IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 9, September 2019

Architecture of Online Multiplayer Gaming Environment

Suyash Kanungo

B.E. (Computer Engineering), Institute of Engineering and Technology, Devi Ahilya Vishwavidyalaya, Indore

Abstract: Multiplayer gaming is useful in fastest growing industrial, defence and educational applications like simulation of piloting aircraft, testing of robots and online learning. These applications require high performance architecture for extensive computations and sophisticated Artificial Intelligent algorithms. In this paper, architecture of a multiplayer gaming environment is proposed which uses peer-to-peer computing for improving gaming experience over the LAN. A* algorithm has been used for finding path from one point to other.

Keywords: Multiplayer Gaming, A* Algorithm, Socket Programming, Dijakstra' Algorithm, Defect Testing

I. INTRODUCTION

Games are interactive multimedia applications working in real time that require high performance Central Processing Units (CPUs) and graphics hardware. Some of the massively multiplayer games having decentralized architecture need the extensive computations for a number of simultaneously playing users. Gaming can be useful in a variety of professional and educational scenarios in the simulations for activities like racing cars and piloting air crafts. In an educational environment, gaming tool provide opportunities for in-depth learning like testing of A.I. controlled robots. To meet the processing needs of users playing simultaneously, multiplayer gaming require high performance computing architecture e.g. peer-to-peer model [9, 10, 16].

A Multiplayer gaming system may have following main Sub-systems of the game: (i) Map generation and presentation, (ii) Game Actor AI Modules and path-finding, (iii) Multi-Player Gaming Mode [18]:

A. Map generation and presentation: The game scenario is generated using simple images and objects in pygame library. The level design is handled by the module which helps in placing trees, vegetation, water bodies, castles and walls. This subsystem creates a level object for the game scene. The level is generated like a grid to use it as graph for path-finding.

B. Path-finding and Game agent: Path-finding in games is a task of finding a path from one point on the map to another without colliding into obstacles or stepping into impassable terrain. The most well-known algorithms for path-finding are depth-first-search (DFS), breadth-first-search (BFS), Dijkstra's and A* algorithms. A* algorithm differs from Dijkstra's only in the way the nodes are sorted on the queue. The problem with Dijkstra's algorithm is that it expands from the starting node in all directions until it eventually reaches to the goal node. However, in games, the graphs are usually planar and the costs of edges roughly correspond to the Manhattan distance i.e. the distance between positions of the points in space measured along axis at right angles. With point p1(x1, y1) and p2(x2,y2), Manhattan distance is |x1-x2|+|y1-y2|[2, 4, 11].

Contrary to Dijkstra's algorithm, the A* algorithm works by processing nodes which are located in the direction of the goal node. This concept is implemented by sorting the nodes in the open queue by finding with the help of heuristic function (called h). This functions indicates how close they are to the starting node (called f value) plus an estimate of how close it is to the goal node (called g value), i.e. h = f + g. The distance to the destination node can be, for example, estimated by the length of a straight line connecting the node and the destination node (Euclidian distance).

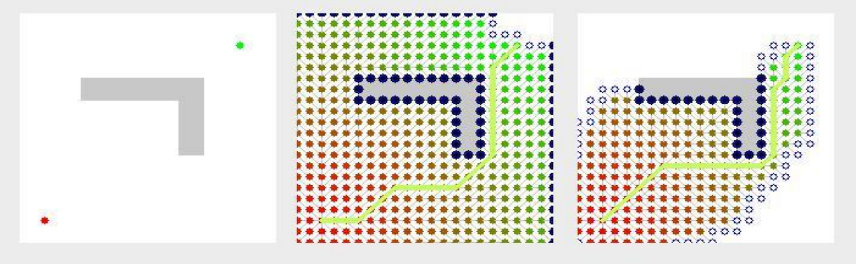

Fig.1 Comparison Performance of (a) Initial and destination nodes (b) intermediate nodes generated in Dijkstra's algorithm (centre) and (c) Intermediate nodes generated using A* algorithm (right).

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 9, September 2019

A* algorithm give much better results as compared to Dijkstra's algorithm on open maps with few obstacles and comparable speed with Greedy approach in simple cases. On a maze-like map, where the shortest path winds across the whole map, their performance converges. Example in Figure 1 shows that A* algorithm has to process fewer nodes than Dijkstra's to find the shortest path on a square grid with one obstacle from the red dot at the bottom left to the green dot at the top right in Fig 1(a). Game's requirements specify that tanks must have a meaningful motion when they aren't controlled by the player. AI-controlled opponents move according to the path given to them by path finding algorithms, but what different actions are to be taken depends in what state the agent is, since the scenario is dynamic and the path given by the path-finding algorithm may have an enemy on the way, so the decision to fire is taken by this AI Module. There may be other obstacles like a brick wall or the castle in front of the agent, and it fires if it lies on the path given to it. The AI module and the path-finding algorithms employed in the game often result in performance issues and result in a lag when the game is played on a network. Therefore, these modules need to be the efficient in case of real time strategy games [8].

C. Multiplayer mode: Most of the functionality is accessible as methods of socket API, which provide methods to create a socket's server [15] and client on the same node to implement the peer to peer local network as shown in Figure 2. Connection related events are all handled using sockets. The UDP protocol is used over TCP because this being a real-time game, forced retransmission by TCP in case of packet loss which may slow down the data transmission and the game while creating lag. Small amount of packet loss can be tolerated but game is response time critical [12, 13, 14].

II. MULTI-PLAYER GAMING ARCHITECTURE

Class diagram in Figure 3 represent relations among different classes which used in the implementation. Class **sprites** is built-in class whose methods like draw () and update () are overridden in its child classes like **tank, bullet.** All the attributes and methods in all classes have public accessibility. Level and cell classes are used to draw map of the game. All the classes are control by gameController class for interaction among tanks and map tiles in the game [5].

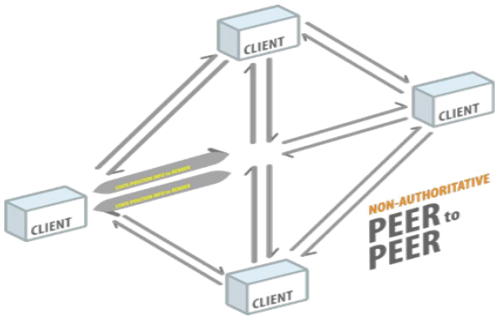

Fig.2 Non-authoritative game server architecture (peer-to-peer)
http://research.drawlabs.com

DOI 10.17148/IJARCCE.2019.8912

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 9, September 2019

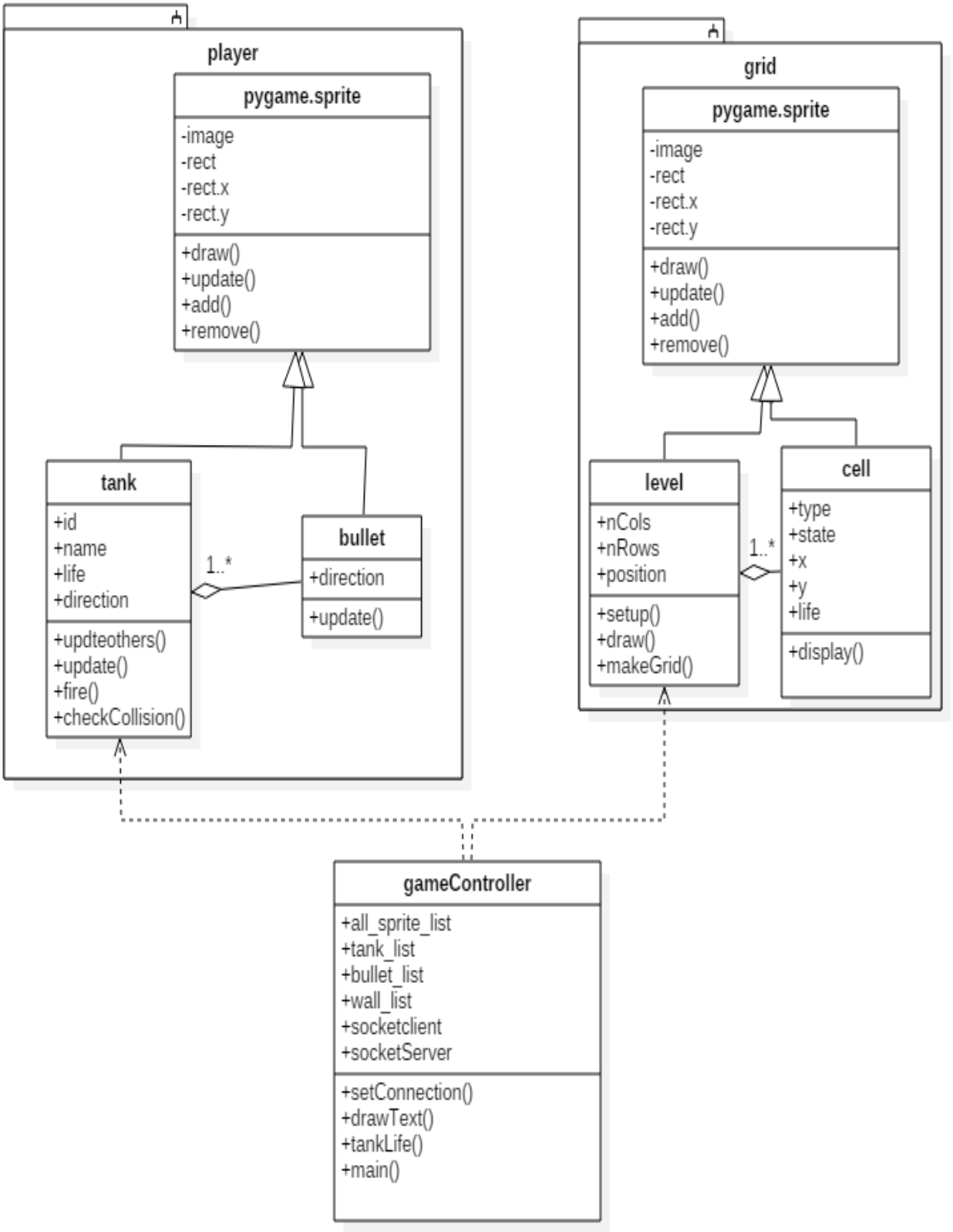

Fig.3: Class Diagram

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 9, September 2019

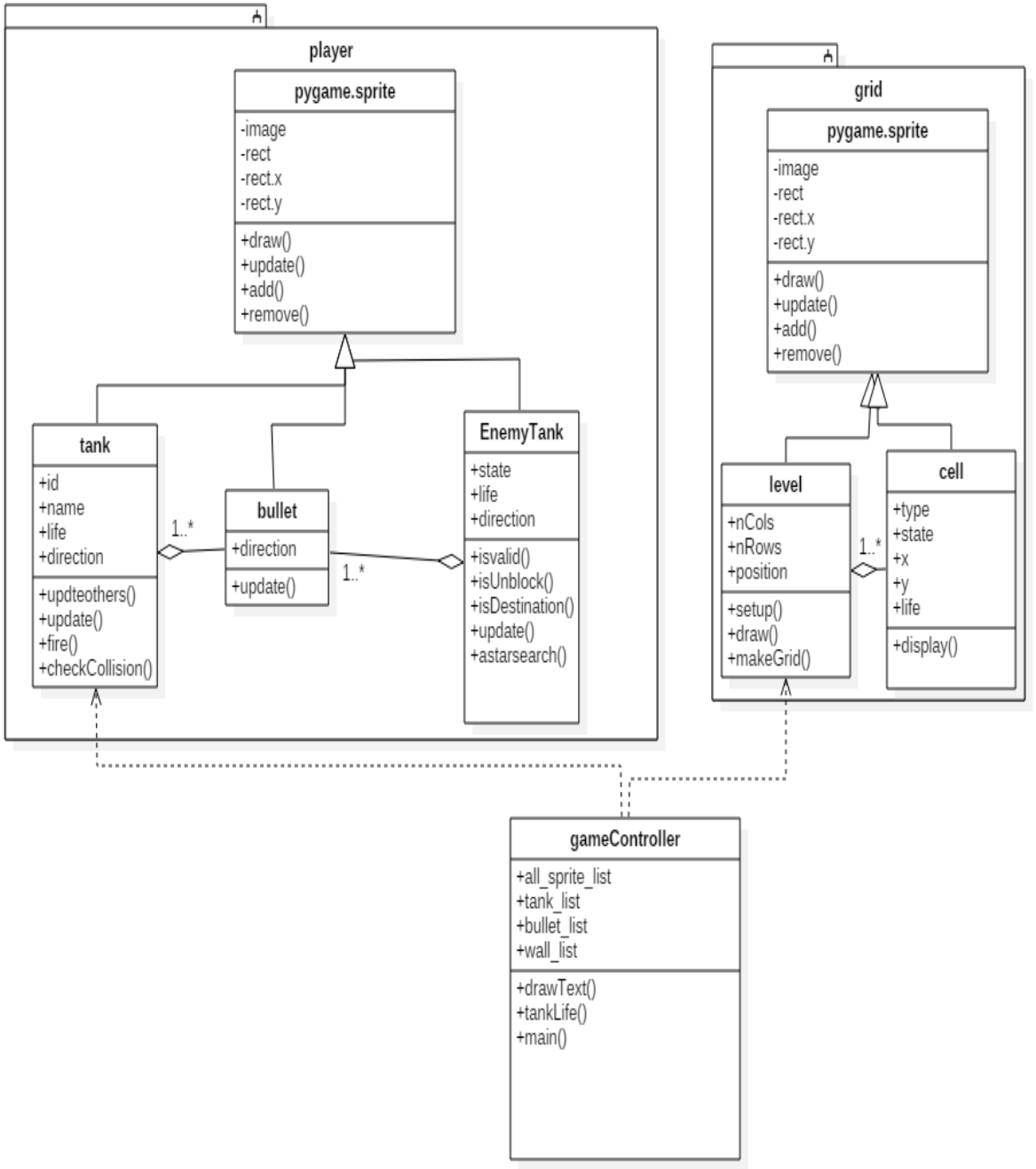

Fig.4: Class Diagram

D. Class diagram in Figure 4 represent relations among different classes created in our implementation. Class **sprites** is built-in class whose methods like draw () and update () are overridden in its child classes like **tank, bullet, enemy Tank.** All the attributes and methods in all classes have public accessibility. Level and cell classes are used to draw our map of the game. All the classes are control by game controller class for proper interaction among tanks and map tiles in game[6, 7].

III. TESTING AND RESULTS

This section discusses the techniques to implement main functionalities of the gaming environment. The systems has been tested and evaluated to find any errors and to tune certain functioning, specifically the algorithms and variables that control the game physics. The system functioning has been validated and the real-world physics demonstrated by

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 8, Issue 9, September 2019

the game has been refined after testing in three areas viz. Integration testing, Defect Testing and User Testing. The process was used throughout the implementation and even after completion. A Number of subsection were uses in each stage [17].
Black-box and White-box testing were used for the defect testing. Test cases for Black-box testing were taken from the Requirements Analysis. On the other hand, test cases for White-box testing were taken from the implementation, to evaluate the functionality of subroutines.

Wherever possible, all paths available within a component have been tested, although this was occasionally difficult due to the obscurity of some of the rule implementation tests, such as testing how the system would react when three or more tanks are potted. Wherever this was the case, the system was set up specifically for the test scenarios, rather than the correct positions in accordance with the rules, although for some scenarios this was still not possible.
On a small subset of gaming system, integration testing has been performed. The reason is that, many of the systems functions work independently. Main components are tested for integration, so that the functions in these mechanisms interact correctly. Test cases are included to monitor values passed to the functions and updating of objects in the system whenever needed. To quantify usability of the system, and to evaluate difficulty relating the performance of fundamental activities within the system, users' evaluations have been used. These testing strategies were sufficient to discover various functional or usability problems in the system. Unit testing was considered but was not required.

A. Defect Testing: Defect testing was carried out on three levels; control physics tests as shown in Table 1, system tests as shown in Table 2 and rule tests as shown in Table 3 using black box technique. As stated previously in the test plan, these areas were tested after functionality of a feature was completed. Black-box testing was used to find many of the high level system errors. White-box testing was therefore designed to test the underlying structure of the key functions within the system. Of the functions tested, not produced any erroneous output, which can be attributed largely to the resolution of errors during the Black-box testing. To ensure that the functions that performed the most important system tasks, perform as expected upon integrated, integration testing was deployed. A rare but significant error was identified in the functions that comprised the firing from a moving tank.

1) Physics tests

TABLE1
BLACK-BOX TESTING - PHYSICS TESTS

S.no.	Test	Expected Output	Actual Output
1.	collision of the tank with a wall	The tank should stop moving, and user has to change directions.	Output as expected.
2.	Tank collision another tank	The two tanks should collide and unable to move if they apply force in opposite directions	Output as expected.
3.	Tank collision with bullet	Do not stop any movements of tanks only updates life when collision between bullet and tank	Output as expected.

2) Control System Tests:

TABLE 2
BLACK-BOX TESTING - CONTROL SYSTEM TESTS

S.no.	Test	Expected Output	Actual Output
1.	Press spacebar to fire a bullet, keep it pressed for additional bullets.	Bullets are fired on keeping the button pressed.	Output as expected.
2.	Tank moves according to the direction keys pressed by user in each system.	Movement is user controllable by direction keys.	Output as expected.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 9, September 2019

3) Rule Tests

TABLE 3
BLACK-BOX TESTING – RULE TESTS

S.no.	Test	Expected Output	Output
1.	Hitting various objects e.g. stone, bullet, grass, water etc. with bricks.	Stone and water blocks the bullet path. Bricks hit by bullet removed from the current scene. Grass bullet is passed through	Output as expected but if game start before all player in game if any updates are made it won't update Actual it.
2.	Tank hitting another tank	Life of the tank player updated in based on the damage taken from bullet.	Output as expected
3.	Tank hitting castle	Castle life updated on the basis of damage taken from bullets if life zero then win.	Output as expected.

B. Testing techniques: Following testing techniques have been used :

1)Test-driven development: *Test-driven development* (TDD) is a commonly used methodology to reduce number of bugs in software and decrease the costs of finding bugs. The process of writing code according to TDD consists of short cycles:
4) A test is created before a method is written, to determine if the method returns correct results for all principally different inputs;
5) The test is run to make sure that it fails (because the tested method is not implemented yet and returns default value);
6) The method is written;
7) The test is run again. If it still fails, the errors in the method's implementation are fixed immediately. The method is considered complete as soon as the test succeeds.

This methodology is best suited for writing libraries and other reusable parts of code, especially those which take some data as input, process it and return some value as result. For games, it is used while developing parts of games engine: rendering, physics, audio, script interpreter and other parts. However, most of this already comes implemented in PyGame engine and doesn't require testing.

***2) Test game scenes*:** A more common way to test game scripts is to create separate test scenes for pieces of game mechanics (test cases). Such scenes are usually tested manually. The developer just runs the scene, performs a series of actions as if he was the player and makes sure that no errors are written to the console and the events happening in the scene are exactly what he expects.
Manual Testing proves to be more efficient because in games it is often much simpler to just run the game and see if "everything is ok" that to attempt to predict what may go wrong and write test cases for this. Many things in games can only be evaluated by a human, for example, if jumping animations looks realistic enough. For each reported bug, the programmer may create a separate test scene to simulate the conditions in which the bug occurs.

3) User Evaluations: As has been stated throughout this report, a game is judged on the opinions of users. Therefore, evaluation was performed to gain a better perspective successful design and implementation. Thereafter, we can quantify the system's usability requirements. Different users would have their own opinion regarding features to be incorporated in the game. One example can be how the life of each tank is implemented and should they respawn. Common suggestion was to show a visible life bar of each tank rather than re-spawning, to avoid frequent obstruction in game flow. According to the majority participants main aspects of the system include:

- Network connectivity was of a good level and represented game on each node well synchronized.
- Control system was well received finding it logical, fulfilling easy to use and memorable.

Altogether, the feedback from users was very encouraging. The features suggested by users were considered at some stage of development, some not found suitable after further analysis carried out and other features were not feasible, given the constraints of the project. Figure 5 shows the snapshot of the game.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 8, Issue 9, September 2019

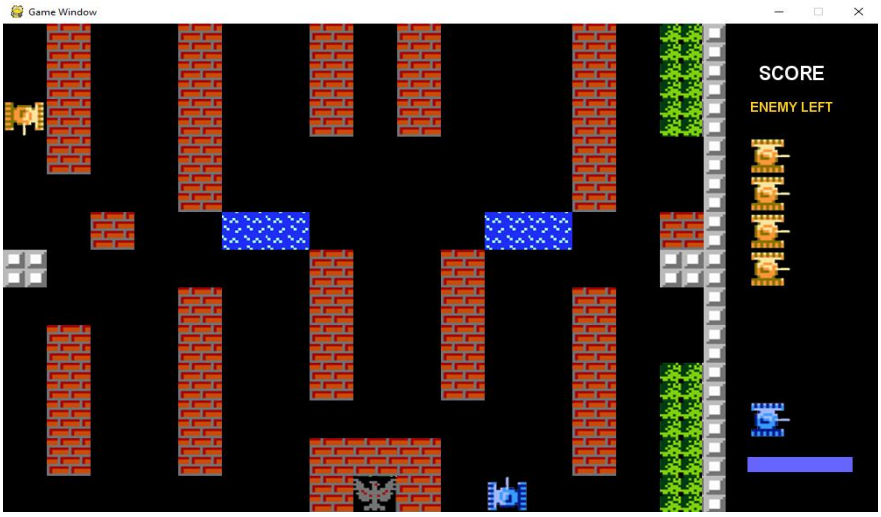

Fig. 5: A working single player game snapshot

IV. CONCLUSION

Developing a multiplayer gaming environment involves advance tools. This proposed game uses P2P Model for gaming environment for improving gaming experience over the LAN. It also implements security issues involved and their resolution. The technology used is based on Python and PyGame. Artificial Intelligence concepts like heuristics functions and algorithms have been used in this game. The game can be further improved by using Neural Networks for training agents, Cluster Computing for high performance, Machine Learning Techniques [1] like Reinforcement Learning and genetic algorithms for massively parallel applications.

REFERENCES

[1]. Amershi, J. Fogarty, A. Kapoor, and D. Tan2011, "Effective end user interaction with machine learning," in 25th AAAI Conference on Artificial Intelligence, pp. 7–11.
[2]. Bundy A., Wallen L. (1984) A* Algorithm. in Catalogue of Artificial Intelligence Tools. Symbolic Computation (Artificial Intelligence). Springer, Berlin, llHeidelberg
[3]. Chen, B. and Maheshwaran, M.(2004a). A cheat controlled protocol for centralized online multiplayer games. In Proceedings of the International ACM SIGCOMM Workshop on Network and System Support for Games (NETGAMES'04). ACM Press, New York
[4]. Cowley B. U. (2016), "How to advance general game playing artificial intelligence by player modelling," arXiv, vol. 1606.00401, Jun.
[5]. Cowley B., Charles D., Black M., and Hickey R. (2013), "Real-time rule-based classification of player types in computer games," User Modeling and User-Adapted Interaction, vol. 23, no. 5, pp. 489–526, Aug.
[6]. Dormans J., "Machinations: Elemental feedback structures for game design," in GAMEON-NA, (2009), pp. 33–40. [40] S.
[7]. /Laird J.andVanLent M.(2001), "Human-level AI's killer application: Interactive computer games," AI magazine, vol. 22, no. 2, p. 15.
[8]. Mehta P, Shah H, Shukla S, Verma S, "A Review on Algorithms for Pathfinding in Computer Games," available on https://www. researchgate.netn/profile/Saurav _Verma2/publication
[9]. Multiplayer game (2007), In Wikipedia, The Free Encyclopedia. Retrieved April 7, from http://en.wikipedia.org/wiki/Multiplayer_game
[10]. Owen Riedl M and Zook A., AI for Game Production
[11]. Russell S.andNorvig P.(2003), Artificial Intelligence: A Modern Approach, 2nd ed. Prentice Hall.
[12]. Sharma R.K., Kanungo P. (2011), " Performance Evaluation Of Parallel Applications Using Message Passing Interface In Network Of Workstations Of Different Computing Powers," Indian Journal of Computer Science and Engineering (IJCSE), Vol. 2 No. 2 Apr-May 2011, pp. 184-187.
[13]. Sharma R.K., Kanungo P. (2011), " Performance evaluation of MPI and hybrid MPI + OpenMP programming paradigms on multi-core processors cluster," IEEE International Conference on Recent Trends in Information Systems, 21-23 Dec, **DOI:** 10.1109/ReTIS.2011.6146855.
[14]. Shen S, Survey of P2P Game, Parallel and Distributed Systems Group, Delft University of Technology, Mekelweg 4, Delft, the Netherlands available on https:// pdfs. semanticscholar.org.
[15]. Socket Programming in Python available on http://www.biogem.org/ downloads/notes/Socket%20 Programming%20in%20 Python.pdf
[16]. Yang B. B. and Garcia Molina H. (2003). Designing a super peer network. In Proceedings of ICDE'03, pages 49 –60. IEEE.
[17]. Yannakakis G. N. (2012), "Game AI revisited," in Computing Frontiers, pp. 285–292.
[18]. Yoshizumi, T., Miura, T., Ishida, T. (2000), A* with partial expansion for large branching factor problems. In: Proc. of the Seventeenth ational Conf. on Artif. Intell. and Twelfth Conf. on Innovative Applications of Artif. Intell. 923–929.

BIOGRAPHY

Suyash Kanungo has obtained his Bachelor of Engineering degree from I.E.T., D.A.V.V., Indore (India) in 2018. He has industry experience of around 1 year. His research interests include Computer Gaming and core programming and is aiming the post-graduate studies in the field of Computer Science.