this->y = pos; }
			void SetLetter(char letter) { this->letter = letter; }
			void Print() const { mvprintw(this->GetPosY(), this->GetPosX(),
"%c", this->GetLetter()); }
			GraphicalObject() = delete;
			GraphicalObject(int x, int y, char letter) { this->x = x; this->y = y;
this->letter = letter; }
	};

	public: Tools() = delete;
};

class Ball : public Tools::GraphicalObject
{
	private:
		bool downward;
```

```
	public:
		void Move() {
				if(this->downward) { this->SetPosY(this->GetPosY() + 1); }
				else { this->SetPosY(this->GetPosY() - 1); }
			}
		void ToggleDirection() { this->downward = not this->downward; }
		Ball() = delete;
		Ball(int x, int y, char letter) : GraphicalObject(x,y,letter) { this-
>downward = true;}
};

void print(const bool& stop, const std::list<Ball>& objs)
{
	curs_set(0);
	while(!stop)
	{
		erase();
		for(const auto& obj : objs) { obj.Print(); }
		refresh();
	}
}

void ball_movement(const bool& stop, Ball& ball, const int& court_length)
{
	int initial_delay = Tools::RandomInt(200,1000);
	int tick_delay = Tools::RandomInt(300,500);
	std::this_thread::sleep_for(std::chrono::milliseconds(initial_delay));
	while(!stop)
	{
		std::this_thread::sleep_for(std::chrono::milliseconds(tick_delay));
		ball.Move();
		if(ball.GetPosY() == court_length || ball.GetPosY() == 0) {
ball.ToggleDirection(); }
	}
}

int BallsExercise(const int ballnum, const int court_length)
{
	initscr();

	/* VARIABLES */
	bool stop = false;
	std::list<Ball> balls;
	std::list<std::thread> threads;
```

```
	/* CREATION OF BALLS */
	for(int i = 0; i < ballnum; i++)
	{ balls.push_back( Ball(i, 0, 'O') ); }

	/* STARTING THREADS */
	threads.push_back( std::thread( print, std::ref(stop), std::ref(balls) ));
	for(auto& ball : balls)
	{
		threads.push_back( std::thread( ball_movement, std::ref(stop),
std::ref(ball), std::ref(court_length) ));
	}

	/* LISTENING FOR SPECIFIC KEY */
	char input = 0; while(input != 'G') { input = getch(); } stop = true;

	/* JOINING THREADS */
	for(auto& thread : threads)
	{ thread.join(); }

	endwin();
	return 0;
}

int main()
{
	std::cout << "Type in number of balls to be used" << std::endl;
	int ballnum = 0; std::cin >> ballnum;
	std::cout << "Type in length of court" << std::endl;
	int court_length = 0; std::cin >> court_length;
	return BallsExercise(ballnum, court_length);
}
```