ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

DOCUMENTATION

Advanced Programming
Gustavo Aranda
2019/2020
Borja Serrano García

Documentation

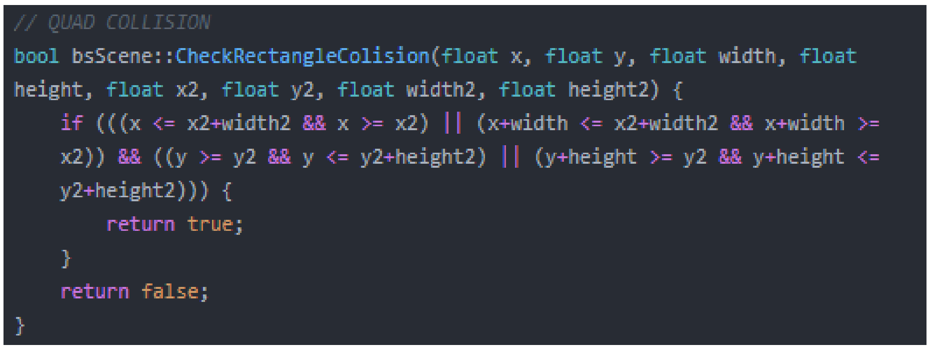
1.- Documentation.

1.1. Algorithms

An algorithm can be defined as a sequence of instructions that represent a solution model for a certain type of problem. Or as a set of instructions that, when carried out in order, lead to the solution of a problem.

In each problem the algorithm can be written and then executed in a different programming language. It is the infrastructure of any solution written later in a programming language.

In practice I use one to detect the collisions of objects with each other.



```
// QUAD COLLISION
bool bsScene::CheckRectangleColision(float x, float y, float width, float
height, float x2, float y2, float width2, float height2) {
    if (((x <= x2+width2 && x >= x2) || (x+width <= x2+width2 && x+width >=
    x2)) && ((y >= y2 && y <= y2+height2) || (y+height >= y2 && y+height <=
    y2+height2))) {
        return true;
    }
    return false;
}
```

In this case, its use is to detect collisions between squares.

The design of an algorithm for each problem is more effective than trying to solve each problem that comes out since there is a previous analysis of the situation and the possible problems that will be had later, also the approach of how to solve the problem is more effective than to put with the problem of blow. Saving time and even having to face the problem again because of a bad approach can be enormous.

To develop an application, the following steps must be followed:

- Analysis of the problem, definition and delimitation of the problem

- Design and development of the algorithm (pseudo code).

- Desktop test. Manual follow up of the steps described in the algorithm. It is done with small-scale tests to check if it works.

- The algorithm is then coded. The language is selected and developed.

- Compilation of the program using the chosen software.

- The program is executed and it is checked if the required is done.

- It is debugged in case of errors or memory leaks.

- The results obtained are evaluated.

1.2. Programming Paradigms.

A programming paradigm is a conceptual framework, a set of ideas that describes a way of understanding program construction, as it defines:

The conceptual tools that can be used to build a program (objects, relations, functions, instructions).
The valid ways to combine them.
The different programming languages provide implementations for the conceptual tools described by the paradigms.

Since a paradigm is a set of ideas, its influence is mainly seen at the moment of modeling a solution to a given problem. It is not enough to know in which language a program is built to know which conceptual framework was used at the time of construction. The paradigm has more to do with the mental process of building a program than with the resulting program.

I will describe different types:

- **Procedural**: this paradigm can be defined as "a series of obligatory steps" (hence the term imperative).

It is like a chain, like a serial system, like the alphabet, with A followed by B, then C, then D and so on. In each "step" of the system, we are in a different state of the system. It's the simplest.

- **Object Orientation:** In object orientation, simple data becomes complex (quiet, complex means it has several parts, not complicated) as it is composed of more data.

This is a data structure, that is, two or more data organized in a logical way, which represents an idea, concept or model of reality, within the program, it is what we call object.

We can define this paradigm as "the simulation of reality", one of the principles of programming, put as a priority. In this, the main thing is the objects, which have characteristics and functionalities, just as in real life.
A car has characteristics: a brand, a model, a year; a type of engine, etc. Well, the object that represents a car can have them too. In object orientation, we call these characteristics properties.

A car has functionalities too, that is, actions: Starting, moving, braking. In object orientation, we call these features methods.

So, an object is composed of properties and methods. As in all things in real life, we can classify objects. "Car" is a class of objects, the cars themselves are the materialization of this classification.

A class is a template of an object. An object has an identity, no car is the same as another, they have characteristics that make them unique, like the plates or the serial number. But all cars have serial numbers.

Classes are only the list of characteristics and functions that an object must have, without identity. Like a format without data, like a template, a mold you use to create cookies. Then, from the classes we create the objects, and as the objects are the materialization of a class, in programming, as everything is virtual, we say that an object is an instance of a class.

- Event oriented: it is a programming paradigm in which both the structure and the execution of the programs are determined by the events that occur in the system, defined by the user or caused by them.

While in sequential (or structured) programming it is the programmer who defines what the program flow will be, in event-driven programming it will be the user himself - or whatever is driving the program - who directs the program flow. Although in sequential programming there may be intervention by an agent external to the program, these interventions will occur when the programmer has determined it, and not at any time as may be the case in event-driven programming.

The creator of an event-driven schedule must define the events that will handle his schedule and the actions that will be taken when each event occurs, which is known as the event manager. The events supported will be determined by the programming language used, by the operating system and even by events created by the programmer himself.

In event-driven programming, initializations and other initial code will be carried out at the beginning of the program execution and then the program will be blocked until some event occurs. When any of the events expected by the program takes place, the program will go on to execute the code of the corresponding event manager.

To develop these paradigms it is convenient to use an IDE. An integrated development environment (IDE) is a software system for application design that combines common developer tools in a single graphical user interface (GUI). Generally, an IDE has the following characteristics:

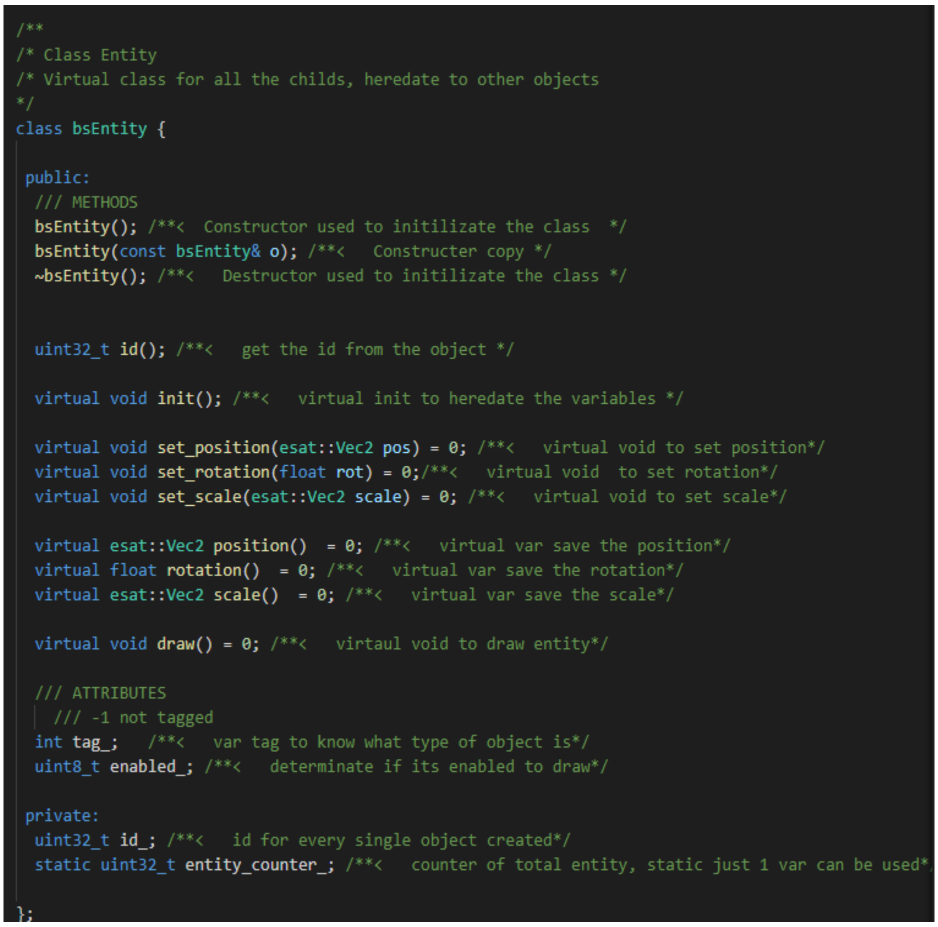
- Source code editor - A text editor that helps write software code with features such as syntax highlighting with visual cues, automatic language-specific populating, and error checking as the code is written.

- Local compilation automation - Tools that automate simple, repeatable tasks as part of creating a local compilation of software for use by the developer, such as compiling computer source code into binary code, packaging the binary code, and running automated tests.
Debugger: a program used to test other programs and show the location of an error in the original code in a graphical way.

IDE's allow developers to quickly start programming new applications, since they do not need to manually set up and integrate various tools as part of the setup process. Nor do they need to spend hours learning to use different tools separately, because they are all represented in the same workspace. This is very useful when bringing in new developers, because they can rely on an IDE to catch up with standard team workflows and tools. In fact, most IDE features are designed to save time, such as intelligent populating and automated code generation, which eliminates the need to write entire character sequences.

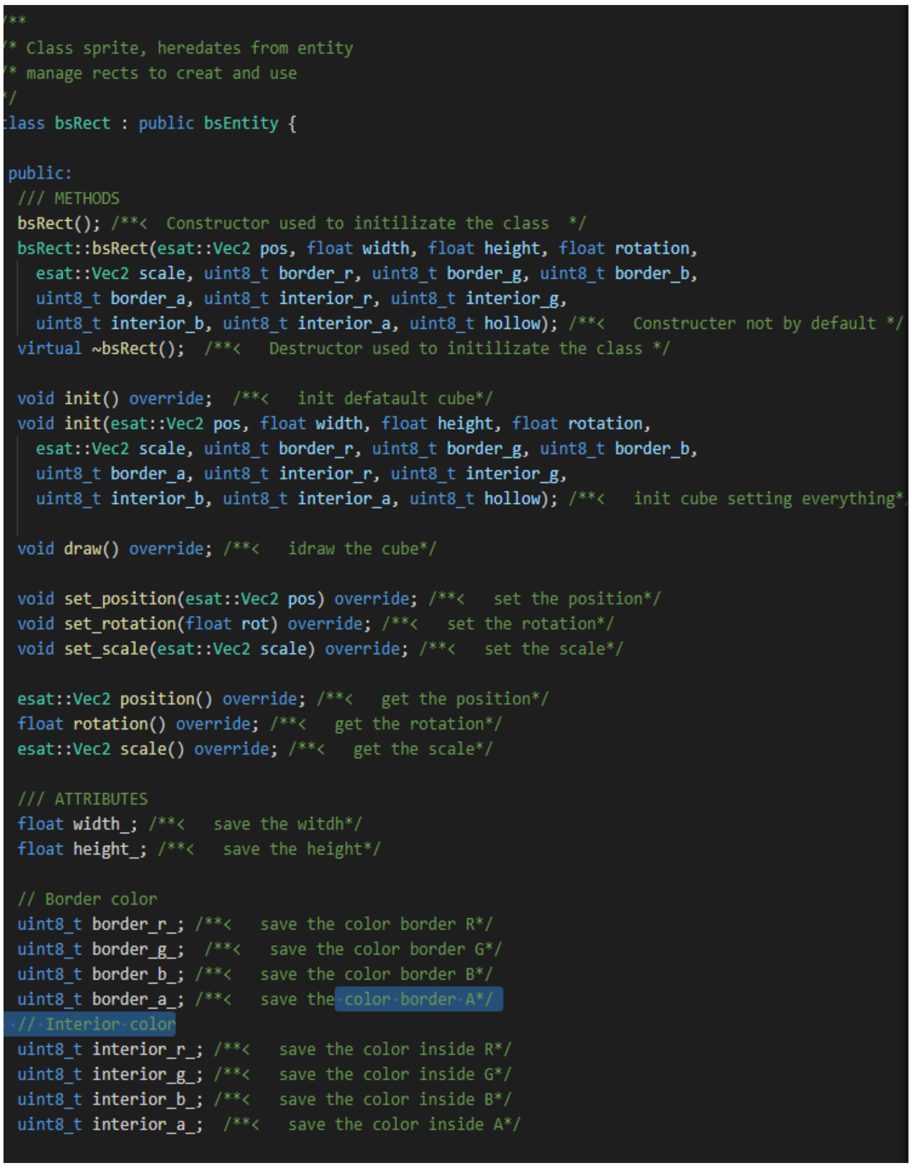
Other common IDE features are designed to help developers organize their workflow and troubleshoot problems. The IDE analyzes the code as it is being written, so failures caused by human error are identified in real time. Because there is a single GUI that represents all the tools, developers can execute tasks without having to switch from one application to another. Syntax highlighting is also common in most IDE's, and uses visual cues to distinguish grammar in the text editor. In addition, some IDE's include object and class browsers, as well as class hierarchy diagrams for certain languages.

In my program I have developed the Object Oriented Paradigm and it will be explained.



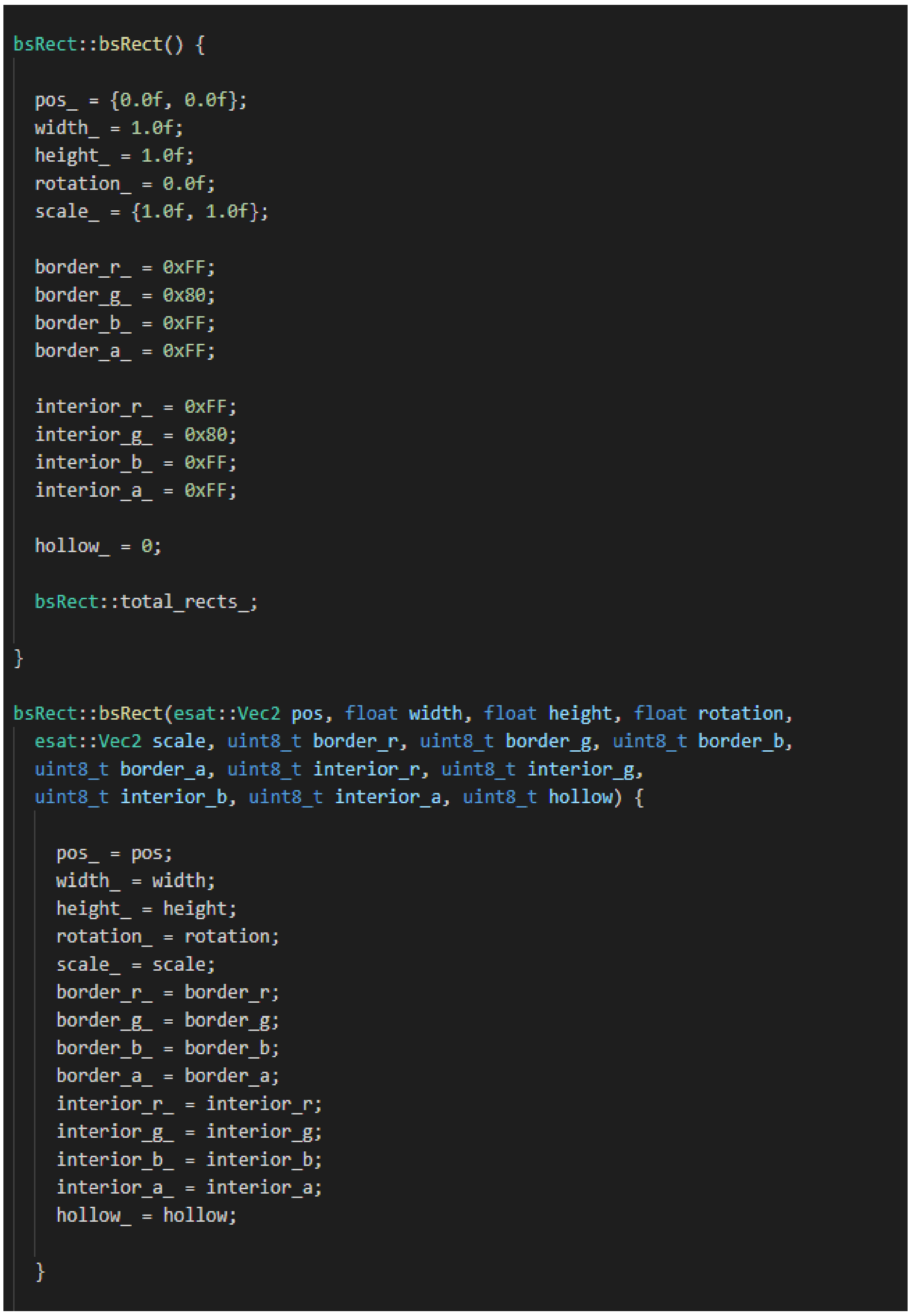
```
/**
/* Class Entity
/* Virtual class for all the childs, heredate to other objects
*/
class bsEntity {

 public:
  /// METHODS
  bsEntity(); /**<  Constructor used to initilizate the class  */
  bsEntity(const bsEntity& o); /**<   Constructer copy */
  ~bsEntity(); /**<   Destructor used to initilizate the class */


  uint32_t id(); /**<   get the id from the object */

  virtual void init(); /**<   virtual init to heredate the variables */

  virtual void set_position(esat::Vec2 pos) = 0; /**<   virtual void to set position*/
  virtual void set_rotation(float rot) = 0;/**<   virtual void  to set rotation*/
  virtual void set_scale(esat::Vec2 scale) = 0; /**<   virtual void to set scale*/

  virtual esat::Vec2 position()  = 0; /**<   virtual var save the position*/
  virtual float rotation()  = 0; /**<   virtual var save the rotation*/
  virtual esat::Vec2 scale()  = 0; /**<   virtual var save the scale*/

  virtual void draw() = 0; /**<   virtaul void to draw entity*/

  /// ATTRIBUTES
    /// -1 not tagged
  int tag_;   /**<   var tag to know what type of object is*/
  uint8_t enabled_; /**<   determinate if its enabled to draw*/

 private:
  uint32_t id_; /**<   id for every single object created*/
  static uint32_t entity_counter_; /**<   counter of total entity, static just 1 var can be used*

};
```

Entity class from which you will inherit the following, uses functions that your children will receive, uses basic variables of each object such as position, scale, rotation. It also saves if it is active and with each object a unit is added to ID to know all the objects that inherit from here.



```
/**
* Class sprite, heredates from entity
* manage rects to creat and use
*/
lass bsRect : public bsEntity {

public:
 /// METHODS
 bsRect(); /**<  Constructor used to initilizate the class  */
 bsRect::bsRect(esat::Vec2 pos, float width, float height, float rotation,
   esat::Vec2 scale, uint8_t border_r, uint8_t border_g, uint8_t border_b,
   uint8_t border_a, uint8_t interior_r, uint8_t interior_g,
   uint8_t interior_b, uint8_t interior_a, uint8_t hollow); /**<   Constructer not by default */
 virtual ~bsRect();  /**<   Destructor used to initilizate the class */

 void init() override;  /**<   init defatault cube*/
 void init(esat::Vec2 pos, float width, float height, float rotation,
   esat::Vec2 scale, uint8_t border_r, uint8_t border_g, uint8_t border_b,
   uint8_t border_a, uint8_t interior_r, uint8_t interior_g,
   uint8_t interior_b, uint8_t interior_a, uint8_t hollow); /**<   init cube setting everything*

 void draw() override; /**<   idraw the cube*/

 void set_position(esat::Vec2 pos) override; /**<   set the position*/
 void set_rotation(float rot) override; /**<   set the rotation*/
 void set_scale(esat::Vec2 scale) override; /**<   set the scale*/

 esat::Vec2 position() override; /**<   get the position*/
 float rotation() override; /**<   get the rotation*/
 esat::Vec2 scale() override; /**<   get the scale*/

 /// ATTRIBUTES
 float width_; /**<   save the witdh*/
 float height_; /**<   save the height*/

 // Border color
 uint8_t border_r_; /**<   save the color border R*/
 uint8_t border_g_;  /**<   save the color border G*/
 uint8_t border_b_; /**<   save the color border B*/
 uint8_t border_a_; /**<   save the color border A*/
 // Interior color
 uint8_t interior_r_; /**<   save the color inside R*/
 uint8_t interior_g_; /**<   save the color inside G*/
 uint8_t interior_b_; /**<   save the color inside B*/
 uint8_t interior_a_;  /**<   save the color inside A*/
```

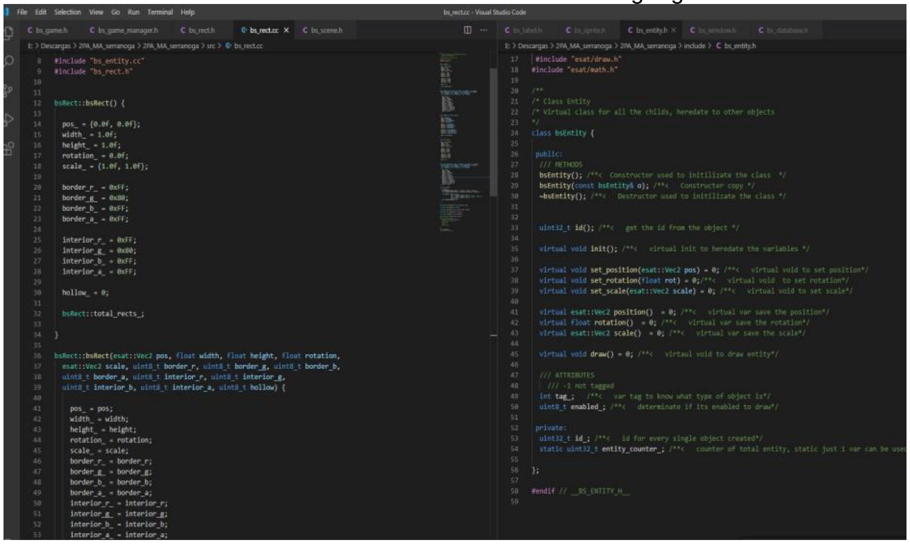
Rect class that inherits from Entity and also uses its own data.



```cpp
bsRect::bsRect() {

  pos_ = {0.0f, 0.0f};
  width_ = 1.0f;
  height_ = 1.0f;
  rotation_ = 0.0f;
  scale_ = {1.0f, 1.0f};

  border_r_ = 0xFF;
  border_g_ = 0x80;
  border_b_ = 0xFF;
  border_a_ = 0xFF;

  interior_r_ = 0xFF;
  interior_g_ = 0x80;
  interior_b_ = 0xFF;
  interior_a_ = 0xFF;

  hollow_ = 0;

  bsRect::total_rects_;

}

bsRect::bsRect(esat::Vec2 pos, float width, float height, float rotation,
  esat::Vec2 scale, uint8_t border_r, uint8_t border_g, uint8_t border_b,
  uint8_t border_a, uint8_t interior_r, uint8_t interior_g,
  uint8_t interior_b, uint8_t interior_a, uint8_t hollow) {

    pos_ = pos;
    width_ = width;
    height_ = height;
    rotation_ = rotation;
    scale_ = scale;
    border_r_ = border_r;
    border_g_ = border_g;
    border_b_ = border_b;
    border_a_ = border_a;
    interior_r_ = interior_r;
    interior_g_ = interior_g;
    interior_b_ = interior_b;
    interior_a_ = interior_a;
    hollow_ = hollow;

  }
```

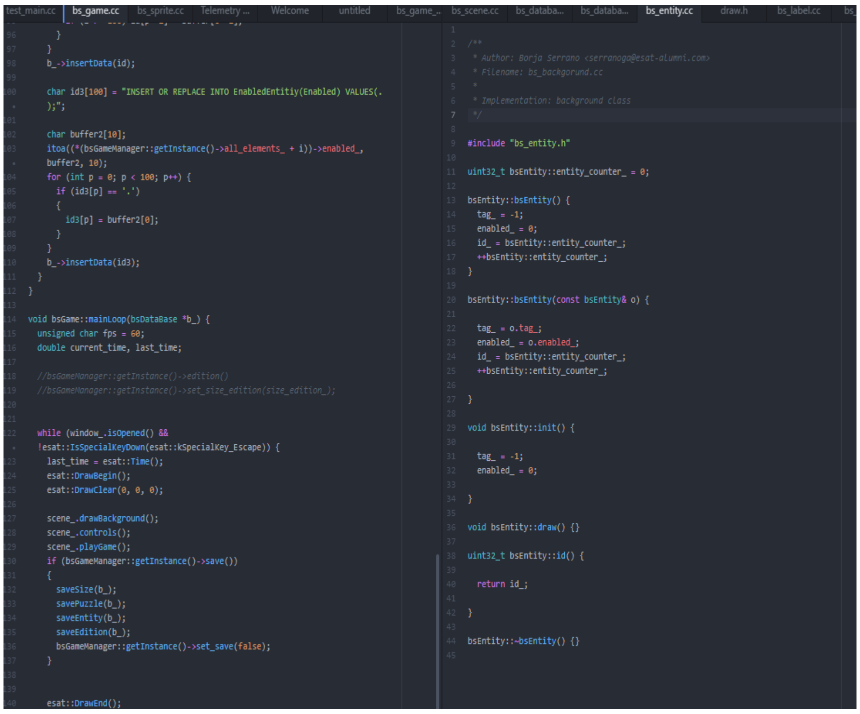
1.3. Implementation and Debugging.

In this case, Visual Studio and Atom have been used as the IDE. They enable the sharing of tools and facilitate the creation of solutions in various languages.


File Edit Selection View Go Run Terminal Help
bs_rect.cc - Visual Studio Code
bs_game.h
bs_game_manager.h
bs_rect.h
bs_rect.cc
bs_scene.h
bs_label.h
bs_sprite.h
bs_entity.h
bs_window.h
bs_database.h
E: > Descargas > 2PA_MA_serranoga > 2PA_MA_serranoga > src > bs_rect.cc
E: > Descargas > 2PA_MA_serranoga > 2PA_MA_serranoga > include > bs_entity.h

```cpp
#include "bs_entity.cc"
#include "bs_rect.h"


bsRect::bsRect() {

  pos_ = {0.0f, 0.0f};
  width_ = 1.0f;
  height_ = 1.0f;
  rotation_ = 0.0f;
  scale_ = {1.0f, 1.0f};

  border_r_ = 0xFF;
  border_g_ = 0x80;
  border_b_ = 0xFF;
  border_a_ = 0xFF;

  interior_r_ = 0xFF;
  interior_g_ = 0x80;
  interior_b_ = 0xFF;
  interior_a_ = 0xFF;

  hollow_ = 0;

  bsRect::total_rects_;

}

bsRect::bsRect(esat::Vec2 pos, float width, float height, float rotation,
  esat::Vec2 scale, uint8_t border_r, uint8_t border_g, uint8_t border_b,
  uint8_t border_a, uint8_t interior_r, uint8_t interior_g,
  uint8_t interior_b, uint8_t interior_a, uint8_t hollow) {

    pos_ = pos;
    width_ = width;
    height_ = height;
    rotation_ = rotation;
    scale_ = scale;
    border_r_ = border_r;
    border_g_ = border_g;
    border_b_ = border_b;
    border_a_ = border_a;
    interior_r_ = interior_r;
    interior_g_ = interior_g;
    interior_b_ = interior_b;
    interior_a_ = interior_a;
```

```cpp
 #include "esat/draw.h"
#include "esat/math.h"

/**
/* Class Entity
/* Virtual class for all the childs, heredate to other objects
*/
class bsEntity {

 public:
  /// METHODS
  bsEntity(); /**<  Constructor used to initilizate the class  */
  bsEntity(const bsEntity& o); /**<   Constructer copy */
  ~bsEntity(); /**<   Destructor used to initilizate the class */


  uint32_t id(); /**<   get the id from the object */

  virtual void init(); /**<   virtual init to heredate the variables */

  virtual void set_position(esat::Vec2 pos) = 0; /**<   virtual void to set position*/
  virtual void set_rotation(float rot) = 0;/**<   virtual void  to set rotation*/
  virtual void set_scale(esat::Vec2 scale) = 0; /**<   virtual void to set scale*/

  virtual esat::Vec2 position()  = 0; /**<   virtual var save the position*/
  virtual float rotation()  = 0; /**<   virtual var save the rotation*/
  virtual esat::Vec2 scale()  = 0; /**<   virtual var save the scale*/

  virtual void draw() = 0; /**<   virtaul void to draw entity*/

  /// ATTRIBUTES
    /// -1 not tagged
  int tag_;   /**<   var tag to know what type of object is*/
  uint8_t enabled_; /**<   determinate if its enabled to draw*/

 private:
  uint32_t id_; /**<   id for every single object created*/
  static uint32_t entity_counter_; /**<   counter of total entity, static just 1 var can be used

};

#endif // __BS_ENTITY_H__
```



```cpp
      }
    }
    b_->insertData(id);

    char id3[100] = "INSERT OR REPLACE INTO EnabledEntitiy(Enabled) VALUES(.
    );";

    char buffer2[10];
    itoa((*(bsGameManager::getInstance()->all_elements_ + i))->enabled_,
    buffer2, 10);
    for (int p = 0; p < 100; p++) {
      if (id3[p] == '.')
      {
        id3[p] = buffer2[0];
      }
    }
    b_->insertData(id3);
  }
}

void bsGame::mainLoop(bsDataBase *b_) {
  unsigned char fps = 60;
  double current_time, last_time;

  //bsGameManager::getInstance()->edition()
  //bsGameManager::getInstance()->set_size_edition(size_edition_);


  while (window_.isOpened() &&
  !esat::IsSpecialKeyDown(esat::kSpecialKey_Escape)) {
    last_time = esat::Time();
    esat::DrawBegin();
    esat::DrawClear(0, 0, 0);

    scene_.drawBackground();
    scene_.controls();
    scene_.playGame();
    if (bsGameManager::getInstance()->save())
    {
      saveSize(b_);
      savePuzzle(b_);
      saveEntity(b_);
      saveEdition(b_);
      bsGameManager::getInstance()->set_save(false);
    }


    esat::DrawEnd();
```

```cpp

/**
 * Author: Borja Serrano <serranoga@esat-alumni.com>
 * Filename: bs_backgorund.cc
 *
 * Implementation: background class
 */

#include "bs_entity.h"

uint32_t bsEntity::entity_counter_ = 0;

bsEntity::bsEntity() {
  tag_ = -1;
  enabled_ = 0;
  id_ = bsEntity::entity_counter_;
  ++bsEntity::entity_counter_;
}

bsEntity::bsEntity(const bsEntity& o) {

  tag_ = o.tag_;
  enabled_ = o.enabled_;
  id_ = bsEntity::entity_counter_;
  ++bsEntity::entity_counter_;

}

void bsEntity::init() {

  tag_ = -1;
  enabled_ = 0;

}

void bsEntity::draw() {}

uint32_t bsEntity::id() {

  return id_;

}

bsEntity::~bsEntity() {}

```

These have been chosen because of their flexibility.

Preparing the code correctly to run it in a secure environment requires a ritual of its own. Operating systems already have pre-installed text editors, but their capabilities are very limited. It is important to be clear that choosing the right editor is crucial when programming, as well as that not all editors are suitable for all levels.

The differences between editors and IDEs when developing a game will be explained below.

Text editors come standard with syntax highlighting for virtually any programming language you can work with. This may seem trivial when you're just starting out, but once you know a language well and are clear about what the code written for it should look like, it's critical for the code to be read and interpreted at a glance. They have an ecosystem of packages to extend their capabilities. Both have a wide range of solutions, from improved word processing, to HTML templates, to GitHub integration, to code quality control tools and much more.

They are recommended if you already have some experience working with a particular language, since you will need to install some packages to fine-tune your skills.

Integrated Development Environments or IDEs.

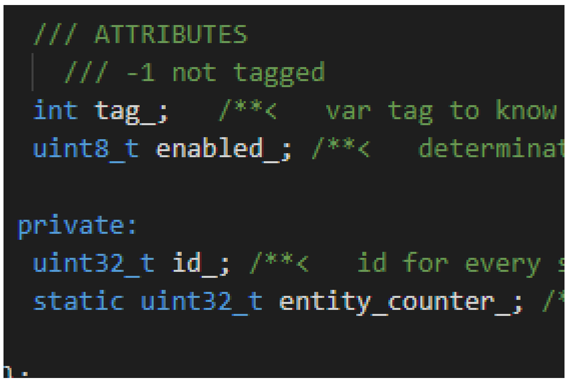
An IDE is a very powerful tool. The fundamental difference with text editors like Atom is that its power can overwhelm the user. Many offer extensions for new programming languages and frameworks that create new IDEs within an IDE.

They also have built-in sentence correctors that detect misspelled words and debuggers that point out errors in the code, which is very good for debugging and efficiency. Debugging is essential when dealing with long and complex programs. Thanks to this feature you can see the code in action while it is running, allowing you to take much more accurate measurements of the errors instead of looking at a bunch of lines on a screen waiting for the solution to magically appear.

In my case, the debugging has been very good as you can break in, see the memory in real time and change variables in real time. The syntax correction also helps, even the intelligence to fill in content automatically.

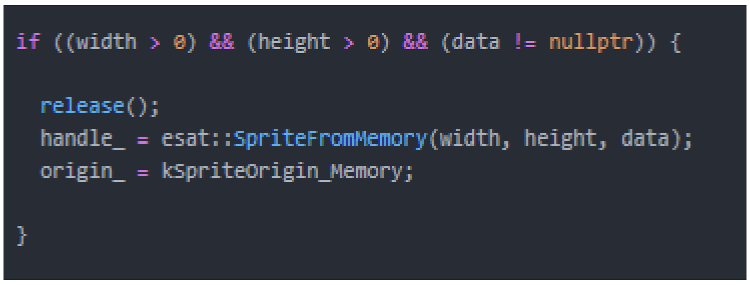
For this practice, we followed a programming rule of the university in which it is written in a certain way. Classes, class functions.

I show an example of the code:



```
  /// ATTRIBUTES
    /// -1 not tagged
  int tag_;    /**<   var tag to know
  uint8_t enabled_; /**<   determinat

private:
  uint32_t id_; /**<   id for every s
  static uint32_t entity_counter_; /*
};
```

The variables that belong to the class are written in lowercase and ending with _. The data type uses 1 space and everything else uses 2.



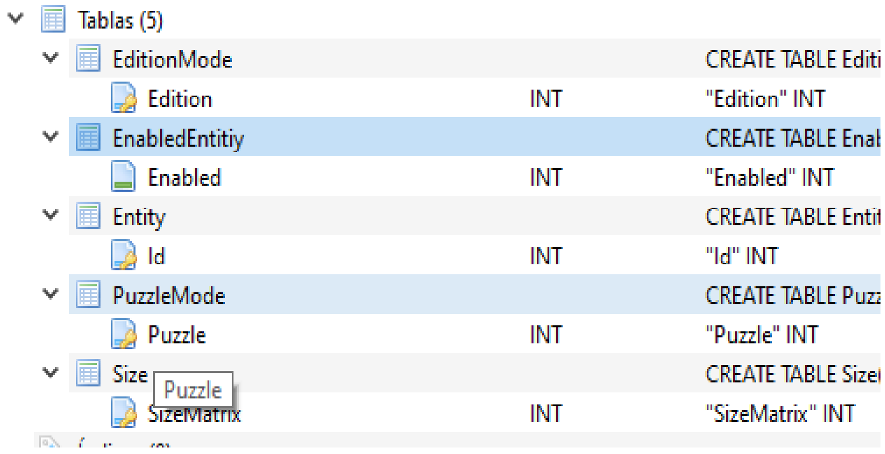
```
if ((width > 0) && (height > 0) && (data != nullptr)) {

  release();
  handle_ = esat::SpriteFromMemory(width, height, data);
  origin_ = kSpriteOrigin_Memory;

}
```

The "ifs" have a certain form in which space is required between the brackets and the first key.

It is important that all people working on a project do so using the same rules so that everything is visible and as understandable as possible and in case someone outside your code has to work on it. The more readable it is and the less time is wasted, the more efficient and the higher the performance of the company.

1.4. Database.

A database has been used in the project in order to save and be able to store and load data that are necessary in the practice.



Tablas (5)		
EditionMode		CREATE TABLE Editi
Edition	INT	"Edition" INT
EnabledEntitiy		CREATE TABLE Enal
Enabled	INT	"Enabled" INT
Entity		CREATE TABLE Entit
Id	INT	"Id" INT
PuzzleMode		CREATE TABLE Puzz
Puzzle	INT	"Puzzle" INT
Size		CREATE TABLE Size
SizeMatrix	INT	"SizeMatrix" INT

There are 5 tables that are used for different things.

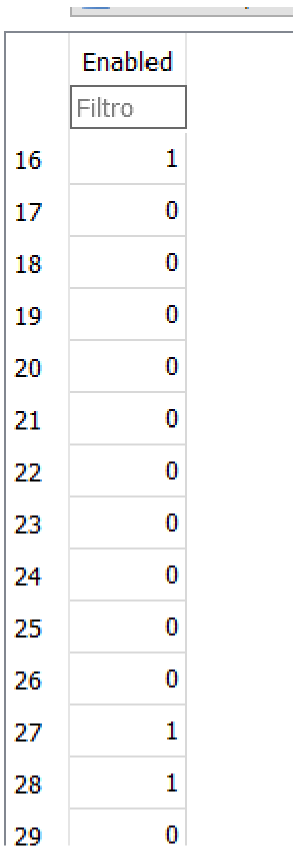
- EditionMode: saves and determines if the program is in edit mode or the game logic is working.

- EnabledEntity: saves and loads each object of the program if it is active or not. It is used together with Entity that stores the ID of each object.

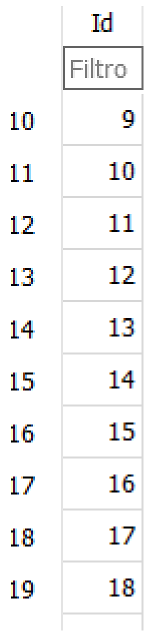
- PuzzleMode: stores if the game is in puzzle mode or in normal mode.

- SizeMatrix: saves and loads the size of the game's matrix.

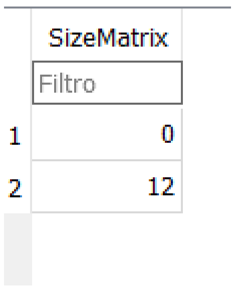
You can check if the data is correctly loaded in the code because it returns errors if it is the case or by accessing the database and checking if the data has been correctly loaded.



	Enabled
	Filtro
16	1
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	1
28	1
29	0



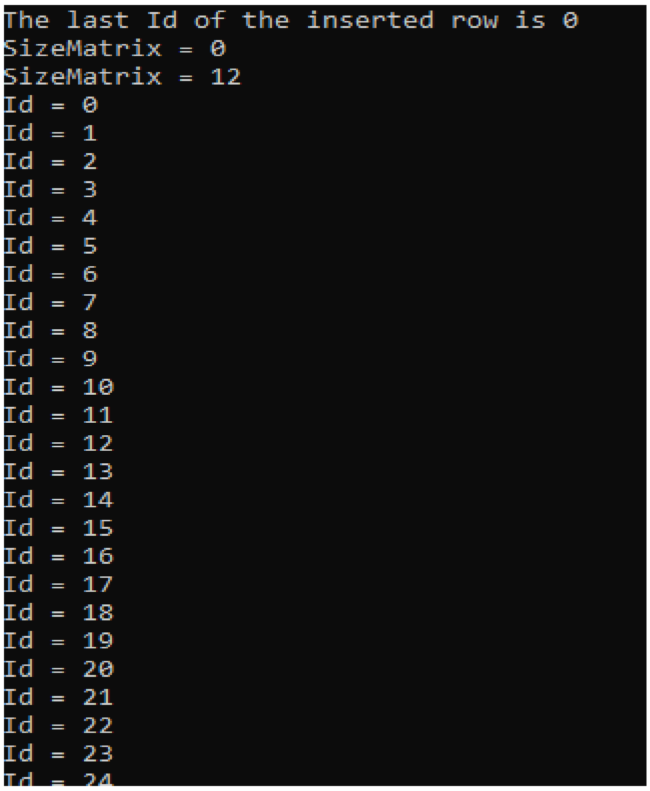
	Id
	Filtro
10	9
11	10
12	11
13	12
14	13
15	14
16	15
17	16
18	17
19	18



	SizeMatrix
	Filtro
1	0
2	12

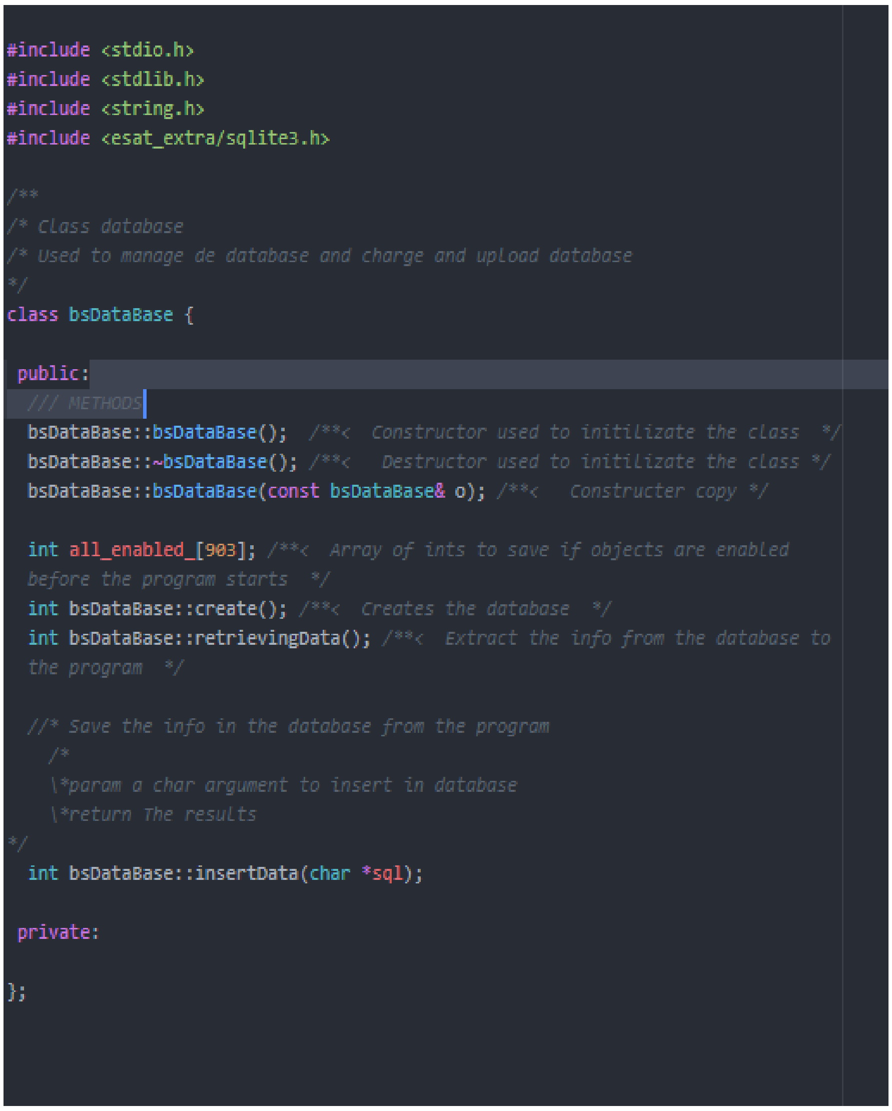
In my practice, what has been developed is in line with requirements and is effective but could be improved. The loading process is fast and is accessed instantly but the saving process is quite slow without returning errors. You save all the objects every time you want and the cost can be there. There are 5 tables with 2 of them with more than 900 elements. Add that it is not the best design of database but it works and it fulfills the required.

Each data of each table of the database is checked when the program is executed.



```
The last Id of the inserted row is 0
SizeMatrix = 0
SizeMatrix = 12
Id = 0
Id = 1
Id = 2
Id = 3
Id = 4
Id = 5
Id = 6
Id = 7
Id = 8
Id = 9
Id = 10
Id = 11
Id = 12
Id = 13
Id = 14
Id = 15
Id = 16
Id = 17
Id = 18
Id = 19
Id = 20
Id = 21
Id = 22
Id = 23
Id = 24
```

The operation of the database will be explained technically. To begin with, you create a database type that will take care of everything you need.

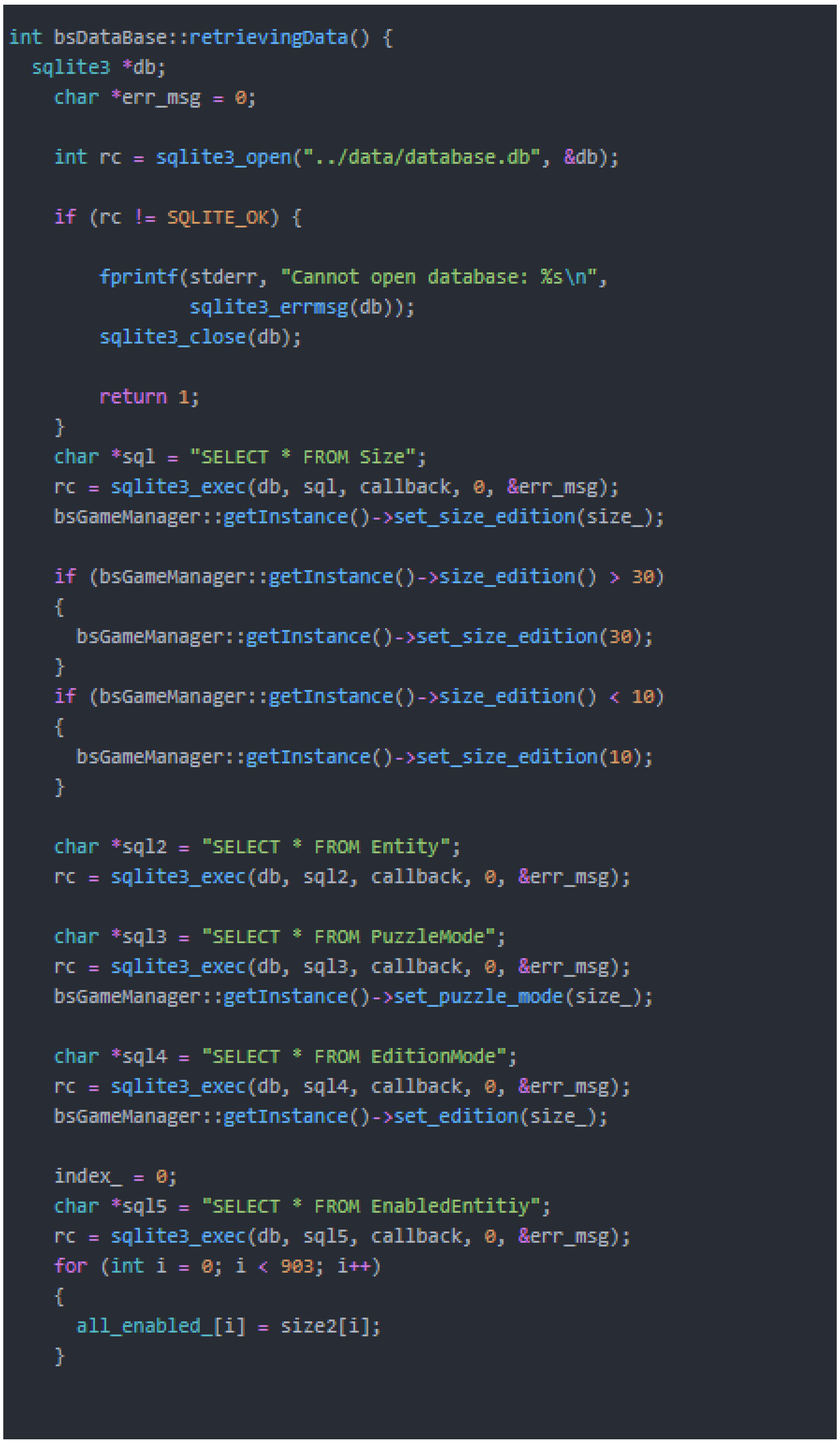


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <esat_extra/sqlite3.h>

/**
/* Class database
/* Used to manage de database and charge and upload database
*/
class bsDataBase {

 public:
  /// METHODS
  bsDataBase::bsDataBase();  /**<  Constructor used to initilizate the class  */
  bsDataBase::~bsDataBase(); /**<   Destructor used to initilizate the class */
  bsDataBase::bsDataBase(const bsDataBase& o); /**<   Constructer copy */

  int all_enabled_[903]; /**<  Array of ints to save if objects are enabled
  before the program starts  */
  int bsDataBase::create(); /**<  Creates the database  */
  int bsDataBase::retrievingData(); /**<  Extract the info from the database to
  the program  */

  //* Save the info in the database from the program
    /*
    \*param a char argument to insert in database
    \*return The results
*/
  int bsDataBase::insertData(char *sql);

 private:

};
```

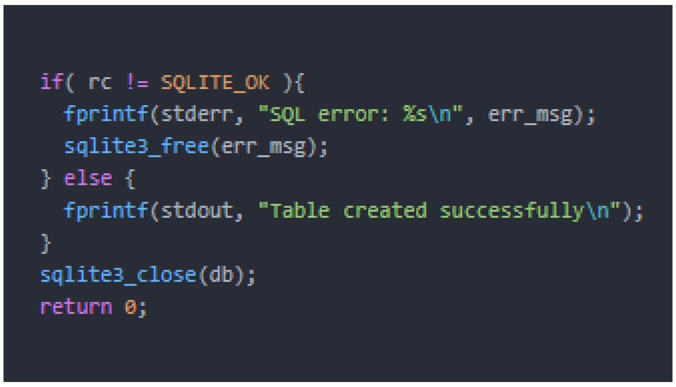
To create the table, you check if the database exists and you create and access or create the 5 necessary tables if the database did not exist.

An error is returned if something unexpected happens.

To extract the data from the database as all are extracted at once and only once, all the objects are created from the beginning and then they are not added or destroyed in real time so there is no need to make different loads.



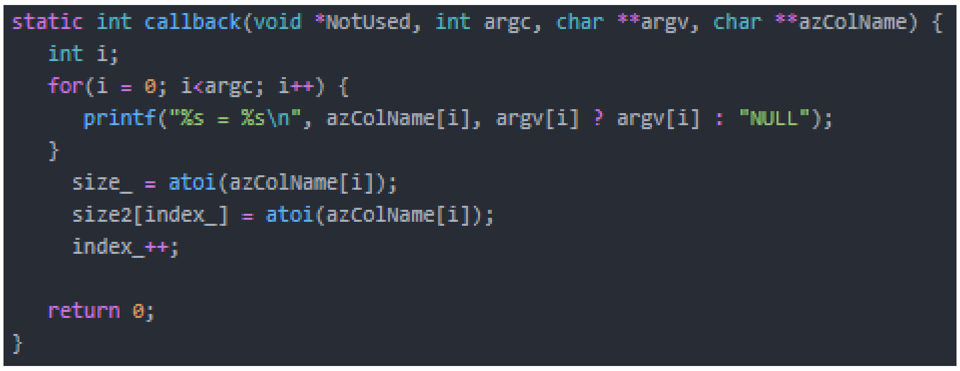
```
int bsDataBase::retrievingData() {
  sqlite3 *db;
    char *err_msg = 0;

    int rc = sqlite3_open("../data/database.db", &db);

    if (rc != SQLITE_OK) {

        fprintf(stderr, "Cannot open database: %s\n",
                sqlite3_errmsg(db));
        sqlite3_close(db);

        return 1;
    }
    char *sql = "SELECT * FROM Size";
    rc = sqlite3_exec(db, sql, callback, 0, &err_msg);
    bsGameManager::getInstance()->set_size_edition(size_);

    if (bsGameManager::getInstance()->size_edition() > 30)
    {
      bsGameManager::getInstance()->set_size_edition(30);
    }
    if (bsGameManager::getInstance()->size_edition() < 10)
    {
      bsGameManager::getInstance()->set_size_edition(10);
    }

    char *sql2 = "SELECT * FROM Entity";
    rc = sqlite3_exec(db, sql2, callback, 0, &err_msg);

    char *sql3 = "SELECT * FROM PuzzleMode";
    rc = sqlite3_exec(db, sql3, callback, 0, &err_msg);
    bsGameManager::getInstance()->set_puzzle_mode(size_);

    char *sql4 = "SELECT * FROM EditionMode";
    rc = sqlite3_exec(db, sql4, callback, 0, &err_msg);
    bsGameManager::getInstance()->set_edition(size_);

    index_ = 0;
    char *sql5 = "SELECT * FROM EnabledEntitiy";
    rc = sqlite3_exec(db, sql5, callback, 0, &err_msg);
    for (int i = 0; i < 903; i++)
    {
      all_enabled_[i] = size2[i];
    }
```



```
if( rc != SQLITE_OK ){
  fprintf(stderr, "SQL error: %s\n", err_msg);
  sqlite3_free(err_msg);
} else {
  fprintf(stdout, "Table created successfully\n");
}
sqlite3_close(db);
return 0;
```

All the tables are selected and the data is loaded. In the case of "Enabled" it is saved in an array of ints and then set to each object.

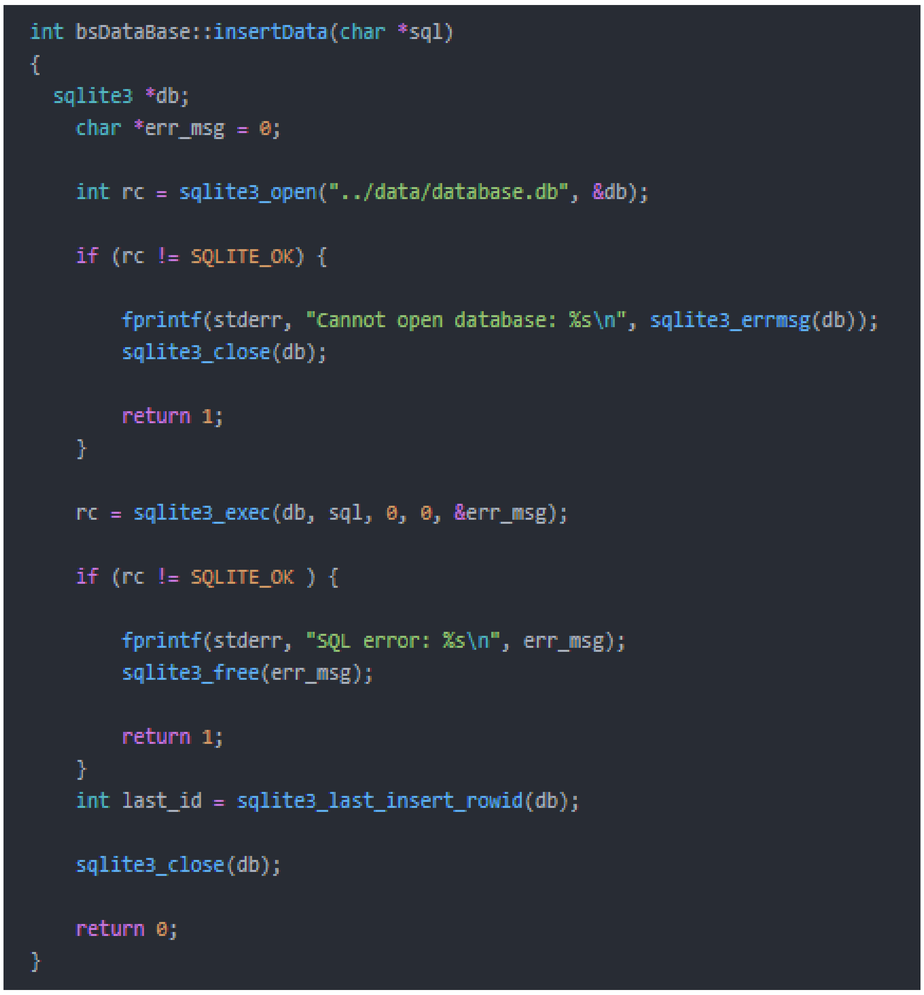
The callback function is called and it is used to go through the whole table.



```
static int callback(void *NotUsed, int argc, char **argv, char **azColName) {
   int i;
   for(i = 0; i<argc; i++) {
      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
   }
     size_ = atoi(azColName[i]);
     size2[index_] = atoi(azColName[i]);
     index_++;

   return 0;
}
```

It is a static function and saves the data in a pointer from pointer to char and becomes an int in order to use it in the program.

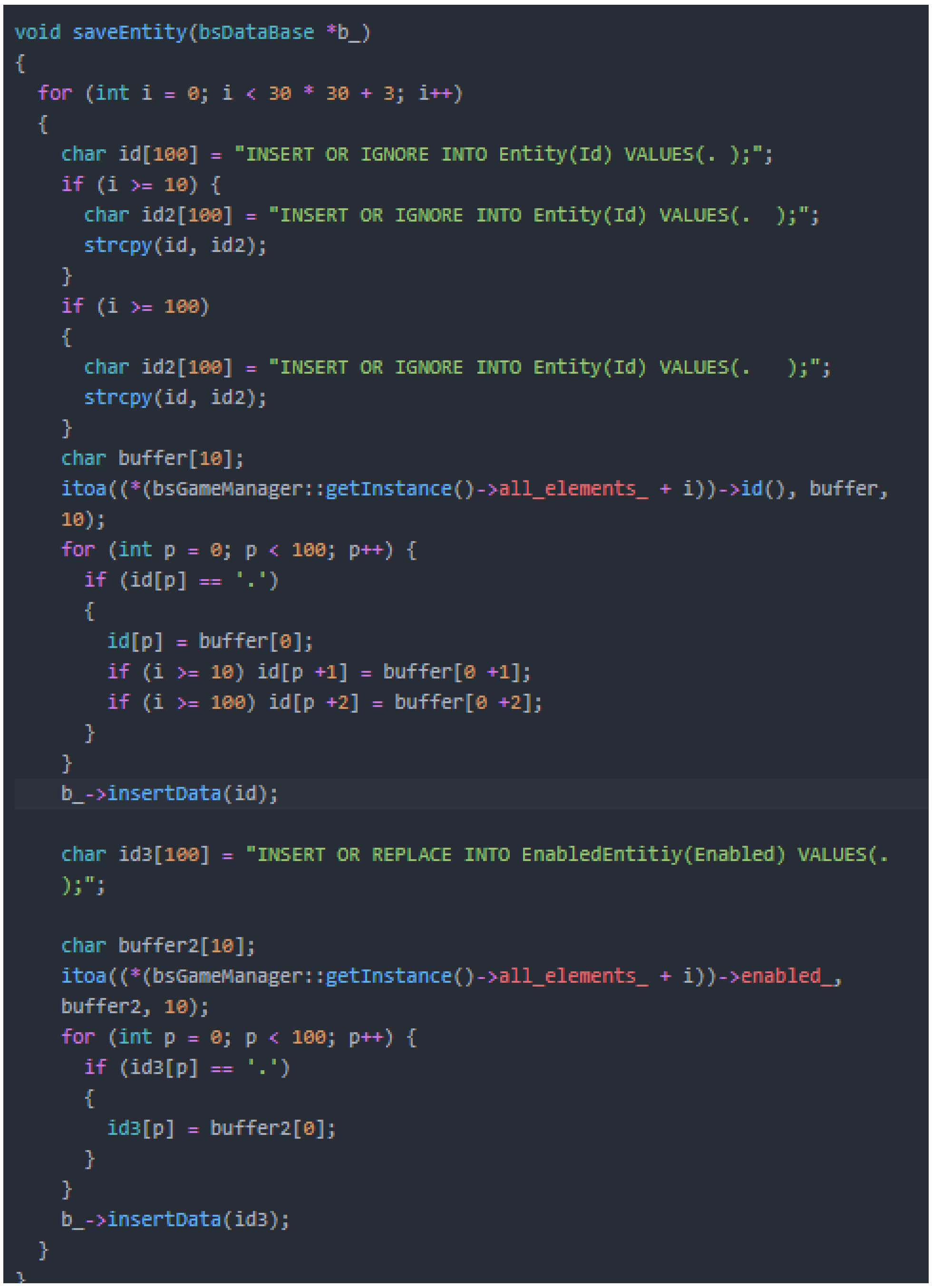
The following function is used to insert or update new data in the database.



```
int bsDataBase::insertData(char *sql)
{
  sqlite3 *db;
    char *err_msg = 0;

    int rc = sqlite3_open("../data/database.db", &db);

    if (rc != SQLITE_OK) {

        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);

        return 1;
    }

    rc = sqlite3_exec(db, sql, 0, 0, &err_msg);

    if (rc != SQLITE_OK ) {

        fprintf(stderr, "SQL error: %s\n", err_msg);
        sqlite3_free(err_msg);

        return 1;
    }
    int last_id = sqlite3_last_insert_rowid(db);

    sqlite3_close(db);

    return 0;
}
```

In which a text is passed which is the text to be saved in the database.

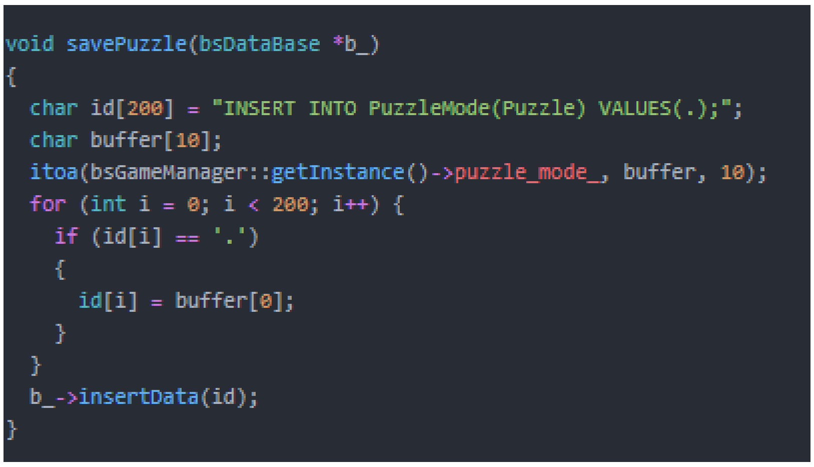
To save Id and Enabled the same function is used to change the text that is passed to this function.



```
void saveEntity(bsDataBase *b_)
{
  for (int i = 0; i < 30 * 30 + 3; i++)
  {
    char id[100] = "INSERT OR IGNORE INTO Entity(Id) VALUES(. );";
    if (i >= 10) {
      char id2[100] = "INSERT OR IGNORE INTO Entity(Id) VALUES(.  );";
      strcpy(id, id2);
    }
    if (i >= 100)
    {
      char id2[100] = "INSERT OR IGNORE INTO Entity(Id) VALUES(.   );";
      strcpy(id, id2);
    }
    char buffer[10];
    itoa((*(bsGameManager::getInstance()->all_elements_ + i))->id(), buffer,
    10);
    for (int p = 0; p < 100; p++) {
      if (id[p] == '.')
      {
        id[p] = buffer[0];
        if (i >= 10) id[p +1] = buffer[0 +1];
        if (i >= 100) id[p +2] = buffer[0 +2];
      }
    }
    b_->insertData(id);

    char id3[100] = "INSERT OR REPLACE INTO EnabledEntitiy(Enabled) VALUES(.
    );";

    char buffer2[10];
    itoa((*(bsGameManager::getInstance()->all_elements_ + i))->enabled_,
    buffer2, 10);
    for (int p = 0; p < 100; p++) {
      if (id3[p] == '.')
      {
        id3[p] = buffer2[0];
      }
    }
    b_->insertData(id3);
  }
}
```

This is an example of inserting data. In a char* I write a point where it has to be inserted and it is given up to a space more than the necessary in the parenthesis in case there is any error. No more spaces are used because from 2 it gives an error.

To load the game mode is something simpler but in essence the same procedure.



```
void savePuzzle(bsDataBase *b_)
{
  char id[200] = "INSERT INTO PuzzleMode(Puzzle) VALUES(.);";
  char buffer[10];
  itoa(bsGameManager::getInstance()->puzzle_mode_, buffer, 10);
  for (int i = 0; i < 200; i++) {
    if (id[i] == '.')
    {
      id[i] = buffer[0];
    }
  }
  b_->insertData(id);
}
```

1.5. Personal Tasks and Workgroup Plan.

The work has been done by one student so there is no distribution with more people. The order that has been made has been the total structuring of the game with all the classes and later implementation of the Singleton of GameManager and then to have implemented the game with its logic and its operation and the change of edition mode.
With that we have proceeded to implement the IMGUI to be able to edit the game in real time and change the attributes of the objects that we want.

With this working, the database has been implemented. It has been the most complicated thing for me and the result although it is correct has not been completely satisfactory because it is slow and I have required the use of 2 global functions to save the data of the static function.

The time has not been distributed correctly but it has been working as it could. High workload the last few days.

1.6. Short User Manual.

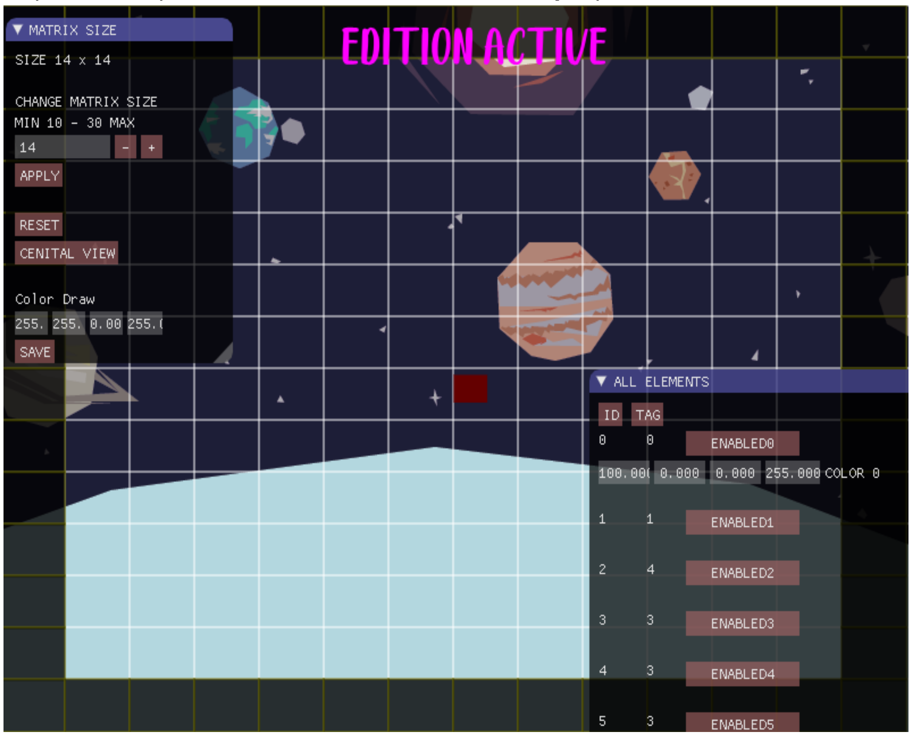
The game controls are:

- Mouse: Used to enable or disable objects and modify values in real time. Left click to enable and right click to disable.

- E': to change the real time edition mode or normal game mode.

- Arrows: the use of the 4 arrows for movement, depending on the mode you use 2 or 4.

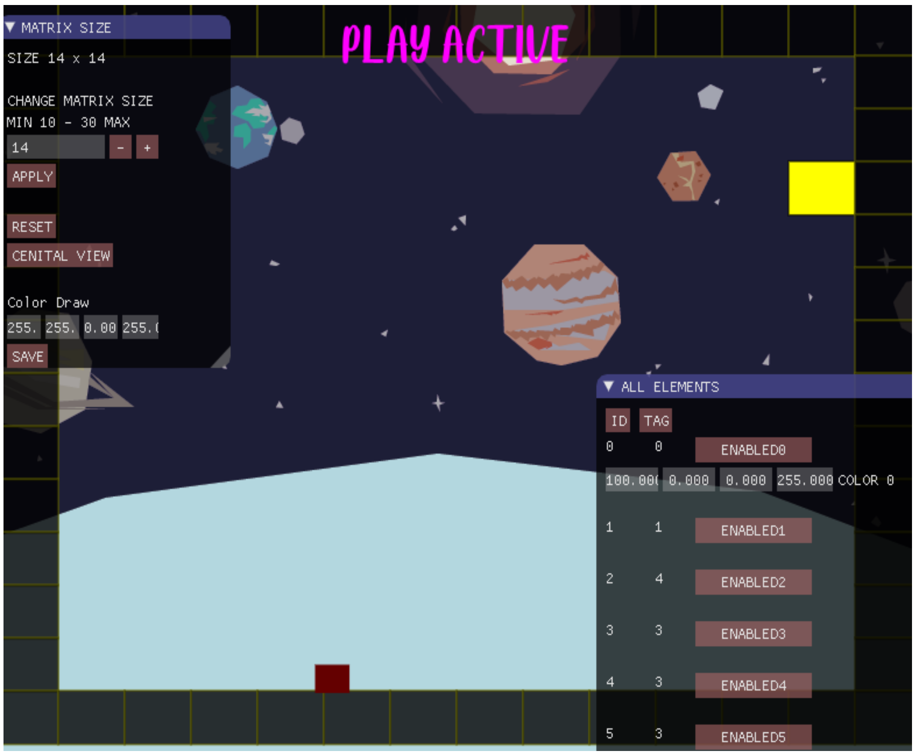
- Space: if the puzzle mode is disabled, it is used to jump.



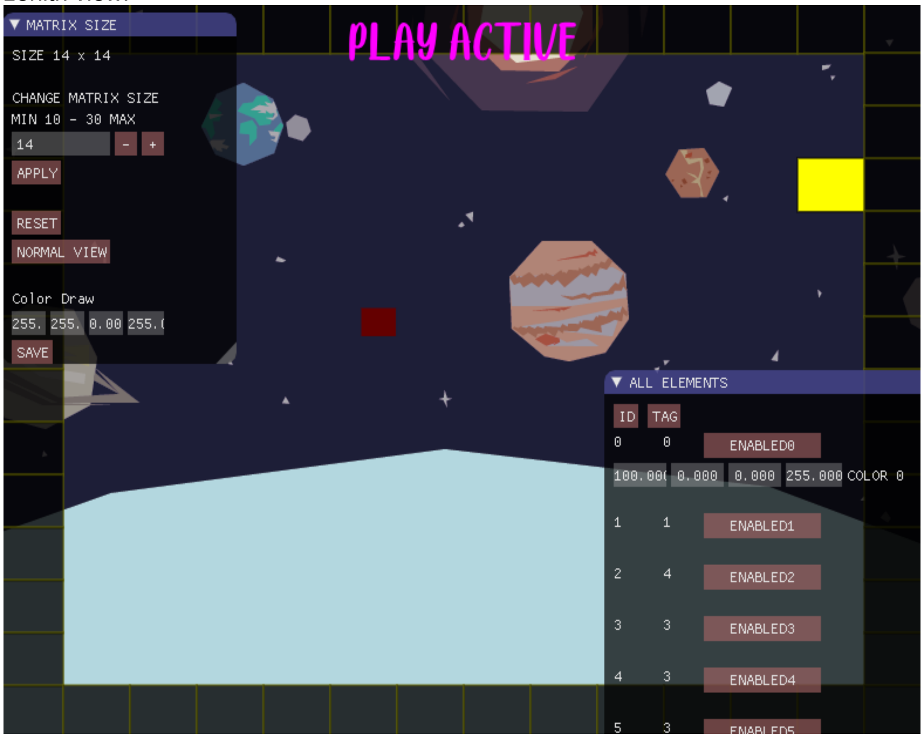
SIZE 14 x 14
MIN 10 - 30 MAX
14 - +
255. 255. 0.00 255.(

ID	TAG	
0	0	ENABLED0
100.00(0.000 0.000 255.000 COLOR 0		
1	1	ENABLED1
2	4	ENABLED2
3	3	ENABLED3
4	3	ENABLED4
5	3	ENABLED5

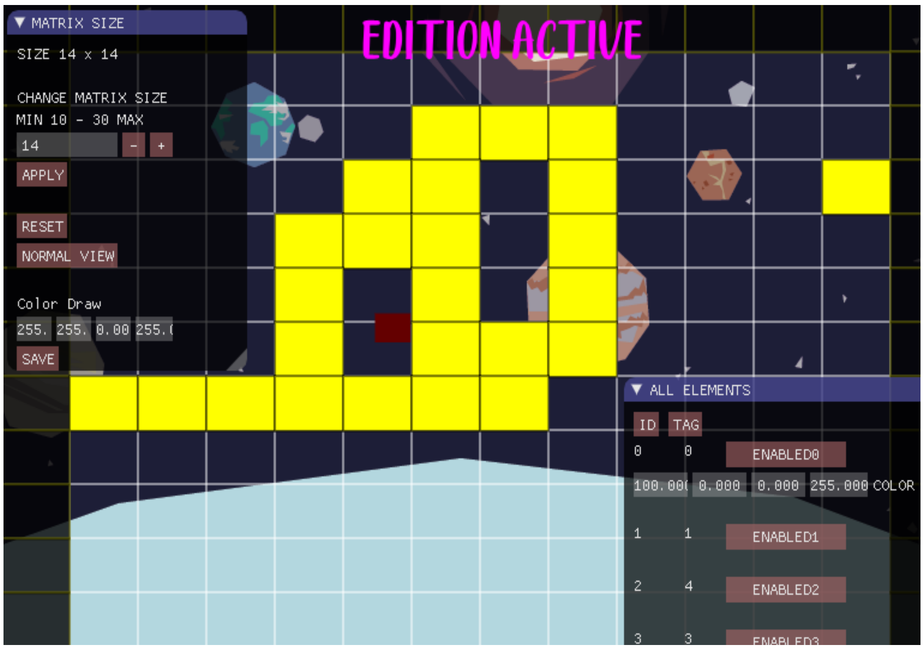
The editing mode or game mode is indicated.



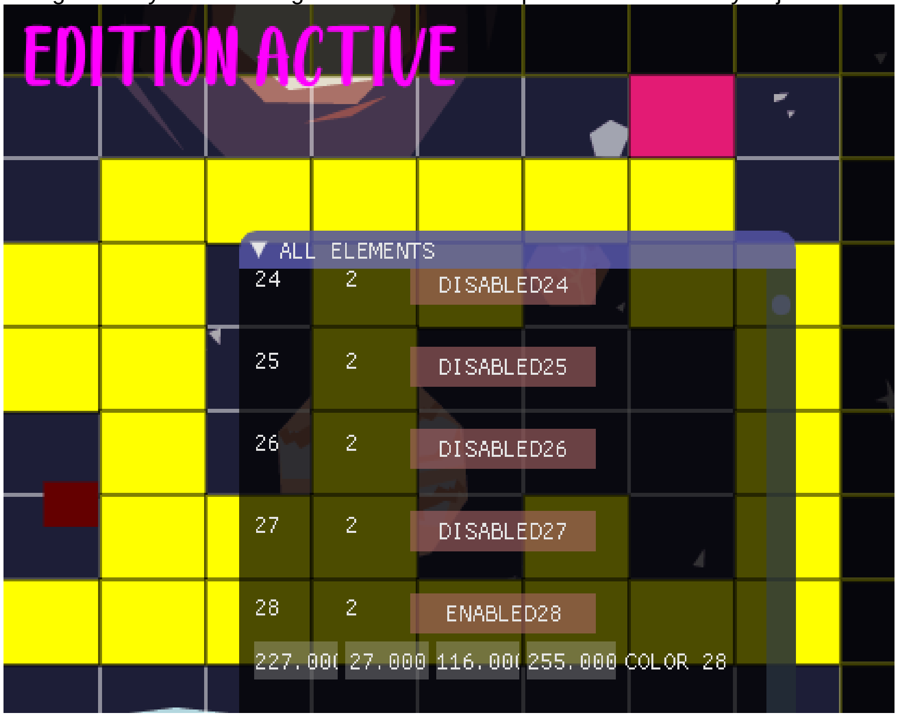
The puzzle mode is the zenith view with which gravity is activated or the game changes to zenith view.



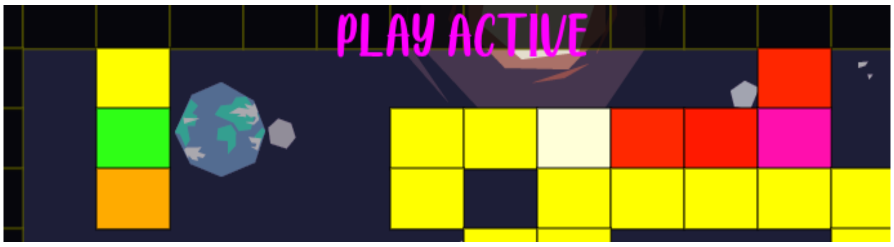
The editing mode allows you to activate the squares in order to make a level.

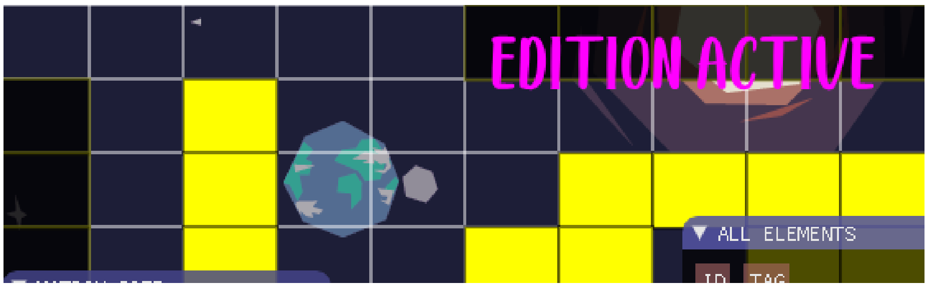


Using IMGUI you can change the colors of the squares or disable any object in the scene.

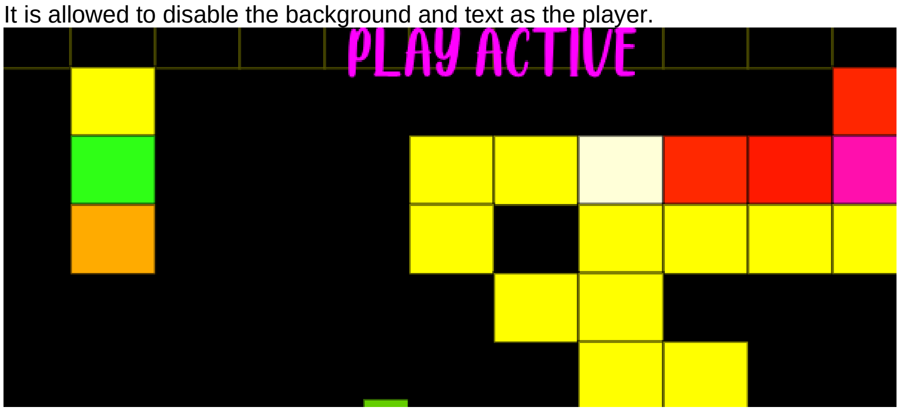


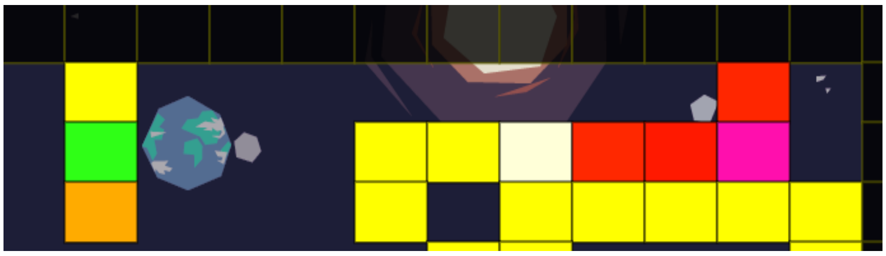
Border color editing is not allowed, but it can be enabled or disabled.



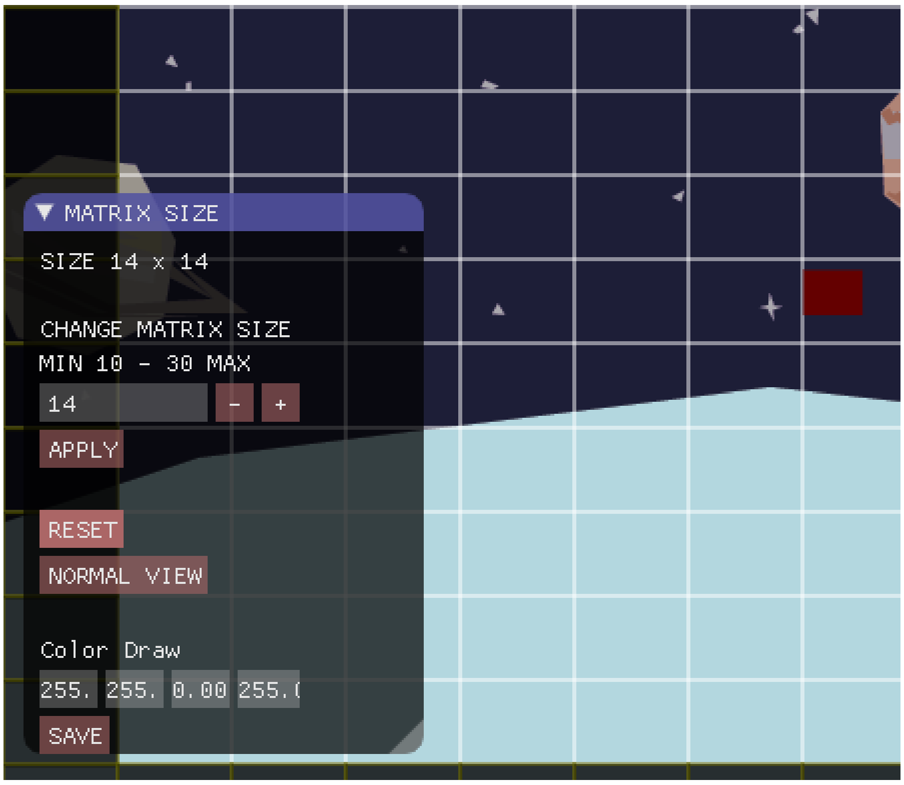


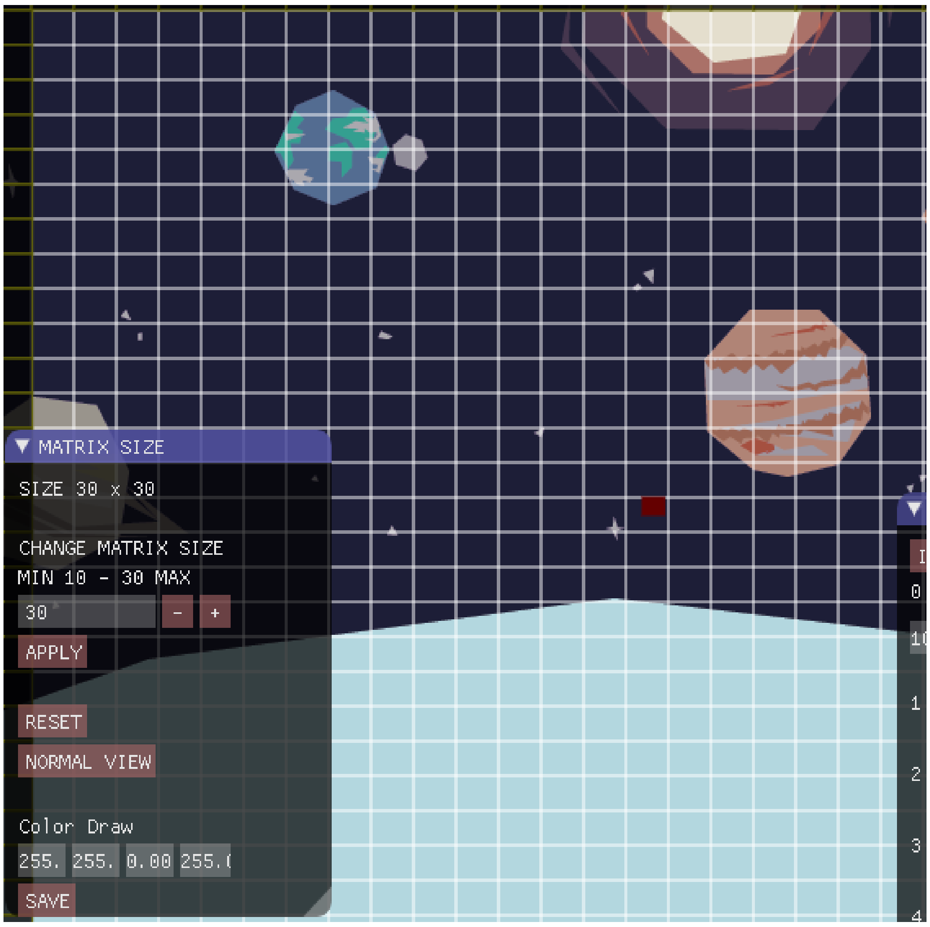
It is allowed to disable the background and text as the player.





You can reset the level to the same size or allow it to be reset.


▼ MATRIX SIZE
SIZE 14 x 14
CHANGE MATRIX SIZE
MIN 10 - 30 MAX
14
-
+
APPLY
RESET
NORMAL VIEW
Color Draw
255.
255.
0.00
255.(
SAVE


▼ MATRIX SIZE

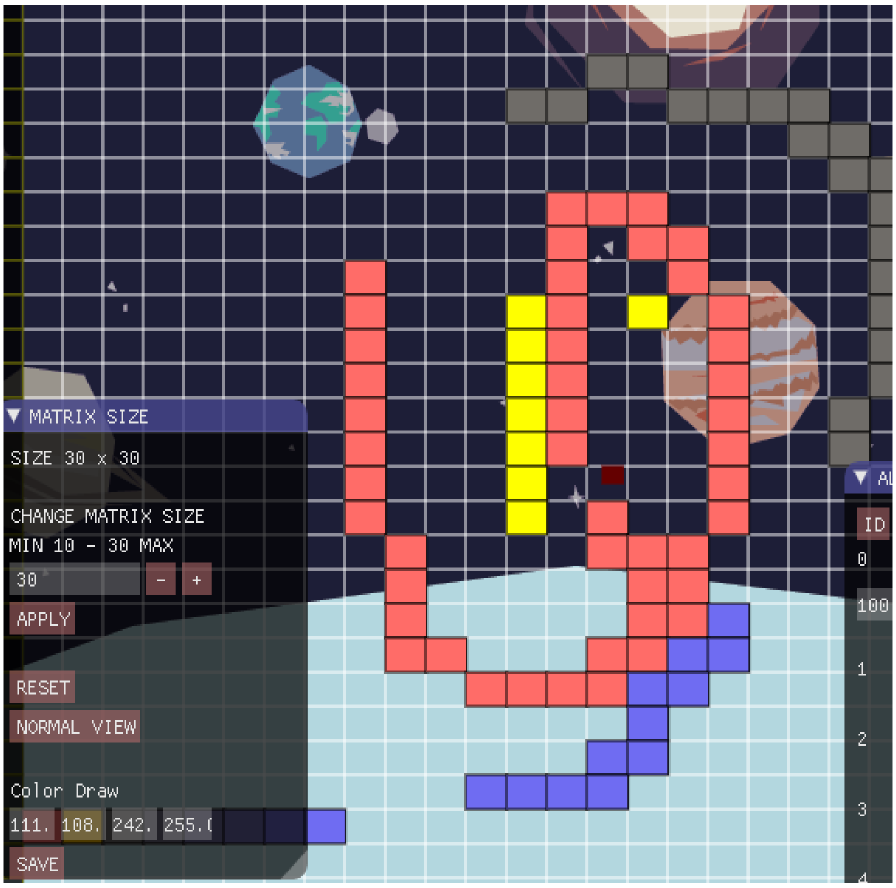
SIZE 30 x 30

CHANGE MATRIX SIZE
MIN 10 - 30 MAX
30 - +
APPLY

RESET
NORMAL VIEW

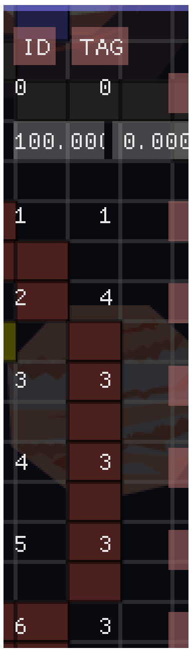
Color Draw
255. 255. 0.00 255.(
SAVE

The character's movement is adjusted to each size and also allows you to change the overall paint color with Color Draw.

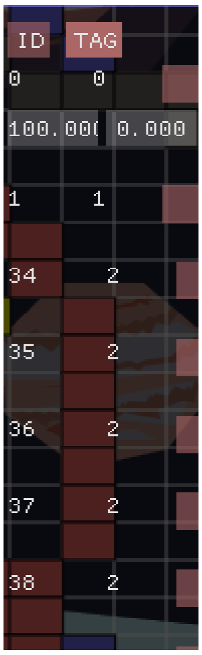


Finally, the entire level can be saved with the save button. The status of the objects is saved.

It is possible to sort the objects by ID or by TAG.



ID	TAG
0	0
100.000	0.000
1	1
2	4
3	3
4	3
5	3
6	3



ID	TAG
0	0
100.000	0.000
1	1
34	2
35	2
36	2
37	2
38	2

1.7. Conclusions and Future Work.

El resultado de la practica ha sido satisfactorio, aunque no la gestión del tiempo. Se cumplen los objetivos y es agradable de ver y de jugar. El modo edición está muy conseguido.

Como mejoras sería la implementación de la base de datos de una forma mas eficiente y se podría añadir control total de cada objeto como guardar su color o intercambiar posiciones.

1.8. Bibliography.

- http://opositare.com/normativa-programacion-didactica/
- https://www.genbeta.com/a-fondo/ides-y-editores-que-diferencias-hay-entre-ellos-a-la-hora-de-escribir-codigo
- https://www.msn.com/es-xl/noticias/microsoftstore/%C2%BFqu%C3%A9-es-y-para-qu%C3%A9-sirve-visual-studio-2017/ar-

AAnK0kx#:~:text=Visual%20Studio%20es%20un%20conjunto,entorno%20que%20sop

orte%20la%20plataforma.
- https://thegroyne.com/2015/05/por-que-usar-visual-studio-code-microsoft/

- https://www.drauta.com/atom-un-ide-para-el-desarrollador-web

- https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/programacion-imperativa/

- https://wiki.uqbar.org/wiki/articles/paradigma-de-programacion.html