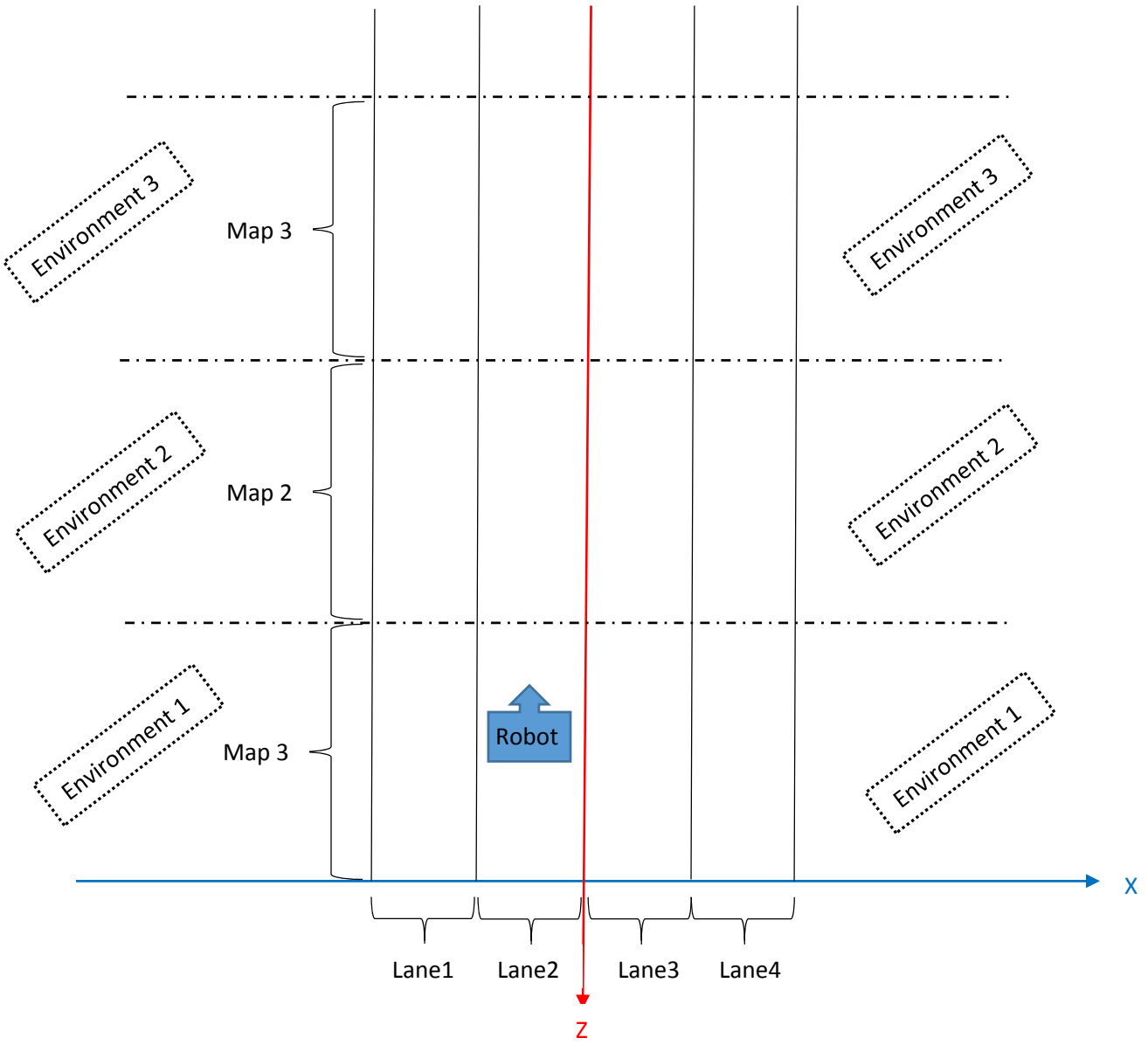
گزارش پروژه درس گرافیک کامپیوتری – شماره یک

محمّدرضا ابوالحسنی 610389084

قبلاً گفتیم که سبک انتخاب شده برای بازی سبک infinite runner است. ایده اصلی این است که ابتدا تعدادی تکه مسیر ساخته می شوند. سپس تعدادی (مثلاً سه تا برای تأمین افق دید) از این تکه مسیر ها انتخاب شده و پشت سر هم نمایش داده می شوند. هرگاه به انتهای تکه مسیر اول رسیدیم این تکه مسیر را از رده خارج می کنیم و یک تکه مسیر جدید انتخاب کرده و به انتهای مسیر اضافه می کنیم.

در این گزارش ابتدا به توضیح تم بازی می پردازیم ، سپس ساختار کد و الگوریتم تولید تصادفی مسیر را بررسی می کنیم.

بازی از یک مسیر مستقیم چهار لاینه تشکیل شده. شخصیت اصلی (که برای جلوگیری از دردسر های ساخت انیمیشن یک ربات تک چرخ در نظر گرفته شده) به طور اتوماتیک در این مسیر پیش می رود. بازیکن باید با حرکت به چپ و راست یا پریدن از موانع موجود جاخالی بدهد! نمای محیط بازی از بالا به این گونه خواهد بود:



اشیاء موجود در بازی دو دسته هستند. اشیاء تزئینی (Decorative) و موانع (Obstacle) . اشیاء تزئینی در دو طرف مسیر و موانع داخل مسیر قرار می گیرند. یک environment شامل تعدادی اشیاء تزئینی است که اطلاعات آنها (شامل نوع ، موقعیت ، زوایا با محورها و scale) از روی یک فایل در زمان ساخت آن لود می شوند که در آن فایل هر خط حاوی اطلاعات یک شیء است. به طور مشابه یک map شامل تعدادی مانع است. یک بلوک (یا همان تکه مسیر) از ترکیب یک map و یک environment تشکیل می شود که هر دو به طور تصادفی انتخاب می شوند.

حال به توضیح کد می پردازیم.

ابتدا لازم است توضیح مختصری درباره مفهوم لیست نمایش (display list) ارائه کنیم. ایده اصلی این است که برای نمایش یک شیء به جای آنکه در هر فریم اطلاعات آن شیء (شامل مختصات رئوس ، نرمال ها ، تکسچر و ...) را به openGL اعلام کنیم، یکبار در ابتدای برنامه این کار را انجام دهیم و از openGL بخواهیم این اطلاعات را در یک لیست نمایش ذخیره کند و یک ID به ما برگرداند تا هر وقت خواستیم آن شیء را نمایش دهیم، فقط آن ID را اعلام کنیم. ID برگردانده شده از نوع GLuint (مشابه unsigned int در c++) خواهد بود. فرض کنیم یک شیء ماشین داریم که اطلاعات آن در فایل car.obj قرار دارد. نحوه انجام این عمل با استفاده از کتابخانه glm به این صورت خواهد بود:

در ابتدای برنامه (مثلاً در روال main()) می نویسیم:

```
GLMmodel* model = glmReadOBJ("objects/car.obj");
GLuint displaylist_id = glmList(model, GLM_SMOOTH|GLM_MATERIAL|GLM_TEXTURE);
```

حال هر بار بخواهیم ماشین را رسم کنیم کافیست بنویسیم:

```
glCallList(displaylist_id);
```

حتّی اگر چند ماشین در صحنه داشته باشیم باز هم نیاز به ارسال چندباره اطلاعات نیست:

```
glTranslatef(محل ماشین اول);
glCallList(displaylist_id);
glTranslatef(محل ماشین دوم);
glCallList(displaylist_id);
...
```

می خواهیم تمام اشیاء را در ابتدای برنامه لود کنیم و ID های لیست های نمایش را در یک آرایه ذخیره کنیم. برای آنکه بدانیم ID هر شیء در کدام اندیس آرایه ذخیره شده تا بعداً به راحتی به آن دسترسی داشته باشیم یک نوع داده ای ترتیبی (enumerated) می سازیم به همراه یک تابع که آدرس فایل obj. هر شیء را بر میگرداند:

```
//file obj_names.h
#include <string>

#ifndef OBJ_NAMES
#define OBJ_NAMES

enum OBJNAME {SHIP, CRANE, BOX, OBSTACLE,/*add other obj names here*/ NUM_OF_OBJS};

string obj_path(OBJNAME obj){
	switch(obj){
		case SHIP:
		return "objects/ship.obj";

		case CRANE:
		return "objects/crane.obj";

		case BOX:
		return "objects/box.obj";

		case OBSTACLE:
```

```
                return "objects/obstacle.obj";

                /*handle other obj names here*/

                default:
                return "invalid";

        }
}

#endif
```

توجّه کنید که آخرین عضو نوع داده ای ترتیبی NUM_OF_OBJS قرار داده شده که به صورت اتوماتیک تعداد اشیاء را بر می گرداند. حال می توانیم تمام اشیاء را در یک حلقه لود کنیم:

```
GLuint displaylist_ids[NUM_OF_OBJS];
GLMmodel* model;
for(int i = 0; i < NUM_OF_OBJS; i++){
        model = glmReadOBJ(obj_path(i));
        displaylist_ids[i] = glmList(model, GLM_SMOOTH|GLM_MATERIAL|GLM_TEXTURE);
}
```

با این توضیحات به سراغ کد می رویم. فایل main.cpp (به صورت مختصر و مفید!) به این صورت است: (به قسمت های هایلایت شده توجه فرمایید)

```
#include <GL/glut.h>
#include "game.cpp"

game* the_game;

//Initializes 3D rendering
void initRendering() {
glEnable(GL_DEPTH_TEST);
        glEnable(GL_LIGHTING); //Enable lighting
        glEnable(GL_LIGHT0); //Enable light #0
        glEnable(GL_LIGHT1); //Enable light #1
}

//Called when the window is resized
void handleResize(int w, int h) {
        //Tell OpenGL how to convert from coordinates to pixel values
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION); //Switch to setting the camera perspective
        //Set the camera perspective
        glLoadIdentity(); //Reset the camera
        gluPerspective(45.0,                  //The camera angle
                                (double)w / (double)h, //The width-to-height ratio
                                1.0,                   //The near z clipping coordinate
                                200.0);                //The far z clipping coordinate
}

int main(int argc, char** argv) {
        //Initialize GLUT
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(600, 600); //Set the window size
```

```
	//Create the window
	glutCreateWindow("The Game!"); //set window title
	initRendering(); //Initialize rendering

	the_ game = new game;

	//Set handler functions for drawing, keypresses, and window resizes
	glutDisplayFunc(the_game->draw);
	glutKeyboardFunc(the_game->handleKeypress);
	glutSpecialFunc(the_game->handleSpecialKeypress);
	glutReshapeFunc(handleResize);

	glutTimerFunc(25, game->update, 0); //Add a timer

	glutMainLoop(); //Start the main loop.  glutMainLoop doesn't return.
	return 0; //This line is never reached
}
```

حال نگاهی به فایل سرآمد کلاس game می اندازیم:

```
//file game.h
#include <GL/glut.h>
#include <stdlib.h>  /* srand, rand */
#include <time.h>    /* time (for srand) */
#include <string>

#include "glm.h"     /* modified glm library.
             original library by Nate Robins*/
#include "obj_names.h"
#include "map.h"
#include "environment.h"

#ifndef GAME_H
#define GAME_H

#define NUM_OF_MAPS 20 //number of maps
#define NUM_OF_ENVS 10 //number of environments
#define B_LEN 50 //blocks length
#define CAM_DIS 5 //camera distance from main character

enum STATUS {LOADING, PLAY, PAUSE, GAMEOVER};

class game
{
public:
	game();
	~game();
	void update(int);
	void draw();
	void handleSpecialKeypress(int, int, int);
	void handleKeypress(unsigned char, int, int);

private:
	unsigned long score;
	unsigned int env1_id, env2_id, env3_id;
```

```
        environment* envs[NUM_OF_ENVS];
        unsigned int map1_id, map2_id, map3_id;
        map* maps[NUM_OF_MAPS];
        robot the_robot;
        GLuint displaylist_ids[NUM_OF_OBJS];
        STATUS game_status;

        void load();
        void reset();
        void draw_ground();
        void collision_detect();
};

#endif
```

کامنت ها و اسامی تا حدودی گویای مطلب هستند. سعی می کنیم نکات نامفهوم را در توضیح روال ها بیان کنیم:

1. کانستراکتور:

```
game::game(){
    env1_id = env2_id = env3_id = 0;
    map1_id = map2_id = map3_id = 0;
    load();
    reset();
}
```

فقط دو روال دیگر را فراخوانی می کند که در ادامه آنها را توضیح می دهیم.

2. لود :

```
1)  void game::load(){
2)          //set game status
3)          game_status = LOADING;
4)
5)          /* pass opengl all objects data and store an id for
6)                  each  object to access them later */
7)          GLMmodel* model;
8)          for(int i = 0; i < NUM_OF_OBJS; i++){
9)                  model = glmReadOBJ(obj_path(i));
10)                 displaylist_ids[i] = glmList(model,
11)                         GLM_SMOOTH|GLM_MATERIAL|GLM_TEXTURE);
12)         }
13)
14)         //build all enviroments
15)         for(int i = 0, i < NUM_OF_ENVS, i++){
16)                 char first_digit = '0' + i/10;
17)                 char second_digit = '0' + i%10;
18)                 string filename = "environment_desciptions/env"
19)                         + first_digit + second_digit + ".des";
20)                 envs[i] = new environment(filename, displaylist_ids);
21)         }
22)
23)         //build all maps (path pieces)
24)         for(int i = 0, i < NUM_OF_MAPS, i++){
25)                 char first_digit = '0' + i/10;
26)                 char second_digit = '0' + i%10;
27)                 string filename = "map_desciptions/map"
28)                         + first_digit + second_digit + ".des";
```

```
29)                 maps[i] = new map(filename, displaylist_ids);
30)         }
31) }
```

این روال فقط یکبار در برنامه اجرا خواهد شد. کاری که انجام می دهد این است که ابتدا وضعیت را در حالت LOADING قرار می دهد. سپس تمام فایل های obj. را می خواند و لیستهای نمایش را تهیه می کند که قبلاً روش آن را توضیح دادیم. سپس تمام map ها و تمام environment ها را می سازد (یادآوری می کنیم که این کار فقط یکبار در طول برنامه انجام می شود)
برای ساخت یک map یا یک environment دو مقدار ورودی لازم است: یکی آدرس فایل متنی که نوع و مشخصات اشیاء را توضیح می دهد و یکی اشاره گر به آرایه ای که در آن ID های لیست های نمایشی را ذخیره کرده ایم تا هر شیء در روال draw() خود بداند که کدام لیست نمایش را باید صدا بزند.
ایده ای که پشت این حرکت است آنست که به جای آنکه در طول زمان بازی مدام تکه مسیر جدید بسازیم، فقط یکبار در ابتدا تمام تکه مسیرها را بسازیم و در آرایه ذخیره کنیم و در طول بازی اعداد صحیح تصادفی تولید کنیم و اگر مثلاً عدد i تولید شد map شماره iام را در انتهای مسیر نمایش دهیم. در ادامه این مطلب را بیشتر توضیح خواهیم داد.

3. ریست:

```
1)  void game::reset(){
2)         //reset score
3)         score = 0;
4)
5)         //reset main character
6)         the_robot.reset();
7)
8)         /*make sure to undo all the changes applied
9)                to used maps before we get rid of them*/
10)        maps[map1_id]->reset();
11)        maps[map2_id]->reset();
12)        maps[map3_id]->reset();
13)
14)        srand(time(NULL)); // seed random func
15)
16)        //choose new maps and environments
17)        map1_id = rand() % NUM_OF_MAPS;
18)        map2_id = rand() % NUM_OF_MAPS;
19)        map3_id = rand() % NUM_OF_MAPS;
20)        env1_id = rand() % NUM_OF_ENVS;
21)        env2_id = rand() % NUM_OF_ENVS;
22)        env3_id = rand() % NUM_OF_ENVS;
23)
24)        //srart new game
25)        game_status = PLAY;
26) }
```

هدف اصلی از طرّاحی این روال این است که هربار که بازیکن می بازد و می خواهد دوباره بازی کند، نیازی به انجام دوباره LOADING نباشد. این روال ابتدا اطمینان حاصل می کند که تمام تغییرات حاصل از دست قبل بازی خنثی شوند (خطوط 3 تا 12). همانطور که قبلاً گفته شد در هر لحظه سه بلوک شامل سه map و سه environment مسیر بازی را تشکیل می دهند که با تولید سه عدد صحیح تصادفی بین 0 و تعداد map ها و سه عدد صحیح تصادفی دیگر بین 0 و تعداد environment ها انتخاب می شوند. وقتی که بازیکن به انتهای بلوک اول می رسد، باید بلوک اول را از رده خارج کرد و یک بلوک جدید ساخت، یعنی یک map و یک environment جدید انتخاب کرد. این کار در روال update انجام می شود که در ادامه توضیح خواهیم داد.

4. آپدیت:

```
1)  void game::update(int value){
2)         if(game_status == PLAY){
3)                 score++;
4)                 maps[map1_id]->update();
5)                 the_robot.update();
```

```
6)              collision_detect();
7)
8)              /*in case we've reached the end of block, we should
9)                      get rid of the first block and choose a new one
10)                                                     to be added to the end*/
11)             if(the_robot.getPosZ() >= B_LEN){  B_LEN : block length
12)                     //reset first map before getting rid of it
13)                     maps[map1_id]->reset();
14)
15)                     map1_id = map2_id;
16)                     map2_id = map3_id;
17)                     map3_id = rand() % NUM_OF_MAPS;
18)
19)                     env1_id = env2_id;
20)                     env2_id = env3_id;
21)                     env3_id = rand() % NUM_OF_ENVS;
22)
23)                     //now main character should get back to the beginning of block
24)                     the_robot.setPosZ(0);
25)             }
26)     }
27)     glutPostRedisplay();
28)     glutTimerFunc(25, this->update, 0);
29) }
```

این روال هر 25 میلی ثانیه اجرا می شود. اگر بازی در حالت PLAY نباشد باید منوهای مربوطه نمایش داده شوند و نیاز به انجام کار خاصی نیست. در غیر اینصورت امتیاز را زیاد می کند، map اول را (که بازیکن همواره در این map قرار دارد!) به روز رسانی می کند (بعضی موانع متحرک هستند و باید مختصاتشان تغییر کند) ، موقعیت ربات را به روز رسانی می کند و روال collision_detect() را فرا خوانی می کند. در این روال بررسی می شود که آیا برخوردی رخ داده یا خیر که در صورت برخورد بازی در حالت GAMEOVER قرار می گیرد.
سپس چک می کند که آیا به انتهای بلوک رسیده ایم یا خیر (خط 11). اگر به انتهای بلوک رسیده باشیم ، بلوک دوم را جایگزین بلوک اوّل و بلوک سوم را جایگزین بلوک دوم می کنیم و یک بلوک جدید به صورت تصادفی انتخاب می کنیم:

```
map1_id = map2_id;
map2_id = map3_id;
map3_id = rand() % NUM_OF_MAPS;

env1_id = env2_id;
env2_id = env3_id;
env3_id = rand() % NUM_OF_ENVS;
```

سپس ربات را به ابتدای بلوک اوّل (بلوک دوم سابق) بر می گردانیم (خط 24).

5. کیبرد

```
1)  void game::handleKeypress(unsigned char key, int x, int y) {
2)      switch (key) {
3)              case 27: //Escape key
4)                      exit(0);
5)                      break;
6)
7)              case '\r': //Enter key
8)                      if(game_status == PLAY)
9)                              game_status = PAUSE;
10)                     if(game_status == PAUSE)
11)                             game_status = PLAY;
```

```
12)                     break;
13)             case ' ': //space bar
14)                     the_robot.signalShoot();
15)     }
16) }
17)
18) //Called when a special key is pressed
19) void game::handleSpecialKeypress(int key, int x, int y) {
20)     switch (key) {
21)             case GLUT_KEY_RIGHT:
22)                     the_robot.signalMoveRight();
23)                     break;
24)
25)             case GLUT_KEY_LEFT:
26)                     the_robot.signalMoveLeft();
27)                     break;
28)
29)             case GLUT_KEY_DOWN:
30)                     the_robot.signalSit();
31)                     break;
32)
33)             case GLUT_KEY_UP:
34)                     the_robot.signalJump();
35)                     break;
36)     }
37) }
```

این دو روال با توجه به کلید فشرده شده سیگنال های مربوطه را صادر می کنند که در داخل کلاس robot ، handle می شوند. (مثلاً اگر ربات در لاین یک باشد حرکت به چپ نباید مجاز باشد و یا اگر ربات روی زمین نباشد فشردن دکمه پرش باید بی تأثیر باشد).

در ادامه کدهای مربوط به کلاسهای decorative و environment را ارائه می کنیم:

```
//file decorative.h
#include <GL/glut.h>
#include "vec3f.h"

class decorative
{
public:
        decorative();
        ~decorative();
        setBody(GLuint);
        void setPos(float, float, float);
        void setRot(float, float, float);
        void setScl(float, float, float);
        void draw();

private:
        GLuint body;
        vec3f pos; //position
        vec3f rot; //rotation
        vec3f scl; //scale
};
```

```
//file decorative.cpp
decorative::decorative(){
	body = 0;
	pos.set(0, 0, 0);
	rot.set(0, 0, 0);
	scl.set(0, 0, 0);
}

void decorative::setPos(float x, float y, float z){
	pos.set(x, y, z);
}

void decorative::setRot(float x, float y, float z){
	rot.set(x, y, z);
}

void decorative::setScl(float x, float y, float z){
	scl.set(x, y, z);
}

void decorative::draw(){
	glPushMatrix();

	glTranslatef(pos.x, pos.y, pos.z);
	glRotatef(rot.x, rot.y, rot.z);
	glScalef(scl.x, scl.y, scl.z);

	glCallList(body);

	glPopMatrix();
}
```

یک environment شامل یک vector از اشیاء decorative است که اعضای آن از روی فایل متنی توضیحات ساخته می شوند:

```
#include <ifstream>
#include <vector>
#include <GL/glut.h>
#include "decorative.h"

#ifndef ENVIRONMENT_H
#define ENVIRONMENT_H

class environment
{
public:
	environment(string, GLuint*);
	~environment();
	void draw();
private:
	std::vector<decorative*> decors;
};

#endif

/*inputs a description file and an array of display list ids
	and creates decorative objecs according to file data
		and stores them in a vector*/
```

```
environment::environment(string filename, GLuint* dlist){
	ifstream fin(filename);
	decorative* decor;
	int bc; //body code
	float x, y, z;
	while((fin >> bc) != EOF){
		decor = new decorative;
		decor->setBody(dlist[bc]);
		fin >> x >> y >> z;
		decor->setPos(x, y, z);
		fin >> x >> y >> z;
		decor->setRot(x, y, z);
		fin >> x >> y >> z;
		decor->setScl(x, y, z);
		decors.push_back(decor);
	}
}

void environment::draw(){
	for(int i = 0; i < (int)decors.size(); i++)
		decors[i]->draw();
}
```

کلاس map مشابه environment است جز آنکه برای انواع مختلف موانع vector های جداگانه دارد(این موضوع برای قسمت collision detection لازم است) و دو روال اضافی update() و reset() نیز دارد که در آنها به ترتیب روال های update() و reset() موانع متحرک فراخوانی می شوند.

در آخر لازم به ذکر است که سعی شده تا ساختار برنامه به گونه ای باشد که تمام بار محاسباتی در قسمت لودینگ در ابتدای اجرای برنامه قرار گیرد و در طول بازی جز هر از چند گاهی تولید یک جفت عدد تصادفی و تغییر مختصات برخی اشیاء و البته collision detection کار دیگری انجام نگیرد تا سرعت اجرا بهینه باشد.