

Advanced Programing

09/12/2020

Francisco Jimenez Garcia

jimenezga@esat-alumni.com

INDEX

Algorhythm	2
Programming Paradigms	3
PROCEDURAL	3
OBJECT ORIENTED	3
EVENT ORIENTED	4
Implementation and Debugging	5
DB : Data Base.	6
JG_Game DB:	6
Queries and Callbacks.	8
Short Manual User.	10
Conclusion and Future Work.	14

Algorhythm

An algorithm is a finite sequence of well-defined, computer implementable instructions, typically to solve a class of problems or to perform a computation.

In this practice i use the and algorithms to determine the relative path to a file in the selected folder. Using brute force I determine the route of the file in my game folder so i don't depend on absolute routes, it still not the most efficient algorithm i could use, there are some libraries where they have optimized this proceed.

Programming Paradigms

PROCEDURAL

Procedural programming is a programming paradigm, derived from structured programming, based on the concept of the procedure call. Procedures, also known as routines, subroutines, or functions, simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program's execution, including by other procedures or itself.

OBJECT ORIENTED

Object-oriented programming (OOP) refers to a type of computer programming (software design) in which programmers define the data type of a data structure, and also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.

EVENT ORIENTED

Event-driven programming is a programming paradigm in which the flow of program execution is determined by events - for example a user action such as a mouse click, key press, or a message from the operating system or another program. An event-driven application is designed to detect events as they occur, and then deal with them using an appropriate event-handling procedure. The idea is an extension of interrupt-driven programming of the kind found in early command-line environments such as DOS, and in embedded systems (where the application is implemented as firmware). 4 My Application uses the object oriented programming. The code is structured in a mother class called GameObject from which derive most of the other classes.

Implementation and Debugging

I mostly use the IDE of Microsoft, Visual Studio to debug the project. When my app closed without apparent reason, I run it in the IDE so I can see where did that happen and why but putting breakpoints, watching the stack memory and using the console to watch the data of some variables.

Developing a game without an IDE is doable but having one opens the possibility of automate process like compilation, also gives you some help while developing since you can navigate the code easily.

Not having a IDE means that you should have to do most of this task manually, essentially losing time and making harder manage larger projects.



Having a normative when programming help me to made my code more readable for others, facilitating both the communication and understanding of it. In a company this will be even better, since you can quickly recognise parts of the code only by the way they are writed.

Resulting in a more efficient work, avoiding confusions at the time of naming variables and functions.

DB : Data Base.

JG_Game DB:

Game Object table.

Table name:		Objects	<input type="checkbox"/> WITHOUT ROWID						
	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ID	INTEGER							NULL
2	TAG	TEXT							NULL
3	ENABLE	INTEGER							NULL
4	TYPE	INTEGER							NULL
5	ORDER	INTEGER							NULL
6	PX	DOUBLE							NULL
7	PY	DOUBLE							NULL
8	ANGLE	DOUBLE							NULL
9	SCX	DOUBLE							NULL
10	SCY	DOUBLE							NULL
11	SX	DOUBLE							NULL
12	SY	DOUBLE							NULL
13	CR	INTEGER							NULL
14	CG	INTEGER							NULL
15	CB	INTEGER							NULL
16	CA	INTEGER							NULL
17	FCR	INTEGER							NULL
18	FCG	INTEGER							NULL
19	FCB	INTEGER							NULL
20	FCA	INTEGER							NULL
21	TEXT_ID	INTEGER							NULL
22	STRING	TEXT							NULL
23	FONT	TEXT							NULL
24	VX	DOUBLE							NULL
25	VY	DOUBLE							NULL
26	VALUE	INTEGER							NULL

Configuration table.

Table name: ☐ WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	TURN	INTEGER							NULL
2	SCORE1	INTEGER							NULL
3	SCORE2	INTEGER							NULL
4	TOKEN1	TEXT							NULL
5	TOKEN2	TEXT							NULL
6	PLAYER_NAME1	TEXT							NULL
7	PLAYER_NAME2	TEXT							NULL

Texture table.

Table name: ☐ WITHOUT ROWID

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1	ID	INTEGER							NULL
2	PATH	TEXT							NULL

Queries and Callbacks.

I use a simple function to open the DB before extracting or loading items in. I record a show an error in case it was a problem.

```
err_code = sqlite3_open("../data/Db/Jg_DB.db", &db);
if (err_code != SQLITE_OK) {
    fprintf(stdout, "SQL error: %s\n", err_msg);
    sqlite3_free(err_msg);
} else {
    fprintf(stdout, "Open DB successfully\n");
}

snprintf(exec, kBufferSize, "DELETE FROM Objects");
err_code = sqlite3_exec(db, exec, NULL, 0, NULL);

snprintf(exec, kBufferSize, "DELETE FROM Config");
err_code = sqlite3_exec(db, exec, NULL, 0, NULL);
```


Then before inserting any items in the DB i remove its contents securing the correct functionality of it.

```

snprintf(exec, kBufferSize, "DELETE FROM Objects");
err_code = sqlite3_exec(db, exec, NULL, 0, NULL);

snprintf(exec, kBufferSize, "DELETE FROM Config");
err_code = sqlite3_exec(db, exec, NULL, 0, NULL);

char temp[kBufferSize];

for (int i = 0; i < scene_data_.size(); ++i) {
    int enable = 0;
    if (scene_data_[i]->enable()) enable = 1;

    snprintf(exec, kBufferSize, "INSERT INTO Objects\n VALUES(%d, '%s', %d, %d,
    %d, %f, %f, %f, %f, %f, %f, %d, %d, %d, %d, %d, %d, %d, %d, '%s',
    '%s', %f, %f, %d)", scene_data_[i]->id(), scene_data_[i]->tag(), enable,
    scene_data_[i]->type(), scene_data_[i]->order(), scene_data_[i]->position().
    x, scene_data_[i]->position().y, scene_data_[i]->rotation(), scene_data_[i]
    ->scale().x, scene_data_[i]->scale().y, scene_data_[i]->size().x,
    scene_data_[i]->size().y, scene_data_[i]->color().r, scene_data_[i]->color()
    .g, scene_data_[i]->color().b, scene_data_[i]->color().a, scene_data_[i]
    ->o_color().r, scene_data_[i]->o_color().g, scene_data_[i]->o_color().b,
    scene_data_[i]->o_color().a, scene_data_[i]->texture_id(), scene_data_[i]
    ->text(), scene_data_[i]->path(), scene_data_[i]->velocity().x, scene_data_
    [i]->velocity().y, 0);
    err_code = sqlite3_exec(db, exec, NULL, 0, &err_msg);
    if (err_code != SQLITE_OK) printf("%s\n", err_msg);
}

snprintf(exec, kBufferSize, "INSERT INTO Config\n VALUES(%d, %d, %d, '%s',
'%s', '%s', '%s')", turn_, score1_, score2_, token1_, token2_, player1_name_,
player2_name_);
err_code = sqlite3_exec(db, exec, NULL, 0, &err_msg);

for (int i = 0; i < fields_.size(); i++) {
    int enable = 0;
    if (fields_[i]->enable()) enable = 1;

    snprintf(exec, kBufferSize, "INSERT INTO Objects\n VALUES(%d, '%s', %d, %d,
    %d, %f, %f, %f, %f, %f, %f, %d, %d, %d, %d, %d, %d, %d, %d, '%s',
    '%s', %f, %f, %d)", fields_[i]->id(), fields_[i]->tag(), enable, fields_[i]
    ->type(), fields_[i]->order(), fields_[i]->position().x, fields_[i]
    ->position().y, fields_[i]->rotation(), fields_[i]->scale().x, fields_[i]
    ->scale().y, fields_[i]->size().x, fields_[i]->size().y, fields_[i]->color()
    .r, fields_[i]->color().g, fields_[i]->color().b, fields_[i]->color().a,
    fields_[i]->o_color().r, fields_[i]->o_color().g, fields_[i]->o_color().b,
    fields_[i]->o_color().a, fields_[i]->texture_id(), fields_[i]->text(),
    fields_[i]->path(), fields_[i]->velocity().x, fields_[i]->velocity().y,
    fields_[i]->value());
    err_code = sqlite3_exec(db, exec, NULL, 0, &err_msg);
    if (err_code != SQLITE_OK) printf("%s\n", err_msg);
}

sqlite3_close(db);

```

Then at the time of extraction, i select the table and all its contents, passing the calling class to the callback and creating the objects while extracted. In case the object can't be created by any problem it continues with the rest.

```
sprintf(exec, "SELECT * from Objects");
err_code = sqlite3_exec(db, exec, loadCallbackObjects, this, &err_msg);

sprintf(exec, "SELECT * from Config");
err_code = sqlite3_exec(db, exec, loadCallbackConfig, this, &err_msg);
```

```
int Scene::loadCallbackObjects(void *data, int argc, char **argv,
    char **azColName) {
    Scene *temp = (Scene*)data;

    bool enable = false;
    if (std::atoi(argv[2]) == 1) enable = true;

    temp->addObject(std::atoi(argv[0]), argv[1], enable, std::atoi(argv[3]),
        std::atoi(argv[4]), std::atof(argv[5]), std::atof(argv[6]), std::atof(argv[7])
        , std::atof(argv[8]), std::atof(argv[9]), std::atof(argv[10]), std::atof(argv
        [11]), std::atoi(argv[12]), std::atoi(argv[13]), std::atoi(argv[14]),
        std::atoi(argv[15]), std::atoi(argv[16]), std::atoi(argv[17]), std::atoi(argv
        [18]), std::atoi(argv[19]), std::atoi(argv[20]), argv[21], argv[22], std::atof
        (argv[23]), std::atof(argv[24]), std::atoi(argv[25]));

    return 0;
}

int Scene::loadCallbackConfig(void *data, int argc, char **argv,
    char **azColName) {
    Scene *temp = (Scene*)data;

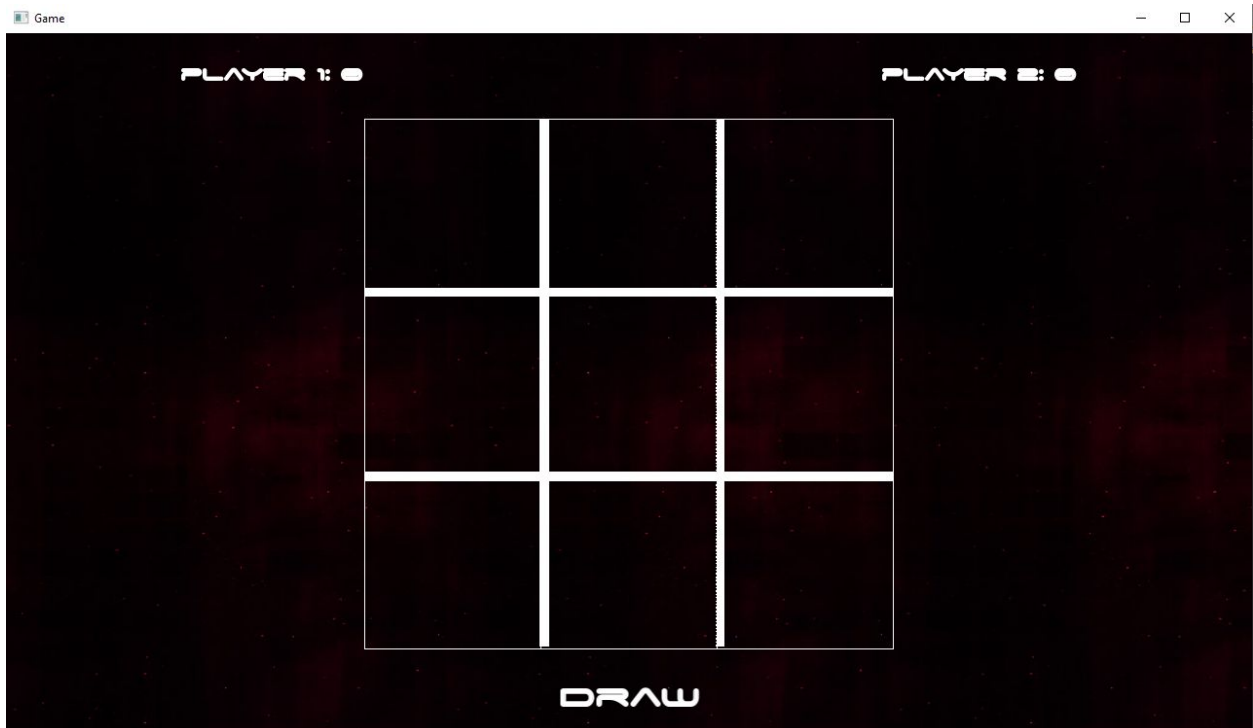
    temp->setConfig(std::atoi(argv[0]), std::atoi(argv[1]), std::atoi(argv[2]),
        argv[3], argv[4], argv[5], argv[6]);

    return 0;
}
```

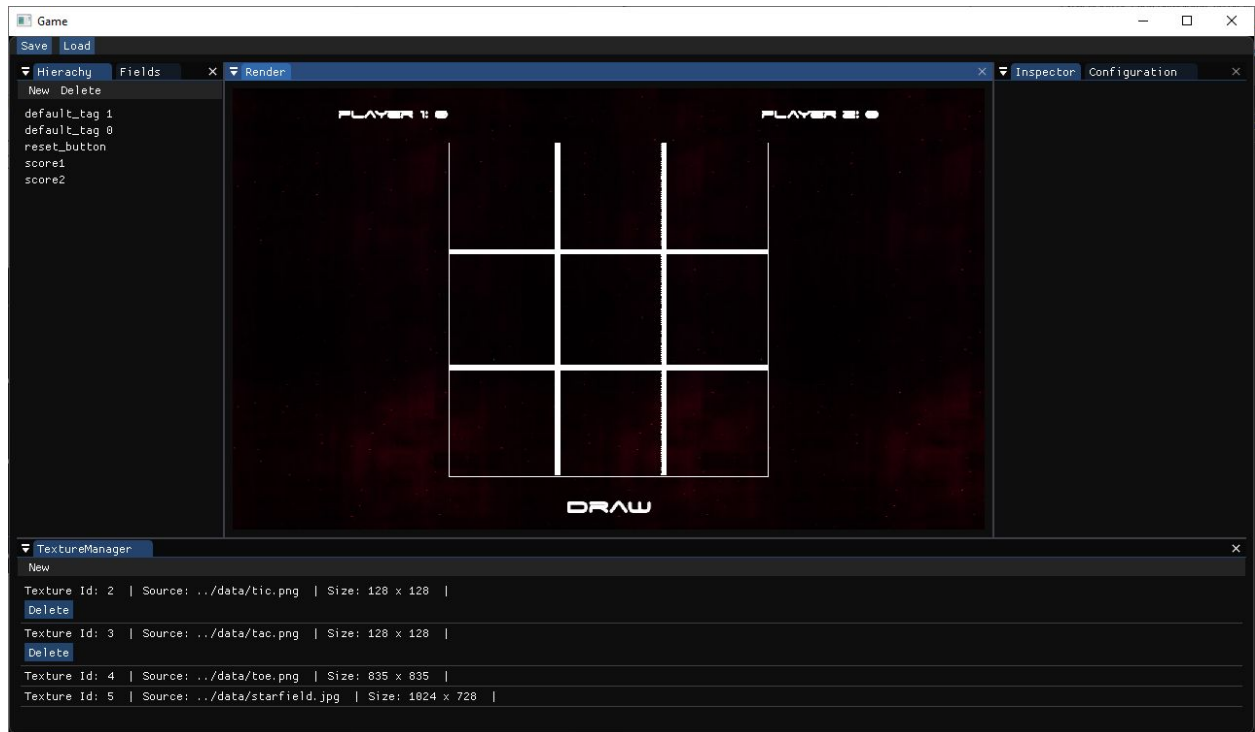
Short Manual User.

The game has to modes, editor and the actual game itself. To enter in edition mode, you must press the Tab key, it can be pressed to change back into Game mode.

GAME MODE:



EDITION MODE:



During edition mode you can navigate using the mouse and keyboard.

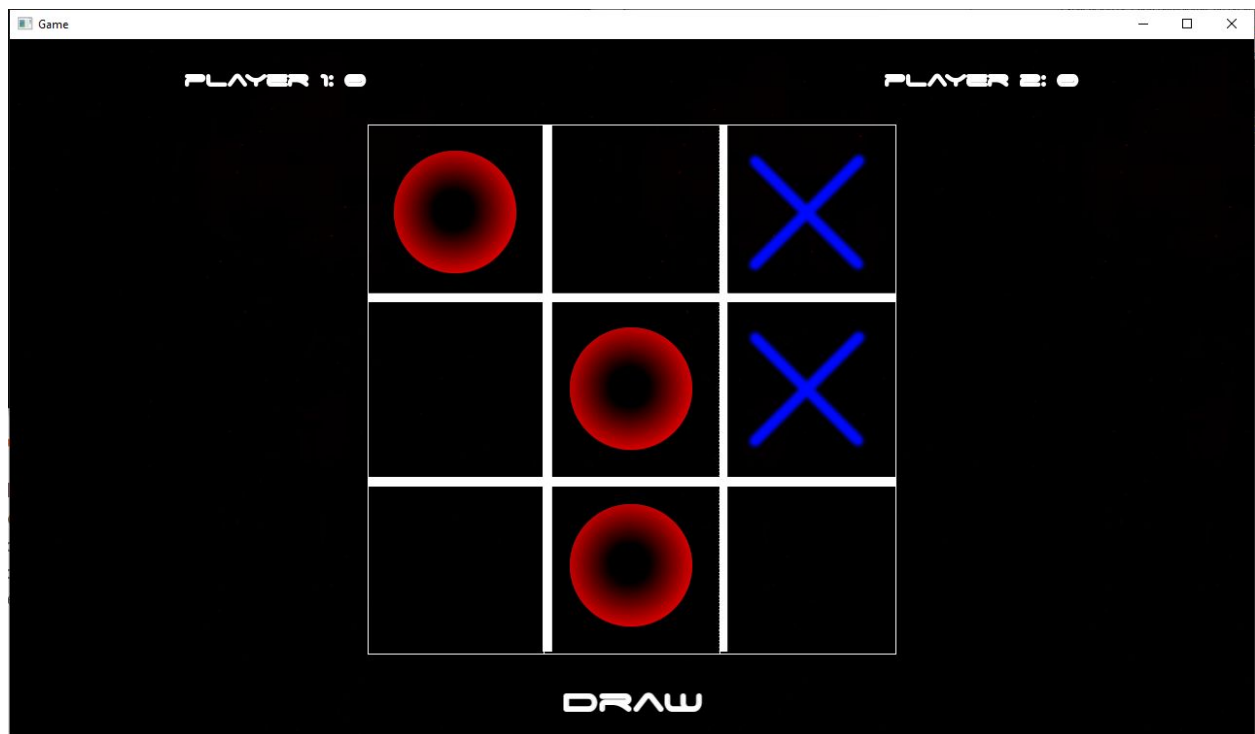
The Hierarchy window shows the basic game objects created in the scene.

The Field window shows the Fields of the game, a special class made for the game.

The Inspector lets you edit the properties of the object selected within the Hierarchy window.

The TextureManager window shows all the textures loaded and a few of their properties.

During Game mode you must click in the distinct fields of the game during altern turns until you put three of your tokens in a line. If the game didn't end and there are no more free fields in the scene, you can press the draw button to reset the fields without adding any score to either of the players.



When someone achieve the goal of the game, a point is added to the player score who made it.

Conclusion and Future Work.

The editor contains a few bugs related most of them in the Texture Manager, where some texture will not be loaded correctly. That will be one of the major points to be solve in future version of the game.

Also when loading the game from the Database, the same problem appear.

Saving and loading the game state will be another point to change, since there is a problem in the Texture manager i couldn't save it, when the state was loaded the texture disappeared but the value and the score remain making it useless to do it.

Improving the way the editor works, making it able to modify some properties using the mouse directly over the object.