An Intelligent Visual Path-Planning Framework

Jingming Guo, Haoyu Ji, Tianchi Fu
College of Software, Jilin University
JLU
Changchun, China
gjm54130209@hotmail.com

***Abstract*—Path-planning is a technology applied widely in many fields such as auto navigation, logistics management, robotics and so forth. In this paper, an intelligent visual path-planning framework (IVPF) is proposed for developing intelligent visual path-planning system, which consists of complex graphs drawing module, two dimension path-finding algorithms, map processing module, collision processing module, and network communication module. The proposed method is evaluated on graphic efficiency test and server stress test. The experimental result shows that IVPF could get not only high efficiency but also good stability.**

Keywords—Path-Planning; Path finding algorithm; Cocos2d-x; Collision calculation; Network Communications

I. INTRODUCTION

Path-Planning, as a key technique in plenty of fields, has been a major thesis for researchers all over the world [1]. Moreover, a large amount of researches have been done on various algorithms related to path-planning [2-8]. Amount these algorithms, the most widely used ones are A star algorithm, ant colony algorithm, genetic algorithm and so forth [9-11]. However, current researches mostly have their restrictions on specific application fields. For example, robotics, map navigation.

To make a path-planning algorithm more adaptive so that it could be workable for more fields' applications, the IVPF comes into discussion. It is a framework in which a module for visual graph drawing has been integrated which is responsible for delineating elaborate graphs with modest time and memory costs. Furthermore, improvements have been brought into the common A-Star algorithm to optimist its practicability and computing efficiency. Moreover, there are modules for handling collisions: map processing module and collision processing module. In order to provide capabilities to be controlling remotely, a network communication module must be indispensable.

All of the aforementioned parts will be introduced as the following orders: Firstly, graph drawing module and path-finding algorithm will be explained in details followed by illustrations of the map processing module and collision processing module. Secondly, network communication module mechanism will be introduced. Lastly, results of graph drawing efficiency tests and server stress tests will be given to demonstrate how well this framework could do its job.

II. RELATED WORKS

A. Cocos2d-x

Cocos2d is a widely used mobile game developing engine [12]. It has following characteristics:

- Cross-platform: once finishing, a project could be deployed on most major platforms (Windows, Linux, Android, iOS and so forth).
- Tree organizing classes, which brought convenience to usage and management
- Abundant drawing functions.
- Capabilities to draw graphs with high efficiency.

In cocos2d-x, there are several classes for organizing a game scene: Scene, Layer, and Sprite. All of them inherit from Node class and form a tree structure. For example, Layer nodes can be added to Scene nodes so that this scene will display several layers' concatenation. And similarly sprites can be added to layers as child nodes for showing new sprites on screen.

B. A Star algorithm

A-star is a way of heuristic search which is commonly used for path-finding. A-star algorithm's evaluation function is:

$$f(n) = g(n) + h(n) \quad (1)$$

In this equation, f (n) represents an evaluation for a single node get by adding g (n) with h (n). g (n) is the cost of this node while h(n) is the heuristic function for evaluating heuristic values of nodes. In A-star algorithm, two tables need to be created: OPEN table and CLOSE table. The former one is used for recording those nodes that has been generated yet not checked while the latter one is used for recording those already been checked.

At the beginning, the f (n) value would be calculated for the start node which would be added into table OPEN after the calculation. Then, a loop would take the responsibility to find the target node. When entering that loop, the optimized node with smallest f (n) would be fetched from the OPEN table, and extend all its accessible neighbors. Then calculate their f (n) values and keep a trace of their parent nodes. Add those already been tackled nodes into CLOSE table. At the same time, add those haven't to OPEN table. This loop would be

© 2016. The authors - Published by Atlantis Press

executed repeatedly until finding the target node or figuring out that there is no path between the start and the end.

It has been proven that A-star could find the optimized path without searching too many nodes as long as g (n) and h (n) functions are defined appropriately.

C. *Box2D*

Box2d is an open source 2-dimension physical engine which was firstly developed by Catto with C++ language in 2006. Now, this engine is available in other versions like Flash, Java and so forth [13].

There a cluster of vital concepts in this engine. They are rigid body, shape, and world. Rigid body is a hard object in which any pair of points on that body keeps a constant distance in any motions. Shape refers to 2-dimension collision geometry which is strictly attached to objects with material characteristics like frictional and ability to restitute.

To use Box2d, there is a common procedure: First, create a physical world. Second, add some objects to this world with specific shapes and decide which of them should be rigid bodies. Third, create moving functions and collision handling functions.

D. *IOCP*

IOCP (I/O Completion Port) is an asynchronous I/O communication model which is frequently used in high performance server implementations.

This model concatenate the Socket interfaces with I/O completion interfaces while communication is done just like a common socket program. Once a socket read/write request or read/write completion event occurs, the related completion port would be put into a queue by the operating system. Asynchronous I/O actions are done by fetching and processing ports sequentially from the queue.

The implementing of such model is tougher than common asynchronous I/O communication models. But it is worth it for this model has plenty of advantages over common models. The most manifest one is that managing thread pool and buffer pool could be done through IOCP which would ferociously cut costs caused by frequent context switching and space allocating so that a better executing efficiency could be guaranteed.

IOCP has been well developed that increasing number of develop platforms are supporting this structure like .Net, JAVA and so forth.

III. IVPF

A. *Framework Construct*

The figure depicts the architecture of an intelligent visual path planning system implemented under the IVPF framework. The system contains two parts: One part is deployed on user devices working as a client, the other parts are deployed on the servers working as server program. Clients and servers transfer data to each other through network communicating. And servers and databases interact through database API for data transmitting.

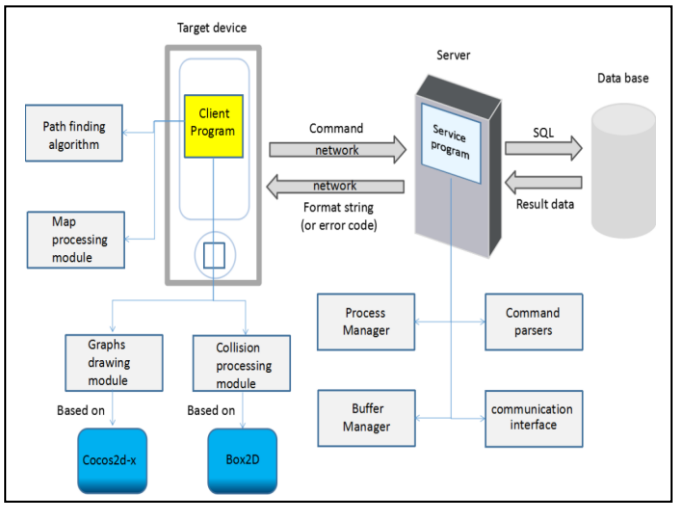

Fig. 1. Framework construct

Client program is consists of four modules. Drawing module is responsible for drawing user interfaces and all required complex graphs. Map processing module's duty lies on transforming input data into a matrix for storing and management. Collision module would handle computations for all collisions occurred on the screen for any objects and it would call corresponding collision reaction functions. Path finding algorithm would take charge of finding an optimized path on a given map stored in a specific data structure.

Server program also has four major parts: Thread manager would maintain the thread pool. Buffer manager would be responsible for managing buffer areas. And command analyzer would tackle with received commands and call related functions to perform functions expected by the client. Communication interfaces implemented under IOCP model would take charge of managing socket and cooperate with thread pool and buffer pool to manage resources on servers.

B. *Complex Graphs Drawing Module*

Graph drawing module is implemented based on current classes of Cocos2d-x with customized reforms. In Cocos2d-x, most graph elements are loaded to the screen with Sprite class, which is a class with sever restrictions for that a Sprite object can only take one picture as its texture while most graph elements in practice are usually consist of several parts with varying movement pattern. In another word, the original Sprite class cannot represent a wanted graph in many cases.

So a class called complex sprite is used to settle this problem to make drawing complex graph elements simpler. Complex sprite classes are organized in a tree structure with every node representing a sprite object. All sprite objects' attributes are described in extreme details in XML files. And there are two types of XML files; one is status while the other one is elaborate attributes description. The root tags in each XML file are dict tags. In another word, an object's all graphic information is stored in a pair of dict tags.

In the file of detailed attributes, the detail of each state which belongs to their relative object is mainly stored. This kind of XML file is composed by one dict label and several

array labels. Content of each array label stands for a real array and each member of the array stands for all the information of an object's state. Among these are types of sources (Frame Sequential or static pictures), the resource path, the anchor's coordinate, relative coordinate and the coordinates' contraction ratio. We can get to know the attributes of different parts of complex figures, furthermore, their mutual relationship in location.

As for the code implementation, firstly we should code a tree whose each node has the type Sprite. Then basic operations could be offered, mainly three operations:

- Acquiring the pointers of nodes according to the token of the nodes of the tree.
- Adding a child node to a known node.
- Deleting one sub node which has a root node according to the tokens.

After that, we should create a class for management whose purpose is to create the objects of complex Sprite class. It is recommended to use singleton pattern while coding this class. The following characteristics are necessary:

- The configuration information of the loaded object is stored in a data structure.(can be implemented by Hash Map).
- Some methods can be used to load the prepared XML file in advance, to obtain the configuration information of an object.
- Manage the configuration information that has been loaded (add, delete, query, modify).
- Provide methods for creating a complex sprite object.
- Store all created objects with a data structure.
- Manage the life cycle of all created objects.

After implementing these functions, the main function of the module is realized. We can use functions which are provided by Cocos2d-x to realize other goals we need.

C. Two Dimension Path-Finding Algorithm

How the A-star algorithm works has been discussed in the second section. And, in the IPVF framework, some optimization have been made on the base of the original A-star algorithm. Following are the incentives for optimizing this algorithm and the way to do it.

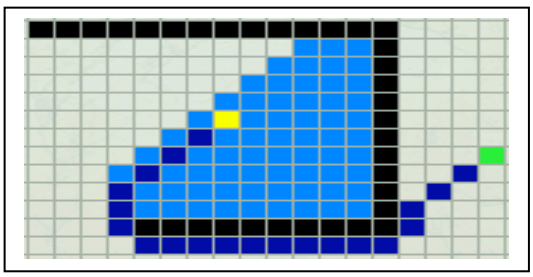

Fig. 2. A star algorithm test result

The figure below depicts a path and all extended nodes acquired by using an original A-star algorithm. Black grids are barriers. Yellow grids represent the target node while the shamrock grids represent all extended nodes. And the dark blue grids mark the optimal path. In the optimized algorithm, g (n) is defined as the distance from start node to the current checking node while h (n) is defined as the distance from current node to the target node.

It is clear that although A-star algorithm found a shortest path from the start to the end, plenty of nodes with low possibilities to be in the optimal path have been computed and compared. Taking the situation of the figure above as an example, the start point is surrounded by three sides which make it impossible for those nodes lying directly on the right of the start point to be in an optimal path. So the work done for checking them was totally in vain. If an adaptation could be made to magnify those nodes' f (n) values which would make them less possible to be checked before those that are more likely to be in an optimal path, there would be less nodes to be extended to find an optimal path.

In order to make this improvement happen, a per-processing procedure has been added to the original algorithm. P(n) is defined as a description of a node's probability to be on a optimal path. Commonly speaking, a node with less barriers around it tends to have larger p(n) value which make it more possible to be in a optimal path. In the per-processing procedure, the p(n) values of every node would be calculated. There is a simple way to initialize them: make those rights next to the barriers nodes have the same p (n) value and make this value bigger than others. After calculating p (n), use equation

$$f(n) = g(n) + h(n) + C * p(n) \qquad (2)$$

to calculate evaluations of every checking nodes. C is a constant representing the weight of p (n) in the computation.

However, when a map is extremely large, it could be difficult to find all barrier grids. To address this problem, another variable w is needed which indicates the searching depth for barriers. In per-processing procedure, only those barriers within w depth would be considered for calculating p(n) values to prevent the per-processing time from being too long.

The figure below shows the search result using the optimized algorithm.

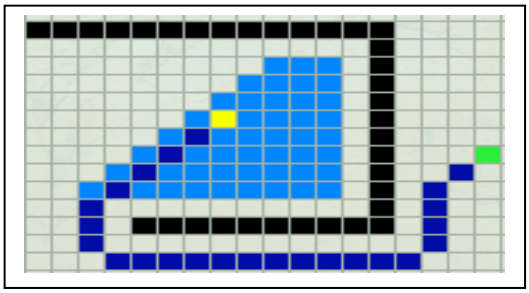

Fig. 3. A star algorithm test result

It can be seen that although the path after the improvement is a little longer than the original path (because it is not that close to the obstacle). But the number of nodes extended is much less. Moreover, considering the practical application, the obstacle is difficult to be indicated completely with blocks. It is not completely without obstacles even if we abstract the model into the field of "plain" in the 2-dimension array, maybe there are relatively few obstacles. This is most likely to occur on the nodes around the obstacles. If it is a robot way finding system and after computing, the optimal path in a node is near the obstacle, then while passing that position the robot may coincide to collide with the obstacle, which may cause other problems. In short, it may be a wise solution to choose a longer path rather than a route which clad with the obstacles.

D. Map Processing Module

For path planning technology, the processing of map information is essential. We create a class of map, this class majors in solving all the problems related to the maps. Its functions are: recording the location of the terrain, displaying the location of each object and providing coordinate conversion function. The module is also responsible for the coordination with the drawing module to correctly display the location of the unit, cooperating with the collision calculation module to calculate the collision event correctly.

Map class is consisted of three main parts. The first part is a 2-dimension array which is used to display map blocks. Before implementing this, the size and partition patterns of a map need to be settled. The size here contains two meanings; one is the actual size of the map, that is, the horizontal and vertical direction pixels. Next is dividing the map into blocks, which also indirectly determines the size of each block. After determining these values, we can define a two-dimension array, whose type is integer or enumeration. While painting a map, pictures should be displayed according to this array. On this basis, a few coordination transforming functions need to be available for transforming coordination between different frame axes like OpenGL axes and map axes. This is a vital part especially when using a 45 degree angle map.

The second part would be a Hash-map, in which keys are tokens of objects on the map (use name or other identifier for token). And a 2-dimension vector or customized type should be used for values which need to be able to represent coordination on a map. The main function of this part is to find its location on the map by Token of an object.

The third part is too a 2-dimension array in which every cell stores a link list for recording objects on maps. The size of this array should be the same as the map array. This part is mainly used to quickly find out what objects are in the certain location of the map.

Once all 3 parts are finished, it would provide convenience for managing objects and would make the collision computing easier. Based on which, a realistic battle scene can be display on the screen.

E. Collision Processing Module

Routing algorithm is not enough to avoid some unknown, unexpected collision, so we also need to deal with the collision with physical engine: Specifically speaking, the problem is when the two objects are in contact with the nature of the collision.

About the collision and the action after the collision of a rigid body, a class named b2ContactListener in Box2D is needed. We have two choices, one is directly modifying the method in this class, and its main method is as follows:

- BeginContact: Call when two objects begin to contact.
- EndContact: Call when two objects are separated.
- PreSolve: Call when two objects continue to contact, obviously this may be called multiple times.
- PostSolve: PostSolve is similar with PreSolve, it need to be called after the PreSolve.

However, considering the re-usability, it is decided to inherit the class and override these methods in the subclass. In fact, b2World, b2ContactListener, and b2ContactFilter to be mentioned below is the only three of all the classes who are not managed by engine management, but need to be created by ourselves.

But we also need to modify the original code of b2ContactListener, such as our sprite class has defined some "no reaction objects "like walls, playing no part in any collision. We turn their name in the collision event into 0, this need to refresh in the Event class. This kind of code is already mature enough.

We need to get their names in order to know the collision between which two objects. This is implemented by contact.GetFixture and A.GetBody. We have defined this name in our own class of complex sprite class. In this way, we can see that when there are no names in the two colliding objects, they throw a collision event. So we can listen to the event and do the logical processing.

An initial action function can be used to do this, by the introducing the function addEventListener. And customize our own handling of events, for example, in the console output information, etc. There are many kinds of output methods.

The following is the beginning of a formal discussion of specific collision processing: that is to determine whether there is a collision action.

Box2D support collision filtering by using population and group index. Box2D includes 16 populations, and we can specify which populations are in which shape. At the same time, it can also specify the shape and the collision with other population. This is accomplished by setting the value of categoryBits and maskBits. CategoryBits is used to define the type of collision they belong to, and the maskBits is the type of which can be collided. If an object is to collide with other populations, and its maskBits value should be the sum of other categoryBits.

But this could be the two meaning. What if the makeBits equals to some sum of other categoryBits of other population? Which population should the collision take place? Actually, categoryBits should be value which is multiple of 2. This solves the two-meaning problem perfectly. Next is the concrete implementation. Class, b2ContactFilter offers the default collision filtering function named ShouldCollide, the core is the determination whether objects belong to the same group and whether it contains categoryBits of other population.

We can also define the ShouldCollide by ourselves. Such as returning the true directly, which means it will collide anyway.

Since the physical world has been set up, the direct use of SetContactFilter is practical. Adding custom filters is also acceptable.

F. Network Communication Module

1) The client

In the communication module, the client program is designed to collect the data, and transmit it to the server to record and analyze the data. The client needs two parts. Part of the operation of the Socket (the actual communication part), the other part is the instruction generator. The part of the Socket operation is relatively simple, we only need to use the variables to record the server's address information and create Socket on the basis of address information and protocol type. After this, create the data which is in the read / write buffer which is for receive/send.

The function of the instruction generator is described below. In fact, the most convenient data type of network communication is the string type. So between the client and the server we can use the mutual normalized string (instruction) way to communicate. The instruction generator generates an instruction that represents the operation, including data, when the user makes an operation. Then send it to the server, through the resolution of the instruction, the server will achieve specific functions. The format of an instruction is given here:

Command_head;req1_para1_para2_para3;req2_para1_para2#

Command head is the head of the instruction, the type of the identification instruction. Req identify request type, para records parameter. Different parameters are separated with '_', ';' is used to separate between different requests for segmentation, '#' is the end of instruction.

2) The server

For the path planning framework, one of the most important problems is the communication efficiency.

First, create a thread pool and buffer pool. The former one is used to organize threads dynamically which would create several threads on the server. When a terminal is linked to the server, a thread would be allocated to that particular terminal device. This practice would avoid lots of I/O costs caused by frequent connecting/ disconnecting to the server or creating / deleting threads. A linear list could be used to store created threads' address, using a stack to store indexes of idle threads. Once a client tries to link the server, the index stack will pop up an index for fetching the corresponding thread. Once the link ends, clear the thread and push its index back to the stack.

Buffer pool is used for a similar purpose with a similar implementation which is used for managing memory dynamically. When the system starts to work, buffers would be allocated automatically. One way is to allocate a large space for future partitioning. And the other way is to allocate small pieces of memory for future use. Then, indexes of every unit of memory would be created. When a client links to the server, a piece of memory would be distributed for reading or writing data. When the connection ends, repossess that piece of memory.

After that, we need to code an instruction parser. Since in the network communication, the transmission of string is very convenient, it is also prevalent in the game development. Previously we have mentioned, the client will send a fixed format instruction to the server. For instance, login;account_xxxxx@163.com;password_123456# stands for that the xxxxx@163.com user attempt to log in using the password 123456. Command parser parse out the various parts of the instructions, calling the other functions to complete the processing of information. The instruction parser may split the instructions above into three parts. 'login' is the command header, standing for 'login' instruction. '#' is the command terminator. Command parser recognizes the instructions firstly, user's name and password is transmitted to the database API called-in function, query whether information is consistent with the database and on the basis of results, do corresponding processing, thus completing the 'login' function.

After the completion of the above work and testing is completed, we can begin to prepare the actual communication part. IOCP technology is recommended here. IOCP is a high-level asynchronous I/O model, through the use of IOCP, we can easily manage the thread pool and cache pool. And headache problem such as how to judge the data transmission has been completed can be completely handed over to the operating system. Take C# as an example, it need to use the SocketAsyncEventArgs class provided in .Net. When the cache manager and the thread manager, which are written in the process, are managed by the SocketAsyncEventArgs class, the following operations are the same as the normal asynchronous I/O model. In other words, defining the callback function of the read / write event, passing the pointer to the SocketAsyncEventArgs class object, and then start in order of Bind, Listen, and Accept.

At last, a 'heart beating' mechanism is needed to send meaningless information to clients in certain intervals which can be used to judge if a user is out of connection according to client's response to the 'heart beat' information.

IV. EXPERIMENTS

Aimed at this framework, we have done some simple performance tests, mainly the drawing efficiency test and the server compression test.

Drawing efficiency test:

Type of devices:

- Dell Ins15RD-9518 (Windows 8).
- Lenovo G480N (Windows 8).
- SAMSUNG Galaxy S6edge (Android).

Test method: After drawing the background in the scene, load different pictures whose specifications are similar, make them move irregularly, observe their frame rate under different circumstances. Use several devices with different types and performances to test. The result is like this:

Device type	Sprites' count	Frame per-second (max : 60)
Ins15R|G480N|Galaxy S6edge	50	60|60|60
Ins15R|G480N|Galaxy S6edge	150	58|60|60
Ins15R|G480N|Galaxy S6edge	250	50|57|60
Ins15R|G480N|Galaxy S6edge	300	44|53|58
Ins15R|G480N|Galaxy S6edge	350	32|49|56
Ins15R|G480N|Galaxy S6edge	400	21|43|52
Ins15R|G480N|Galaxy S6edge	450	7|33|44
Ins15R|G480N	500	25|34
Ins15R|G480N	550	15|22
Ins15R	600	4|14

Fig. 4. Test of drawing efficiency for different devices

In general, when the sprite in the interface is less than 400, basically we can guarantee that the picture is very smooth, higher than 400 below 500, a little lag, close to 500, and 500 above the lag is more serious.

Comparing test results of different devices, it can be concluded that the computing efficiency and drawing CPU and graphics card have a great relationship. When CPU computing performance is more powerful, the graphics efficiency is higher, the graphics card computing power is more powerful with a higher the efficiency of drawing.

Server stress test:

As a server, the middle-configuration laptop is used to test the pressure of different connection number of the server. The string sent has a length of 30(Instruction average length). Test results are as follows:

Connections	Network delay(ms)
100	< 1
200	2~3
300	5~6
500	20
700	43
900	65
1200	100
1500	140
1800	176
2000	220

Fig. 5. The result of server stress test

From the test results we can conclude: when the number of connections is under 900, the sense of the delay is basically insensible.

When it comes to 900-1800, slight delay appears. When up to 2000, there will be more severe delay.

V. CONCLUSION

This paper presents an intelligent visual path planning framework. The framework can be used to develop a variety of fields of intelligent visual path planning system. In the framework, the function module includes the complex image, the path searching algorithm, and the collision process and so on. It can be applied to the processing of the actual path planning problem. By testing the efficiency of the drawing and the compression capability of the server, it can be observed that the framework is very practical. The next step is to continue to optimize the core of the routing algorithm, improve the framework of universal, and do more relevant performance tests.

REFERENCES

[1] Z. Guanglin, H. Xiaomei, C. Jianfei, Z. Lei, Y. Tao, "Path Planning Algorithm and Applications Summaries", Modern Mechanisms, pp. 85-90, 2011(5).

[2] Y. Jianli, V.K roumov, S. Zenqi, S. Sahilu, "A Fast Dynamic Nerve Net Path Planning Algorithm", Robotics, pp. 201-205, 2001.

[3] F. Mengyin, L. Jie, D. Zhihong, "A Route Planning Algorithm for the Shortest Distance Within a Restricted Searching Area", Journal of Beijing Institute of Technology, pp. 881-884, 2004.

[4] F. Mengyin, L. Jie, D. Zhihong, "A New Route Planning Algorithm Based on the Hierarchical Road Network", Journal of computer-aided design computer graphics, pp. 719-722, 2005, 17(4).

[5] S. Hongbing, Z. Yibing, T. Ruofeng, D. Jixiang, "Path Planning for Automated Navigation in Virtual Environment", Journal of computer-aided design computer graphics, pp. 592-597, 2006.

[6] H. Yinghui, Z. Lianming, "A Path Planning Algorithm for Mobile Robot", Techniques of automation and applications, pp. 8-10, 2003, 22(5).

[7] W. Xiaotao, S. zengqi, "Algorithm based on massively parallel connectionist network for path planning", Tsinghua Science and Technology, pp. 67-71, 1996(5).

[8] W. Guaiwei, F. Mengyin, "An Algorithm of Path Planning for Car-like Robots Based on Neural Network", computer simulation, pp. 112-116, 2010, 27(7).

[9] W. Zhongmin, Y. Hong, L. Jiyan, "Path planning of mobile robot based on improved simulated annealing algorithm", Computer Engineering and Applications, pp. 59-60, 2005, 41(19).

[10] L. Jun, L. Guangrui, "Convergence analysis of path planning based on ant colony algorithm", Machinery Design & Manufacture, pp. 164-165, 2010(8).

[11] X. Meiqing, S. Chenliang, "Genetic algorithm path planning based on raster map", science and technology information, pp. 76-77, 2011(31).

[12] L. Jianzhuo, "Cocos2D-X game development technology", Posts and Telecommunications Press, 2013.

[13] X. Yan, "Rigid body motion and collision simulation based on Box2D physics engine", Computer programming skills and maintenance, pp. 125-126, 2011(24).