Patrolling AI Systems in Video Games

S. Balapriya, N. Srinivasan

Abstract: *Artificial Intelligence is a common element in digital video games and it is one of the most essential component in modern games. Modern games are populated with Non-Player Characters (AI Characters) that performs different activities. Realism is dominating in games and AI behavio*

rs are expected to be more realistic in games. Games that has poor unrealistic AI elements are facing heavy criticism from the player bases. Further, modern games are highly dynamic. Classical games had static environment with less or no changes in it. Such environments made implementation of AI easy and simple. In modern games, Progressive terrain generation and other such content generation methods increases the complexities of building an efficient AI for games that has many changes in real time. One of the most common AI in games is Patrolling AI especially in Shooter and Adventure Games. Patroling AI involves path finding and obstacle attack or defense. RRT algorithm and its variants are highly successful Probabilistic Determination AI that produced effective results in real time robotic movement. In order to build efficient Patrolling AI for games, a real time RRT variant called RT-RRT* algorithm was employed. The algorithm is flexible enough to add various behaviors in addition to path finding which makes it more suitable for games. The algorithm takes samples from the environment and construct the efficient path. Also the algorithm inspect the environment in run time to ensure that no moving obstacle blocks the path. In such case, it rewires and create a new path. In order to manage the dynamic obstacles, a Real Time Obstacle Handling Algorithm was designed and employed in a dynamic game environment. The algorithms inputs the obstacles types and parameters. On identifying the obstacle approaching the AI in terrain, it tells the agent to perform the needed actions. The simulations was carried out using Unity Game Engine. The model proposed helped to create efficient patrolling AI that handle two major aspects of patrolling which is Path finding and Obstacle handling. The model will be highly suitable for dynamic game environments with lots of uncertainty and emergence.*

Keywords **: Artificial Intelligence, Path Finding, Probabilistic Determination, Patrolling, RRT*, Real time environment, Video Games.**

I. INTRODUCTION

Artificial Intelligence is a process of adding intelligent abilities to machines thus empowering them to act and serve independently with respect to the formulated problem. Video Games are products where AI is employed in big scale for a long time. Many modern games has Non-Player character which is being controlled by AI systems. Video Games involves entities doing various complex activities such as player following, fighting, competing, managing, etc. AI have become one of the crucial element in designing the challenge and fun factor of games [4]

Automatic synthesis of robotic motion with various levels of tasks is highly interesting topics in Artificial Intelligence. Path finding is evidencing constant development with respect to the ever growing demands as it is being applied in most of the real time robotic applications including video games. Path finding is an ability of AI to find the best possible path in an environment and travel in that path to reach the specified destination. Observing and learning an environment, planning an optimal path from an informed source and destination is the most basic action of intelligence [6].

Video games contains a state space involves three transitional and three rotational degrees of freedom. The problem of path finding is to discover a continuous, collision - free path from an initial node to the destination node in the game world graph.

Technically, Game world is collection of polygons that comprises a playable environment. In games, generally the game world will divided into small sub portions called Maps. The objective of this approach is to simplify the complexities in managing the game world including path finding. Dividing the game world into maps helps us to reduce the search space of the game. This will help the path finder AI to easily plan and travel over the environment [5].

The two AI methods that were commonly employed in video games are Navigation Meshes and Waypoints. Navigation Mesh works based on an Algorithm that generates a discrete set of convex polygons that represents the moveable area in the map. Thus it will represent the entire map through the collection of polygons. The movement generated by Navigation Meshes will be in a Straight line travelling from one node to pre-configured adjacent node. This leads to an unrealistic movement of AI elements. Also this method of pre-configured path representation will help AI only in static environments. In case of dynamic environment, the dynamically moving obstacles cannot be identified by AI agents as they works on the pre-configured path representation [8].

On the other hand, waypoints enables developers to create a collection of nodes that could be linked together. AI will travel form source node to destination node by traveling through the available waypoint nodes in between. The routing between the nodes will be a simple sub problem.

Manuscript published on November 30, 2019.
* Correspondence Author
S. Balapriya*, Research Scholar, Sathyabama University, Chennai, s.balapriya@gmail.com.
N. Srinivasan, Professor, Department of Computer Applications, BS Abdur Rahman CRESCENT Institute of Science and Technology, Vandalur, Chennai, professorsrini@gmail.com.

This method presents an advantage of keeping certain portions of the game world not accessible to AI. The accessible portions of the game world will be comprised with waypoints. Thus this method simplifies the motion planning problem. But, this method also face the same issue as it works on the pre-configured path representation that was designed manually.

Waypoints produce good results in a static environment but fails in a constantly changing environment.

This paper deals with the specific problem of designing an efficient Patrolling AI for games. Many Action, Stealth and adventure games involves Patrol Enemy AI.

Patrol forces are more common in real world. Patrol forces is to act as a security guard and prevent intruders in their maps. The core aspect of patrolling AI is randomized movement to guard and if the target is found, moving towards it and do the necessary action. Patrolling AI are widely employed in security games, action and adventure games. Hence the major events the patrol AI handles in addition to automated movement is discovery of players' intrusion, attacking the players with the weapons or powers designed (Shoot / Fight). Also, the patrol AI should be able to combat and defend themselves, handle their environment and other important assets of the game. Patrol AI should strategize its actions based on the observation of the environment and the players' activities. Further the patrolling system should accommodate animations of the AI character with respect to the action taken.

Modern games are more contextual and incorporates AI that handles multiple forms of data like sounds. Patrol AI are often designed in such a way that they react to sound and perform intelligent actions. Building a realistic patrol AI involves handling all the above problems. Further content generation mechanics deployed in games creates huge impact in the design of AI.

Utilization of procedural content generation is increasing in games as it gives many advantages of game developers like reducing the cost of game design and development. Also, this reusable algorithm helps designers to come up with huge possibilities of environment design. Also, designers can build contextual game world with respect to the dynamically changing factors. This adds more value to dynamic content generation algorithm.

The biggest challenge in dynamic environment is the uncertainty it creates. Therefore, an AI system should be designed that efficiently handles the uncertainty the game world creates. AI to handle real time environment that were employed in robots and automated cars are addressing this problem. But in terms of efficiency, they are poor. Games require AI components that are highly dynamic and fast in handling the task [12].

In order to build an efficient Patrolling AI, an efficient movement system and onstacle modelling method has to be build that tackles the problems of dynamic environment. Without proper movement behavior, all other high level operations of Patrolling AI will not help to achieve realistic results. Hence path finding is the major element of a Patrolling AI and a poor path finder will collapse the entire patrolling behavior.

II. LITERATURE REVIEW

Motion planning problem was generally addressed by various methods like Grid based search, Interval based search, Geometric algorithms, Sampling - based algorithms.

Classical games used various search algorithms considering the game world as a 2D search space such as Dijikstra's Algorithm, bread first search algorithm, depth first search algorithm and heuristic search algorithms. The emergence of A* path finder algorithm created a huge impact.

The application of A* path finding algorithm in Strategy games and Maze environment was researched [2]. The key advantage of A* algorithm is bringing out the feature of both Uniform cost search and heuristic search. They experimented A* algorithm over various game maps and evaluated the ability of A* path finder to find a shortest path between the source and destination. The work concludes A* as a best path finder for static 2D game environment.

A detailed study on applying A* algorithm in 3D environment handles difficult path finding situations [3]. The problem region will be divided into cells. As 3D environment consist of neighboring objects of different heights, the algorithm initially masks them into cube shaped cells. Every movement in the routing was represented by transportation between the cube shaped cells. As 3D environment simulates real world physics, the algorithm rejects vertical movement. The research improvises the A* algorithm in terms of performance and efficiency. The parallel computing, reduced usage of cache and cell sampling have shown progressive results in the employment of A* start algorithm for path finding in games.

An agent – centered distributed approach was approached that allows the agents to make decisions based on the agents' knowledge about the problem [7]. The present an argument that insist building intelligence environments in their game. Certain amount of intelligence in path finding algorithms will be built in the environment in a distributed manner. This offers an intelligent map functionality that help people to handle path finding efficiently in dynamic real time environments. Smart Map functionality triggers the path finding process as and when the environment affected by changes and also help the agents to handle ongoing navigation process with respect to the changes.

Several Sampling based Planning algorithms evolved over recent times and the research work provides a detailed comparison study on all the major algorithms [9]. The paper discusses Rapidly Exploring Random Tree (RRT) family algorithms and the results over various environments. They have done a comparative study with different criteria such as path cost, run time, number of nodes. The study showed that RRT* - Smart, an extension of RRT* have found to be perform with faster convergence when compared with other variants of RRT.

Several extensions are proposed to traditional path finding algorithms. They found that traditional path finding algorithms highly focuses on finding a collision free path to reach the destination rather than generating high quality path [10].

They propose creating an algorithm that combines the good features of various novel probabilistic algorithms to generate high quality results. T-RRT was recommended to be improved by taking configuration-cost function into account in a more effective way.

Sampling based algorithms evolved and they have paved ways to various developments in the area of path finding AI. Improved A* algorithm with sampling based approach for real time robot path planning was also researched [11]. The method have shown probabilistic completeness and resulted in producing effective motion planning results. The work was concluded with evidencing a slight degrade in performance of Sampling based A* algorithm.

Every search algorithm has their own advantages and disadvantages. The major problem identified was the algorithm that produce efficient results holds poor performance and vice versa. Also adaptability of the algorithms in games became a noticeable problem. The proposed model focus on an algorithm that works well with the architecture of games. A feasible trade-off between realistic results and efficiency was aimed. Various literature's were analyzed and a working model was devised which have been explained in the next section.

III. PROBLEM STUDY

The proposed model is based on sampling based AI algorithm. Virtual environments in video games are built as a complex system. Sampling based path finding algorithms have shown efficient results in solving complex planning problems in high dimensional space with numerous degrees of freedom.

Sampling Based Planning (SBP) algorithms applied in various industrial sectors such as automated self -driving cars, aerospace, surveillance operations, industrial automation, computer animation, protein folding in biology, drug design in medicine, etc. The major advantage of using SBP algorithms is their conceptual simplicity. The research started with detailed case study on SBP algorithms.

The two sampling based algorithms that yields probabilistic completeness are Rapidly Exploring Random Trees (RRT) [13] and Probabilistic Roadmap (PRM) [14]. The two algorithms created significant development over the Sampling based algorithms. Both the algorithms have gone over several improvement over the time.

A. Probabilistic Road Map

Probabilistic Road Map (PRM) algorithm will be suitable for video games as it is a multiple – query path planning algorithm. The algorithm inputs the path planning problem that provide the state space C, initial configuration qinit and the goal to be reached qgoal. The PRM iteratively builds a graph that spans in all possible direction through Cfree – collision free spaces. PRM rather than computing the C space, it samples the environment thus it became capable of handling C-space with more than three degrees of freedom. The roadmap will be a graph G. The algorithm progress by sample free vertices in the environment and establish an edge from the nearest known vertex to the free vertex. The edge will give a collision free path to travel. The roadmap graph G contains the complete map of the sampled environment which can be queried for multiple travel. Any isolated or disconnected nodes may arise which have to sorted out by refining the roadmap dynamically.

B. Rapidly Exploring Random Trees (RRT)

RRT functions by constructing a sampling tree over the search space. The tree starts with the initial state Zinit and attempts to find a path to reach Zgoal. Every iteration, a random node Zrand will be selected and examined whether it lies in free space i.e collision free space. If yes, a nearest node will be computed as per the metric P. If the random node Zrand could be reached by Znearest as per the predefined step count, the tree will be expanded by attaching the nodes Znearest and Znode. RRT algorithm is then presented.

PRM is efficient and also it provides multi - query support. On the other hand, RRT is single- query and it is proven to be more suitable for high dimensional spaces.

Experimentation of both the pathfinder algorithms evidenced good performance in static environment. But in case of highly dynamic environment, both of them fail to find path as the sampled collision free spaces are modified in run time.

The characteristics of Patrolling AI such as Moving around a space range, Player followup requires an AI that is fast and significant to perform in the small range. Also an algorithm that is dynamic and fast to be able to react and handle the player reflexes is required. A dynamic game environment with moving characters and objects was created. The environment was designed carefully in such a way that it adds the common challenges AI pathfinders face in a dynamic environment.

A dynamic RRT* variant RT-RRT* is chosen as it provides the flexibility to add upon different patrolling events [1]. The algorithm derives from RRT* and Informed RRT* variants. The algorithm was adapted to be employed in a fast changing dynamic environment with more enemies. The algorithm focuses on tree expansion and tree rewiring technique. The randomly sampled nodes will be used in rewiring process. As the root of the tree and free nodes are changing constantly, an efficient rewiring strategy is required for the AI to reach the destination. In order to model obstacles, Real Time Obstacle Handling Algorithm was designed and employed.

C. Problem Formulation

The path finding problem can be represented as follows. The planner needs to find a path from initial state P start to Pgoal i.e P0,P1,P2,……Pgoal in a state space A. A represents a bounded work space that contains moving obstacles. In games, player will also be considered as an obstacle for AI Agents. Aobs∈ A represents all the obstacles found in the workspace. Obstacles will be modeled for efficient handling. Further, there are set of free spaces Afree that comprises spaces in A that are not occupied by obstacles i.e Afree=A\Aobs. As the environment is dynamic, Aobs and Afree sets will be updated over runtime as per the degree of dynamics in the environment. T will be pre-defined limit for the expansion of path planning from node.

The major constraints for path planning in a real time environment will be path length and travel time. A real time algorithm should be able to response promptly. As probabilistic completeness is given much importance, the probability of finding a path will be closer to 100.

In case a path cannot be found to the goal space, the planner has to find a path closer to the goal. This will be computed by a cost function that involves cost to reach values between nodes and cost to go value from initial node to goal node. The distances will be calculated using Euclidean length. This helps the planner to compute a path with minimum length. Also, the planner should take a reasonable time to compute and travel in the path.

Players' position and obstacle positions keeps on changing in the environment. Thus the planner need to handle multiple – queries. This raises the possibility of any node from A free to become A goal and A obs and any node from A obs to become A free. Also, NPC position A0 need to change with respect to the tree root node in order to keep the path in the tree traceable in all the iterations.

Our solution deals with the following major modules. Tree Expansion involves sampling free nodes, adding nodes to trees, finding neighbor nodes as in RRT*, Updating free nodes with respect to the movement of dynamic obstacles. Rewiring the random parts of the tree to the neighbor nodes or the tree root. Calculating the density of the tree, time and length of the path planning and other such constraints.

```
Algorithm
Input: $P_a$, $A_{obs}$, $P_{goal}$
Create and Initialize T with $P_a$, $Q_r$, $Q_s$
loop
Update $P_{goal}$, $P_a$, $A_{free}$ and $A_{obs}$ // initialize for rewiring / expansion
while loop that test any left for expansion and rewiring of tree do
Input: T , $Q_r$, $Q_s$, $k_{max}$, $r_s$
Sample $P_{rand}$ // random sample
$P_{closest} = \arg\min_{P\in A_{SI}} dist(P, P_{rand})$ //compute distance
if (line($P_{closest}$, $P_{rand}$) $\subset A_{free}$ )then // check node free or not
$^{A}{}_{near}$ = nearestNode($P_{rand}$, $A_{SI}$)
If( $|A_{near}| < k_{max}$ ) || ( $|Pc_{losest} - P_{rand}| > r_s$ ) then //add new node
Input: T , $P_{new}$, $P_{closest}$, $A_{near}$
$P_{min} = P_{closest}$, $c_{min}$ = cost($P_{closest}$) + dist($P_{closest}$, $P_{new}$)
for xnear $\in$ Xnear do
$c_{new}$ = cost($P_{near}$) + dist($P_{near}$, $P_{new}$)//calculate cost
if $c_{new} < c_{min}$ and line($P_{near}$,$P_{new}$) $\in A_{free}$ then
$c_{min} = c_{new}$, $P_{min} = P_{near}$
$V_T \leftarrow V_T \cup \{P_{new}\}$, $E_T \leftarrow E_T \cup \{P_{min}, P_{new}\}$//attach the node
Push $P_{rand}$ to the first of $Q_r$
else
Push $P_{closest}$ to the first of $Q_r$
Input: $Q_r$, T
repeat
$P_r$=PopFirstNode($Q_r$), $A_{near}$=nearestNode($P_r$, $A_{SI}$)
for $P_{near} \in P_{near}$ do
$c_{old}$=cost($P_{near}$), $c_{new}$=cost($P_r$)+dist($P_r$, $P_{near}$)
If( $c_{new} < c_{old}$ and line($P_r$, $P_{near}$) )$\in P_{free}$ then
$E_T \leftarrow (E_T \setminus\{Parent(P_{near}), P_{near}\})\cup\{P_r, P_{near}\}$
Push $P_{near}$ to the end of $Q_r$
until there is time or $Q_r$ is empty
Input: T , $P_{goal}$
Check the status of tree. If (current node is same as $P_{goal}$) then
Modify traversal path from $P_{goal}$ to $P_0$ if (traversal path is rewired)
($P_0$, ..., $P_k$) $\leftarrow$ ($P_0$, ..., $P_{goal}$)
else
```

```
for $P_i \in (P_1, P_2, ..., P_k)$ do
$P_i$=child node of $P_{i-1}$ with min $f_c$=cost($P_c$)+H($P_c$)
if ($P_i$ is leaf node) || ($P_i$ children are blocked) then
($P'_0$, ..., $P'_k$) ← ($P'_0$, ..., $P'_i$)
Block $P_i$ and Break;
Update the discovered optimal path with ($P'_0$, ..., $P'_k$) if needed
($P_0$, ..., $P_k$) ← decide to be in $P_0$ or follow optimal path
return ($P_0$, ..., $P_k$)
if ($P_a$ is close to $P_0$ )then
$P_0$ ← $P_1$
Start Moving the Agent to $P_0$
end loop
push the smart map data $M_n$
loop for all $M_i$ in $M_n$
Scan each $M_i$ (Properties, attacks, priority, etc)
identify the approaching dynamic obstacle (enemy) $E_i$.
Push the Enemy Behavior $B_i$.
Push the action lists $X_i$.
Push the Weapons List $W_i$.
Pick the actions ($T_i$) corresponding to the approaching enemy $E_i$.
end loop
```

D. Real Time Obstacle Handling Algorithm

An Obstacle Modeling Algorithm called Real Time Obstacle Handling Algorithm was employed. The environment contains of grid based map that will help to identify the nearby nodes easily. Also the environment contains various critical information about the dynamic obstacles involved in the environment such as transformation of the obstacles, type of obstacles and required actions on the obstacles. As patrolling involves securing areas, covering or attacking against different dynamic enemies, an environment is proposed that contains the necessary details to be inputted to the path finding agent.

A separate obstacle modeling unit was created and maintained to store the key properties of the environment like the obstacle initial position, type, projected events against the obstacle, desired challenge range, actions need to be delivered, etc. The environment was configured with the necessary data and passed to the agent. In case of an open world game, multiple game objects move around the world. The smart environment proposed will hold the dynamic information to manage the environment and enable fast reaction from AI agents.

On facing obstacles, the AI approaches it, inspect and decide the actions need to be taken based on the type of obstacle with the help of the smart map data.

In order to create a Smart Map, a dedicated data structure was created that contains references to all the objects in the Map M. The Map data will contain all the obstacles/enemies E added in the environment. The map will only contain the spawn point of each and every dynamic obstacle. The algorithm will inspect and identify the moving obstacles during run-time. AI agents should generate meaningful and sensible responses to the approaching obstacle.

All the obstacles and traps involved in the world were collected and referred through Map M. Each obstacle is sampled in such a way that it will contain details such as its type and rank (that delivers the level of difficulty). Every Enemy / Obstacle require the AI agent to perform different actions or attacks. This obstacle sampling helps the agent to identify the nature of the obstacle exactly. Agent has to learn the respective behaviors which can be achieved through several learning algorithms.

Various maps and deployed the algorithm and tested against various obstacles. Unity 3D Game Engine with C# Scripting is the platform using which it was experimented.

IV. OBSERVED RESULTS

The results obtained from the simulation carried out demonstrates high flexibility of AI agents on handling the obstacles. The model used proved to be working as it handle both the major tasks of a Patrol AI that is, Path finding and Obstacle Modeling. The Real Time Obstacle Handling Algorithm employed was tested with different environments and with different types of Obstacles. As Games involves Dynamic Obstacles of different types, a detailed list of different obstacles were created and incorporated in the smart Map M along with the other environmental details.

Obstacle (E_i)	Obstacles Action List (B_i)	Agent Action List (T_i)
Door	Idle	Open, Close, Break
Crate	Idle	Jump, Break
Animal	Attack by hit	Shoot, Hit with Knife

Soldier	Attack by Gun & Hit	Defence, Shoot, Fight with Weapons (Pick weapons form the list and appropriateness of the enemy weapon), Attack with Punches and Kicks
Zombie	Attack by Hit	Defence, Shoot

The following figures shows the simulation results. The tree generated is re-wired every time it faces a dynamic obstacle. Also, every time a dynamic obstacle was detected, the smart map data was scanned and necessary actions were performed.

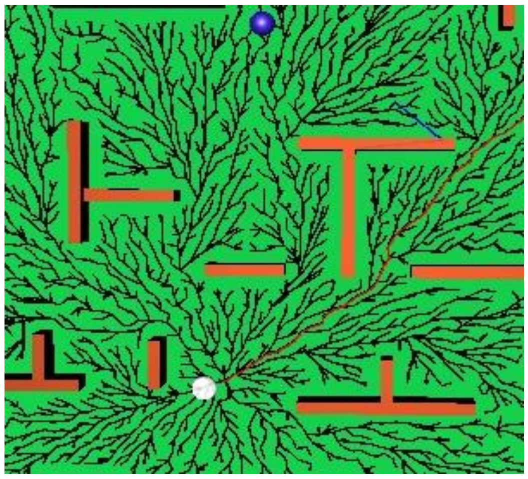
Figure 1: Agent movement

V. ANALYSIS

The two major criteria that have to be considered importantly are the iteration count/time for path finding, and obstacle modeling time. Several simulations were carried out and results delivered proved that the AI takes less time to approach and identify the obstacle. In addition to reduced time, the simulations also showed that the path cost to be low. Also, the simulation showed that the algorithm is complete as it delivers optimal results over all the tests. The path discovered to reach and track the obstacles were realistic and found to be the shortest. As Iteration time found to me small in terms of small range with moderately dynamic environment, the count keep increasing as the range grows. As patrolling involves covering a small portion of the terrain this issue does not make big cost for the game.

The range of the Patrolling agent will be defined at the beginning and it will help the algorithm to make informed decisions on iteration count thus produce better outcomes. Using RT-RRT* at the core for path planning requires more memory capacity in order to handle the expanding tree. Also with respect to the environment, the tree size will be increased and also processing the whole tree repeatedly cause more processing time. Further, obstacle modeling incurs its own challenges such as delayed responses caused by obstacle analysis time.An in-depth analysis was carried out that delivers the following results.

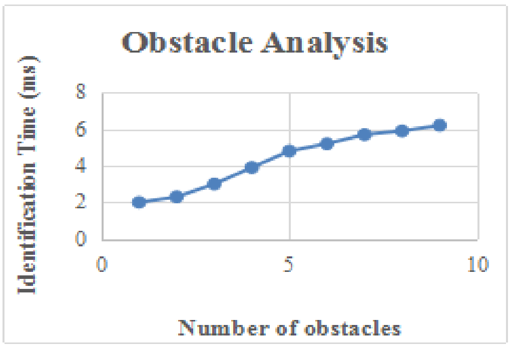

Figure 2: Obstacle Analysis

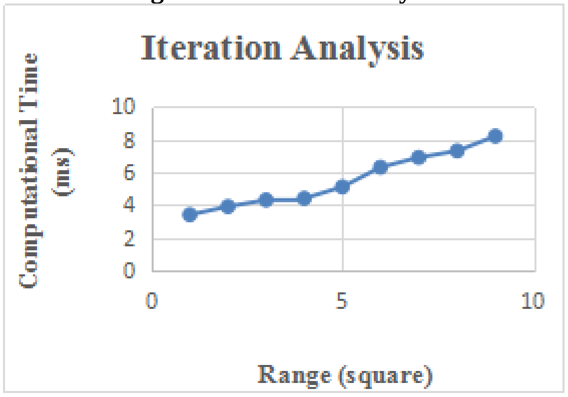

Figure 3: Iteration analysis

Obstacle Modeling Time can be controlled by improving the Smart Map Data Structure. Collecting the range of obstacles and scaling down the number will help the algorithm perform better. As, the algorithm is flexible enough to control over the time and range, good performance is achieved in the conducted simulations.

VI. CONCLUSION

Patrolling AI need to be smart and pro active as they are one of the major game element that decides the challenge factor in action and similar genres of game. Path finding the core behavior of patrolling AI have undergone various methods. Considering a dynamic environment with lots of uncertainty and high level of variable emergence, most of the classical algorithms were not performing as expected. As games are becoming more complex and realistic, the need to build a dynamic path finder is unavoidable and this work have took one such algorithm and adapted it into game environment. Path finding method employed in this work delivered satisfying results. Emergence pays a major role in games and obstacle handling is the crucial part of an Patrolling AI. A real time obstacle handling algorithm that inspects, analysis the dynamic elements in the environment was designed and implemented. It also helps the algorithm to decide the actions that can be performed against the approaching obstacle. Various simulations were carried out using Unity and results shows that the AI agents movement were highly realistic unlike the jerky movements resulted due to the previous methods. Both the path finding and obstacle handling methods were found to be performing well during the analysis phase.

As it is using a real time algorithm to handle motion planning and AI movement, consumption of resources would be comparatively high.

REFERENCES

1. Naderi, K., Rajamäki, J., & Hämäläinen, P. (2015, November). RT-RRT*: A real-time path planning algorithm based on RRT. In Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games (pp. 113-118). ACM.
2. Barnouti, N. H., Al-Dabbagh, S. S. M., & Naser, M. A. S. (2016). Pathfinding in strategy games and maze solving using A search algorithm. Journal of Computer and Communications, 4(11), 15.
3. Niu, L., & Zhuo, G. (2008). An improved real 3D A* algorithm for difficult path finding situation. Proceeding of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 37.
4. Cui, X., & Shi, H. (2011). A*-based pathfinding in modern computer games. International Journal of Computer Science and Network Security, 11(1), 125-130.
5. Kamphuis, A., Pettre, J., Overmars, M. H., & Laumond, J. P. (2005). Path finding for the animation of walking characters.
6. Chen, A., & Ji, Z. (2005). Path finding under uncertainty. Journal of advanced transportation, 39(1), 19-37.
7. Botea, A., Bouzy, B., Buro, M., Bauckhage, C., & Nau, D. (2013). Pathfinding in games. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
8. Graham, R., McCabe, H., & Sheridan, S. (2003). Pathfinding in computer games. The ITB Journal, 4(2), 6.
9. Noreen, I., Khan, A., & Habib, Z. (2016). Optimal path planning using RRT* based approaches: a survey and future directions. Int. J. Adv. Comput. Sci. Appl, 7(11), 97-107.
10. Devaurs, D. (2014). Extensions of sampling-based approaches to path planning in complex cost spaces: applications to robotics and structural biology (Doctoral dissertation, INP DE TOULOUSE).
11. Persson, S. M., & Sharf, I. (2014). Sampling-based A* algorithm for robot path-planning. The International Journal of Robotics Research, 33(13), 1683-1708
12. Lipovetzky, N., Ramirez, M., & Geffner, H. (2015, April). Classical planning algorithms on the atari video games. In Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence.
13. LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning.
14. Kavraki, L. E., Kolountzakis, M. N., & Latombe, J. C. (1996, April). Analysis of probabilistic roadmaps for path planning. In Proceedings of IEEE International Conference on Robotics and Automation (Vol. 4, pp. 3020-3025). IEEE.