wjert, 2021, Vol. 8, Issue 1, 141-163. Review Article ISSN 2454-695X

World Journal of Engineering Research and Technology

WJERT

www.wjert.org SJIF Impact Factor: 5.924

AI IN GAMES – A REVIEW

***Prof. Sanjay Deshmukh, Preyas Hanche, Akash Dubey, Nishant Desai and Meghaj Agrawal**

[1]*Assistant Professor NMIMS MPSTME* Computer Department Mumbai, India – 400056.

[2,3,4,5]*B.Tech Student NMIMS MPSTME* Computer Department Mumbai, India – 400056.

Article Received on 09/11/2021 **Article Revised on 29/11/2021** **Article Accepted on 19/12/2021**

***Corresponding Author**
Prof. Sanjay Deshmukh
Assistant Professor NMIMS MPSTME Computer Department Mumbai, India – 400056.

ABSTRACT

Artificial Intelligence has always been at the core of interactive and simulated environments for a long time but the development in these fields has always been constrained to a single monolithic process that governs all logic within these environments. Simulated Environments are often integrated with centralized artificial intelligence systems that control objects, characters within the environment. We aim to highlight a new system involving the implementation of a decentralized Artificial Intelligence system that acts and feels like it controls all objects, characters independently while practically it does not. The system will revolve around pathfinding techniques, perception systems amongst other key components that are integral to these environments. This implementation will not only enable cutting down on compute cycles, but also steering towards non-intensive multi core processing while adding a sense of realism. Thus there will be increased overall entertainment and provide an immersive environment.

***Index Terms*—Keywords:** Behaviour Tree, Artificial Intelli- gence, Path-finding, Perception Systems, Game Engine, Game Development, Decentralized Functions, NPC Systems.

I. INTRODUCTION

The computer game industry is one that involves a wide range of roles that manage the turn of events, showcasing, and adaptation of computer games. Because of the enormous advancements in computer based technologies resulting in quicker CPU's, and committed co-processors and the reduced costs, the industry's growth shows no signs of slowing down.

This growth has led to more development and innovation in terms of specialized areas like AI, graphics and mechanics. The game industry has surpassed all other entertainment industries since a while now. All this has resulted in more and more game engines being developed. Since the early days of the game industry, there has always been a never ending demand for better games than the previous ones. To cater to such needs of the users, new game engines keep on being developed, while some popular engines have kept evolving. Therefore, today we see games that have extremely complex mechanics and very realistic graphics. Many popular game engines have become free to use, encouraging independent development and building of a huge community of developers. This has also enabled more people to conduct research on topics such as content generation, artificial intelligence, computer graphics and many more.

Artificial intelligence has become an integral part of most video game genres and is key to the overall experience of the game. Non player characters (NPCs) have a significant role in modern games. NPCs provide a way to assign tasks and give quests to the player. Furthermore, NPCs contribute to the progression of the game's story. The purpose of this paper is to provide a literature review of Behavior Trees and its optimizations for the development of AI NPC controlling systems, the various engines available for developing AI based games, as well as some of the popular path-finding methods adopted into professionally developed games.

II. LITERATURE SURVEY AI Concepts used in Gaming

BTs (behavior tree) have nodes which can be categorized into root, control flows nodes or execution nodes. The parent nodes and children nodes are connected to each other hierarchically. The root has no parents and decisions work their way down from the roots where control flows from left to right. All executions start from roots where on every tick, which is the signal sent to the child nodes and these return values to the parent nodes along with any execution values. Behaviour Tree plays a major role in Unreal Engine, which we have chosen as our primary development engine. BT in Unreal Engine consists of two key components:

- Blackboard – which is used as the memory of the AI and stores data as a dictionary. Informed decisions can be taken with the help of the data which can be transferred around the BT.
- Behaviour Tree – Decision making program that acts as the head and makes the

important decisions.

- The BT consists of some key components:
- Root – This is a unique part of the decision trees where it is considered the starting point. It has a primary connection that usually connects to the decision-making nodes. This node has no properties on its own but is used to set up the Blackboard Assets to the tree.
- Composite – These nodes define the root of the branch and provide the base rules of the branch.
- Task – These are the decision and execution making nodes and do not have an output connection and areconsidered the leaves of the decision tree.
- Decorator – The IF conditions of the BTs, these provide the pathway to the conditionals of the trees.
- Service – Attach to composite nodes where these are being executed when the branch is being utilized.
- These provide checks and update any Blackboard variables.

The utilisation of BTs help provide a visual understanding of decision making, event actions and reactions taking the Behaviour through different interactions with the tree and all relating variables. Unreal Engine makes interesting use of BTs with additional nodes such as decorators, composite nodes etc., and helps provide an additional dimension to the objects within the simulated environment such as sight, sound.

Marcotte R and Hamilton HJ in,[2] delve into the use of behavioural trees and how it is modelled around an object that acts as a warrior and its interactions with the environments, in different scenarios. Behaviour Trees are an important part of most current gaming titles which supplanted the previously used FSMs. These BTs run on a game execution loop that makes decisions relating to sub-engines for physics, artificial intelligence, graphics, etc.

It further talks about the status code that a behavior tree returns to the parent component after execution. These codes can either be success, failure, running or error. Success is returned when the task is performed successfully, whereas failure is returned when the task is completed but unsuccessfully. If a task to be performed is incomplete then running status code is returned. When there is an error while processing or completing the task, the error status code is returned. The error could even be a programming error. It also expands upon

some other components such as reference, action, condition and control flow. Reference component is used to refer to another behavior tree linked to the first one. The action component performs an action and subsequently changes entit2 state. According to the execution of the action, it returns corresponding status code. Condition componentcontains a boolean question which can be true or false. Status code is returned based on this value of the component. Control flow is used to group a set of components together. An order can be given which tells which components in a group can be executed.

There is also a comparison made between different behavior tree editors available. Of the editors used for comparison, Unreal Engine's editor has more number of features such as extensibility, specific BT debugger. This makes Unreal Engine one of the better editors for constructing BTs.

Sekhavat, Yoones,[3] explain how Behaviour Trees can be used in decision making as well a be advantageous with respect to scalability in video games. They also explain how a Behaviour Tree is made and what the structure is composed of.

Finite State Machines are becoming less and less feasible since it makes the use of states and the transitions between one another to handle individual behaviours. As games become more and more complex, the directed graph that is used to represent the FSM gets bigger and bigger which leads to scalability issues. Even after applying techniques like fuzzy logic and neural networks, the problem has not been completely solved. In this case, Behaviour 3rees can be used, in which an agent performsa DFS and executes the leaf node. Behaviour Tree Structure: It is basically organized as a di- rected tree with nodes and edges. A node which is not a leaf can either be a sequence or selector node. Selector node's job is to run any one child node possible. In a sequence node, all the children nodes are executed one after the other. The children nodes may consist of either actions or conditions. These actions may be running an animation, or transitioning to a different character state. A condition can be something like Line Of Sight, where if the character is able to see some object, an action will be run correspondingly. Other nodes may include parallel and decorator nodes. A decorator node only has one child node. It basically acts as a controller for the child node which allows it to make the decision till when the child node should be executed. A parallel node is used when some action has to be executed along with other actions simultaneously. For example, a character when given a movement input should start moving in the global coordinate space, and should have a corresponding animation running at the same time.

There 4re also Priority nodes and Custom Action nodes that are added in some trees, depending on the software. A blackboard is used to separate the data from the behaviours and can be used by a task node.

[3] Explains how Behaviour Trees can be formed:

1. Genetic Algorithms: For this method, an initial set of BTsare required. Operators are then used to make better trees by encoding the behaviours for different parts and then are used to create new behaviours. A function is used to check the new trees. The syntax can be checked either by context-free grammar or by applying the AND-OR graph to the sequence and selector nodes. This method does not necessarily create better behaviour trees.
2. Case-Based Reasoning: For this method, a knowledge base is required. A querying functionality is used for a particular node so that the best behaviour can be chosen from similar cases in the knowledge base. Hence, using CBR we can build the behaviour of an NPC at run-time.
3. Q-Learning: This method makes it convenient to de- termine when is the optimal time to execute a certain AI function. It is useful for optimising the Behaviour Trees. But the disadvantage of this method is that it has performance issues.
4. Learning Examples: As the name states, examples are used as knowledge on the basis of which the behaviour is set. These examples are represented as a sequence of cases in which the pair of actions and observations are stored along with a similarity metric.
5. BT Generation: The above cases were for automatic Behaviour Tree generation. For manual generation we canuse authoring.
6. We can also modify a standard Behaviour Tree with extra knowledge. The techniques used to do this are as follows:
7. Genetic Algorithms: This integrates both Finite State Ma- chines with Behaviour Trees. This is done since standard BTs do not have internal states. Another method is toadd a parameter to the parent behaviour to store the current state. One issue with this technique is that an additional time manager would be required to handlethese parameters.
8. Emotional Factor: In order to emulate human behaviour, emotions must be incorporated so that behaviour can be influenced according to the current emotional stateof the AI. To implement this, an emotional selector is used, which uses risk perception, time and planning as weights and are used as importance factors which are multiplied with their respective weights and summed to get the overall weight. This can be used by the AI

designer to change the behaviour accordingly.

9. Behaviour Tagging: To prevent complexities in using just a single behaviour tree structure, behaviour tags can be used to keep constant certain features of a character, and behaviour messages are used to change it.
10. Hinted Execution Behaviour Trees: It allows designers to give more control and flexibility by giving hints to the AI, which can decide for itself if it wants to do what the hint says or not. This way the designers do not have to redo the whole AI system for small changes.
11. Dynamic Behaviour Trees: Nodes in this case have queries, which are then substituted for behaviours at runtime. This way, more abstract trees can be made instead of implementing all the details and functionalities.
12. Self-Validating Behaviour Trees: This allows to check for any issues with the BTs when they are being designed. Parameters are present so that the state can be stored, compared to standard BTs which do not take any param- eters.

The focus of,[4] is on the use of Behaviour Trees and the addition of emotional factors in BTs. Emotions are an important part of human beings; they play a very important role in decision making and affect the decision-making process depending on the emotions faced in a particular scenario. With the addition of emotions in games, the same object would interact differently in the same scenario. But, adding real emotions in a simulated environment is a very difficult thing to do, to circumvent these emotions are referred to as values for each NPC or object within the scenario. Emotions such as happiness or sadness among other emotions in simulated environments depends on the history of the object, how it has interacted with other objects so far and this decides the interactions in the future. Apart from the execution flags such as success, failure or running, additional flags were added so as to signify emotions within the behaviour trees or decision-making process.

With the use of emoBT, Risk perception and Time Discount were given extra importance in the entire decision-making tree. Emotions and Decision-making play into the use of positive and negative options that affect how actions take place. Risk Perceptions play an important role in how decisions are made for example, a human would be averse to taking risks if they are sad whereas they take more risks if they are happy and leading content lives. Time also plays into the emotional state of a person where people who are sad are less patient compared to others. This trend also applied to planning where happy people carefully plan their actions out as opposed to sad people who stick to shorter tasks with quicker results.

The implementation of the aforementioned trends in video games is dependent on the values associated with these emotions. Taking an example of a shooting game, a character depending on the level of stress would either call for reinforcements, shoot the player or in high stress situations throw grenades. As emotions play an important role in all situations, it is being readily adopted into games and provides a sense of realism.

Several new concepts are explored in,[5] like smart objects and smart areas, as well as how behaviour objects implement Object Oriented Programming.

- ✓ **Behaviour Objects:** These entities consist of behaviours which are analogous to code, data and the logic part which is called the brain. These objects can be assigned to Non-Playable Characters (NPC). The data can be used to change the behaviour of the NPC according to the surrounding context and multiple NPCs using the same Behaviour Objects can communicate with each other. The brain can influence the NPC's behaviour by altering the data accordingly. BO data can be split into two categories: environment data and state. Environment data relates more to how the NPC or a certain object which is given the BO is supposed to react in that particular context or surrounding and can communicate with other objects whereas state data tells the object information about itself, for example if an AI is busy or idle and has to run the corresponding animation.
- ✓ **Smart Objects:** These are entities that handle minute behaviours for items in the game. The items are mostly inanimate three-dimensional models that the character and other NPCs interact with. They do not have their own brain as compared to Behaviour Objects and act only when an external stimulus is given by any live character. For example, chair animations when a character decides to sit on them. An extension to this concept is Navigation Smart Objects, which is essentially an item that acts as a bridge between two areas in the map or the level when the player interacts with it. For example, when a character opens a door, the animation should run along with the subsequent loading of the new area into the memory and rendering it, along with timing it to the end of the door animation.
- ✓ **Smart Areas:** These entities are connected to entire parts of the map. This may not be a visible or a tangible object for the character, but is mainly used to change certain features and behaviours over the part of the map it is held over. For example, a chair placed in a safe area of the map should make the character show a more relaxing posture, since the chair has a parent-child relationship with respect to the smart area. Whereas, in a dangerous area of the map, if the character sits on a chair, he or she should have a more

alert posture. Hence, the chair performs all the low-level functionality of the animations and subsequent position changes while the smart areas handle the high- level behaviour of subtle emotion changes.[6] has elaborated the development of a racing simula- tor using various AI techniques. The background literature explains the concept of bots which are AI components that imitate human behaviour. This addition leads to a more immersive experience. The general AI algorithms used in the early development stages were finite state machines, fuzzy state machines and decision trees. Another new process was created in the form of an SDK called Flexbot in which multiple bots were controlled to emulate human behaviour. Next, a racing game called Racer is used to explain the slight adjustments an AI needs to make so that it does not break immersion by taking decisions too perfectly and giving it human-like characteristics. These slight adjustments were designed in the following manner:

- ✓ **Waypoint System with Vector Calculations:** The way- points are essentially important positions on the track in the 3D coordinate system. These are stored in a sorted list using a List class. These points determine the AI car's driving as the car needs to pass through these waypoints iteratively. In the case of a curve between two points on the track, vector calculation is done using the current waypoint as well as the current position and direction of the car. The output vector is then used to control the steering and the acceleration/braking accordingly. The disadvantage of this system was that the input and output vector did not have any linear relationship, and the car could not be controlled using this method.
- ✓ **Conditional Monitoring System:** This system was re- quired since the first method was incompatible for the car movement mechanics. This monitoring system is able to adjust the values of steering and acceleration to produce a more optimal traversal for the car.
- ✓ **Artificial Environment Perception:** This method uses trigger detection, which is essentially a mechanism to detect when a game object is near another object. This is built within the car logic, so that when the car is neat some kind of obstacle, it brakes and steers accordingly. This is implemented along with a trigger area, which is basically a rectangular mesh around the car which acts as the detector. Another method to implement this mechanism is using ray-casting. It involves making a line from an object towards the direction the car is moving, which is then used to check if the object is going to hit the car or not. But this function can only check one direction.

In,[7] comparison of two of the most prominent artificial intelligence implementation

mechanisms, Fuzzy State Machines (FuSM) and Emotional Behaviour Trees (EBT) is shown. The two implementations are contrasted, pointing out any similarities, working models and differences. These mechanisms are used in a massively online multiplayer game which controls NPC decisions and any tasks undertaken by these NPCs.

This paper focuses on Multiplayer Online Battle Arena (MOBA) styled games as this genre of games has become massively popular as of late with most prominent engines adding support of Network Engines and other assets used to develop these environments.

FSM played an important role in most games and were considered the best technique to control decision-making in these simulated environments. Modern games need a high efficiency computational system which controls assets and allocation of assets in situations which with the usage of FSM could be simplified, hence FSMs were used for fast action, strategy games where advanced and adaptive objects were not required. Fuzzy logic is essentially a form of logic where the variable values can be any real number from 0 to 1. This concept can be used in cases where a fact is neither true nor completely false.
All these systems contrasted together for usage in a MOBA style game where each method had their pros and cons. The FSM while simple and easy to compute would become difficult to manage larger systems with a huge amount of simultaneous computations. Fuzzy Logic worked great in these non-deterministic scenarios but would take longer to develop and in the absence of expert developers could lead to unwanted actions being taken. Behaviour Trees were finally chosen due to their easy scalability, modularity and extensibility but with the usage of BTs various others control nodes could also be added leading to emotional controllers and reactors in particular scenarios. This led to testing these observed methodologies in various scenarios within the simulated scenarios. The implementation of FuSM leads to more user fighting which is one of the most important parts of the game, however with the usage of BTs and the cooperatively used Emotional nodes, it leads to a higher level of user interaction which helps the player immerse themselves in the game.[8] contrasts the existing AI system developed in a game called StarCraft. StarCraft is a real-time strategy game that has to make decisions based on the decisions made by the user and react to make it a challenging situation. The original AI used controls not only the attack and defense patterns but also all interactions inherent to the RTS game, like building placement, skills used by player, upgrades if any along with player characteristics. In this case study, primary focus was just the combat AI. Comparisons done

throughout the paper were done with the original AI and the modified AI that is used to control the combat objects in game. Changes done to the behaviour tree try to highlight NPC coordination in a decentralized manner such that each NPC does not have to communicate with each other but act in a clustered manner. This approach can lead to a rewarding experience while also providing a challenging experience to players.

The new implementation uses an implicit type of communication where objects do not directly interact with each other but leave environmental highlights that can be read by the other objects. Due to this system, the NPCs interact with each other and any actions that take place propagate within all the objects. Compared to the original AI system which is basically defined as cannon fodder, the new system gave a sense of realism to the NPCs which led the players to believe that each NPC was independently controlled. The purpose of creating a decentralised AI control system while not independently controlling each facet of the attack system was concluded as a successful implementation.

Pathfinding and related Systems

[9] talks about A* algorithm and its use in pathfinding in games. The search space needs to be upto a certain size forthe algorithm to work in an acceptable time period. Hence, some of the optimizations of A* are discussed thereafter. Hierarchical pathfinding A* involves breaking up the game world into divisions, hierarchically. A feasible path is then found from the division containing the source point through the different divisions. The path is searched in an abstract level as this improves the speed. If an optimal path is found then the path is completed by traversing through it and reaching the destination. Hierarchical pathfinding hence reduces the complexity of the problem.

Navigation Mesh or NavMesh, is another popular method to improve pathfinding. Here the walkable path on the map is covered by a number of convex polygons. At each step, the character has to determine the next polygon to travel to, until the destination is in the same polygon as the character. NavMesh finds a near optimal solution to the problem which is better than many other pathfinding techniques.

Using a heuristic function in A* that overestimates the cost of travel by a small amount leads to less number of nodes explored to reach solution. Hence, the time required for the search is reduced and a feasible path is given. A simple implementation of A* algorithm consumes a significant amount of resources in large world levels. To help reduce the

required memory, a fixed portion of memory can be allocated to the process. If the allocated memory gets usedup during the search, a new buffer of flexible memory should be created in order to allow the search to complete. Another method to reduce memory usage is to implement IterativeDeepening A*, where a path is abandoned when its total cost exceeds that of a fixed threshold. In case no path reaches the destination, the threshold is increased and another search is done.

Next, pathfinding methods of popular games Age of Empires and Civilization V were mentioned which are A* and placement of hexagonal tiles. Both the games suffer from bad pathfinding and by comparison, A* is more effective when used in first person shooter games or games where the number of simultaneously moving agents are less. Finally, a comparison is made between waypoint technique and Navmesh for navigation, with Navmesh proving to be more efficient than the waypoint system.

Grids.[10] represent the world map with the help of vertices connected to edges that make up the terrain. Performance of the pathfinding algorithm depends upon how the grid is oriented. There are two kinds of grids:

1) Regular Grids: These include regular polygons . The shapes used are squares, hexagons and triangles to makea grid out of our world map. This process is calledtessellation. Regular grid includes:

- ✓ 2D Square Grid: This is the most common type of grid used in games. The benefit of using this type of grid is that we can use another algorithm called Jump Point Search algorithm (JPS) instead of the regular A* algorithm. This algorithm exploits the fact that the grid is regular. Unlike A*, we do not need to search all possible paths and instead pick a single optimal path, which will reduce the time complexity drastically. It is approximately 10 times faster than A* and has a smaller memory overhead.
- ✓ 2D Hexagonal Grid: This type of grid is less com- mon compared to square grid, but has smaller time and memory complexities. There have been studies which revealed that this grid produces better results in small and medium maps but requires more timein larger maps for example, open world maps.
- ✓ 2D Triangular Grid: This is the least used 2D grid, compared to Square and Hexagonal.
- ✓ 3D Cubic Grid: This type of grid is based on a 3D environment. There are some limitations in this typeas the algorithm used in this grid consumes more time, and instead of all 26 dimensional directions it can only use six neighborhood cells to navigate to the

goal cell.

2) **Irregular Grids:** The techniques used are

- ✓ Visibility Graph: A simple algorithm is used to find an efficient path through a series of obstacles. It uses a geometry-based approach which is constructed from the concept of how much the target is able to see the obstacles, instead of cell-based methods seen in regular grids. Although this method did not create optimal solutions, it was faster than othertraditional approaches.
- ✓ Mesh Navigation: This mesh represents all the ter- rain which is walkable by the character. Shapes like triangles or even different types of polygons can be used to represent this mesh. HAA is an extension of the A algorithm that can be deployed in this type. A path for agents with different sizes and terrain overcoming capabilities can be created with this algorithm. The disadvantage of this technique is that it does not produce optimal paths. One ad-vantage of using this algorithm is that it reduces the computational effort required for low-level searches exponentially.

Hierarchical Techniques: To overcome the constraints of requiring lots of memory space, we can use hierarchical techniques. Hierarchical techniques discretize the continuous environment, thereby reducing this memory space problem. A more accurate representation can be produced by increasing data gathering levels.

1) **Probabilistic Road Maps:** It uses a pure pursuit al- gorithm, which is a tracking algorithm employed in navigation problems. The algorithm calculates the curve (or the arc) that will move a vehicle from its current position to the desired destination. The limitation of this technique is that this framework can only handle static obstacles and is not compatible with any moving ob- jects. Dynamic and real-time obstacles make the motion planning more complicated.

2) **Quadtrees:** Quadtrees are generated by repeatedly divid- ing each grid cell in a square grid into four equally sized grid cells. The division process finishes either when it reaches some smallest allowable size or a grid cell contains no obstacles. The algorithm mentioned in this case takes regions of source and destination and limits the search space according to the subtree. The technique implemented by this algorithm can easily handle bigger crowds and also adjust itself to dynamic and ariable environments. A rough grid is used to represent thehigh- level distribution of the crowd and motion can be compared to fluid dynamics, obtaining a solution for path planning and congestion avoidance.

S Balapriya, N Srinivasan in,[11] focus on the patrolling systems used in games and how different path-finding al- gorithms cater to the movement used in static or dynamic environments. Path-finding refers to the process of finding a path between two locations without interfering with the environment while also working within the confines of the simulated environment. This basically refers to the movement and path of objects taken from source to destination without crashing into other objects. As path-finding works on a 2D area whereas games are considered a 3D environment, this leads to the problem of finding a continuous, collision free path involving three rotational and translational degrees of freedom. Due to this problem, games divide the area into maps which makes it easy for the AI to navigate. The most commonly used modes of navigation involve nav-mesh. As movement is limited to polygons, it leads to the unrealistic and unsteady movement. The other method used is way-points - when nodes are linked from source to destination. Although this leads to smooth movement from source to destination, it is limited and used in staticenvironments.

As patrolling systems involve an ever-changing environment, the aforementioned path-finding systems do not work well, also the added complexity of reactions provided to these characters in games adds a layer of complexity which can be solved with these methods. The proposed models discussed in this paper involve a sampling based planning algorithm(SBP), an efficient algorithm that can solve complex problems that have a high degree of freedom. The SBP algorithms discussedare as follows:

1) Probabilistic Road Map: This is a multiple query path finding algorithm that iteratively builds a graph plotting all the possible movement spaces in all possibledirections.

2) Rapidly Exploring Random Tree: This algorithm constructs a sampling tree over the search space. This also works iteratively where on each iteration a random node is selected and verified if it lies in a collision free space. After the initial check of the availability of that node, the distance is verified from source and destination. If a particular node provides the least cost for the path it is selected as the next predefined step.

For activities such as patrolling, the algorithm used must be high functioning and decisive so as to provide a goal plan for the AI to follow.

In[12] Unreal Engine is used to implement AI to simulate a pedestrian in a driving simulator.

The authors have created a pedestrian AI class inside Unreal Engine which has the

components: Movement, AI Perception, AI Controller, ID Model and Root Collision. The movement in the 3D space inside the navigation zones is managed by the movement component. The root collision is used to check if obstacles or pedestrians are in the way and depending on that information, the route is made. The movement component changes the direction and magnitude of movement of AI in real time. The brain component present inside the AI controller has a behavior tree which is used to describe the logic of the AI.

The service starts out with the traffic signal operation where the AI has a certain set possibility of ignoring the red light. When this occurs, the pedestrian class is invoked to run away from danger. The system also creates the illusion of a crowded city by bunching up a set number of pedestrians in a small block of the city, instead of populating the whole city and using up resources.

Along with the mentioned components, they have also used an Environment Query System which is used to choose the optimal route to a destination node by adding a set of points in the world map in the direction of the player and filtering out the points which either are too far or do not have a path to travel by, which is done by EQS-Queries.

Gaming Industry

Hubert Zhukovsky in[13] highlights the importance of the gaming industry as a whole. This industry spans fields that equate to the fields combined by music, film and book markets. This growing field is estimated at a value of 100 billion dollars and this only will grow in the future. Apart from the massive scale of the industry, it also combines all the other industries as games have parts that are movies, songs and are also interactive in nature. To make the development of games easier, the advent of game engines provides a very conducive environment which makes the development of games and controlling all the assets in the game easier to manage.

Most game engines provide a basic set of tools such as:

- ✓ Rendering Engine – a module that interacts and controls the rendering features of the entire simulated environ- ment.
- ✓ Physics Engine – module responsible for forcing the laws of physics on the objects within the environments.

Apart from these engines, most engines also have a network engine which connects the data and exchange of data and the Audio Engine which controls sound used in the environment

and how sound is produced.[13] Focuses on CryEngine, a massively popular engine that was used to develop very intensive games such as Crysis and Unity Engine. Unreal and Unity Engine make up a massive part of the market due their ease of access, user assets and scale of these environments. These engines are calculated on the number of objects that it can handle, how the modules interact with each other and the efficiency of these engines. From the tests conducted, CryEngine was seen to have achieved better frame rates at low level of objects where efficiency deteriorated when the number of objects were increased whereas Unity Engine provided a level of consistency, granted that the efficiency drops when the number of objects is increased but this drop is still better than the drop of frames, modules functioning seen in CryEngine.

The analysis and comparative study of multiple game en- gines and the specific features of some of the mainstream game engines available and used in the industry is done in [14]. The authors have started about by first explaining the components of a traditional game engine like rendering, loading and animation along with physics to detect collision between meshes and objects. The game engine basically acts like a middleware, providing abstraction so that the same game can be played on different machines of the same type, and even machines of different types, with minimal changes to the code. Game engines can also be synergised with different specialisedsolutions like Havok which acts like a physics system and Binkfor a video system.

The paper also explains about different types of engines

- ✓ **MMO means Massively Multiplayer Online Game Engines:** This type is more complex compared to an engine for a single player game. These engines provide additional networking features compared to other engines. One popular example is the Hero Engine which acts as the middleware for games like The Elder Scrolls Online and Star Wars: The Old Republic.
- ✓ **FPS means Front Person Shooter Game Engines:** These are the most common type of engines in the industry and have been in use since 1998 and were popularised by games like Unreal Tournament and Halo: CombatEvolved which set the benchmark for games at that time. These engines have an emphasis on combat systems which include hitbox collision detection and bullet physics.
- ✓ **Visual Novel Engines:** These are used for very niche types of games called visual novels and are not very resource intensive. An Example for this type is NScripter.

The paper then talks about the evolution of game engines which started in early 1989 with the Ultima Underworld Engine which introduced algorithms for texture mapping for giving different designs to the floors, walls and ceilings of particular levels. The next iteration in game engine develop- ment came in 1993, when ID software made the Doom Engine, which ingeniously displayed 2D objects, like characters and maps in a layout such that it gave the illusion of it being three- dimensional. Hence rendering was very quick. In 1992, Voxel Engine was made and it had its own method of representing 3D models. It used bitmaps instead of vector graphics. Bitmaps usually have more detail and render faster than vector graphics. Build Engine used a concept similar to what ID Software did, creating 3D environments using 2D objects, and bifurcated the map into sectors, each with a different height. By switching the height of the player when he moves from one sector to another, the engine created the effect of level change at runtime. The first true 3D game engine was built by ID Software, which was named Quake Engine. It had a feature that removed certain areas of the map that players were not able to see from their position using a method called Z-buffering.

The paper concluded by analyzing the different game engines currently being used in the industry, based on parameters like platform and language support as well as API and physics engine compatibility. They also discussed the overall popu- larity of engines with respect to different consoles and lastly discussed the current state as well as trends of game engines.

III. PROPOSED MODELS

In the conventional system, the model used in implementing artificial intelligence involves a linear process that executes a decision upon reaching certain conditions. This does not indicate an artificially intelligent but tries to mimic intelligent behaviour. With the implementation of a decentralized model, the actions and reactions of the environment, world and other characters change certain properties of its agents which then help them react in a certain manner.

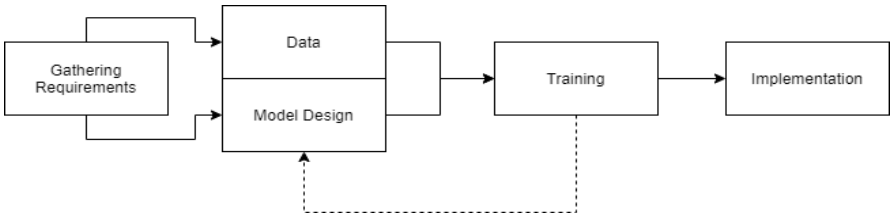

Fig. 2: Linear Model of AI Systems.

In the linear system, the intelligent actors are given set actions and reactions. This allows them for fast judgement when a particular stimulus is experienced. The development team firstascertains the requirements i.e., the actions and reactions the actors go through. These are then linked to the character mod- els and fine tuned until all character movement and reaction seem cohesive and follow natural laws. After this, the model moves toward training where it undergoes regressive testing to visualise working and fine tuning the response of the model under the influence of different factors as well as scenarios. This proposed system is based on the linear system where we design each component keeping in mind the decentralized nature of our AI systems. The model and its key components are:

1) Gathering Requirements

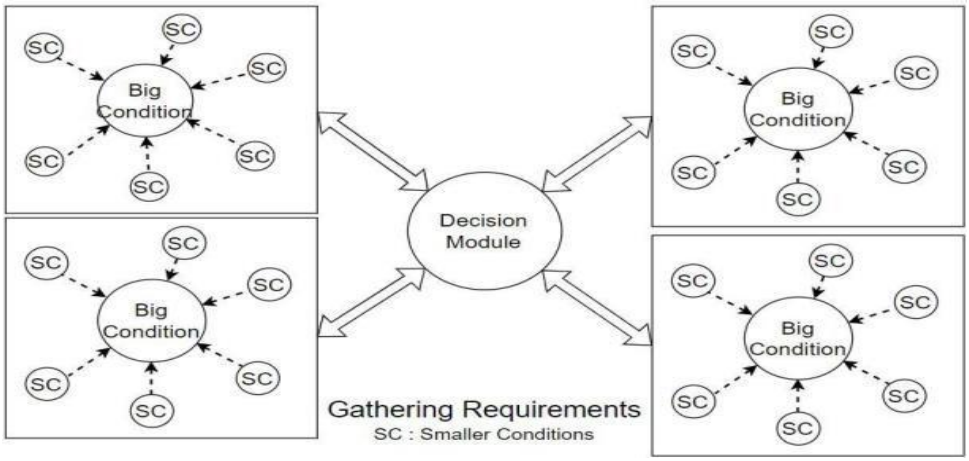

Fig. 3: Gathering Requirements Module of the Proposed System.

The decisions take place depending on some base conditions. Each of the smallest conditions are dependent on simple queries. Taking an example, if a character is hit, a true value is forwarded to the bigger nodes that helps relay the action. Small conditionals merge to form larger decisions that ultimately lead to the decision module. The simultaneous working of one or more modules leads to finalized decisions that helps in the gathering of requirements. The decentralized conditionals help simplify conditions and also makes the decision making more efficient.

2) Data

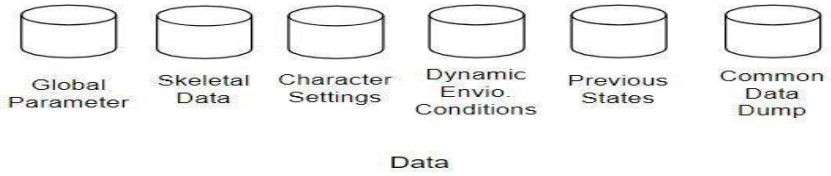

Fig. 4: Data Module of the Proposed System.

The data module consists of 6 key elements which helps provide a comprehensive set of data values which lead to each conditional having all the data required to execute and make decisions. These data values are either constantly updated or pull data directly from the simulated environment. The data sets consist of

- Global Parameter: Includes data such as global positioning of the requesting character which is the X, Y and Z coordinates along with any rotational values related to the location of the character. This helps provide a sense of location of the character and is very useful in the simulated environmentwhich is very useful for targeting systems, strategic systems etc.
- Skeletal Data: This entails the skeletal data i.e., the skeleton system of the character which involves the skin or mesh and the bones that are connected in a hierarchical pattern. This allows for the detection forany collisions from external forces and also helps the bones to work as a collective.
- Character Settings: Information pertaining to char-

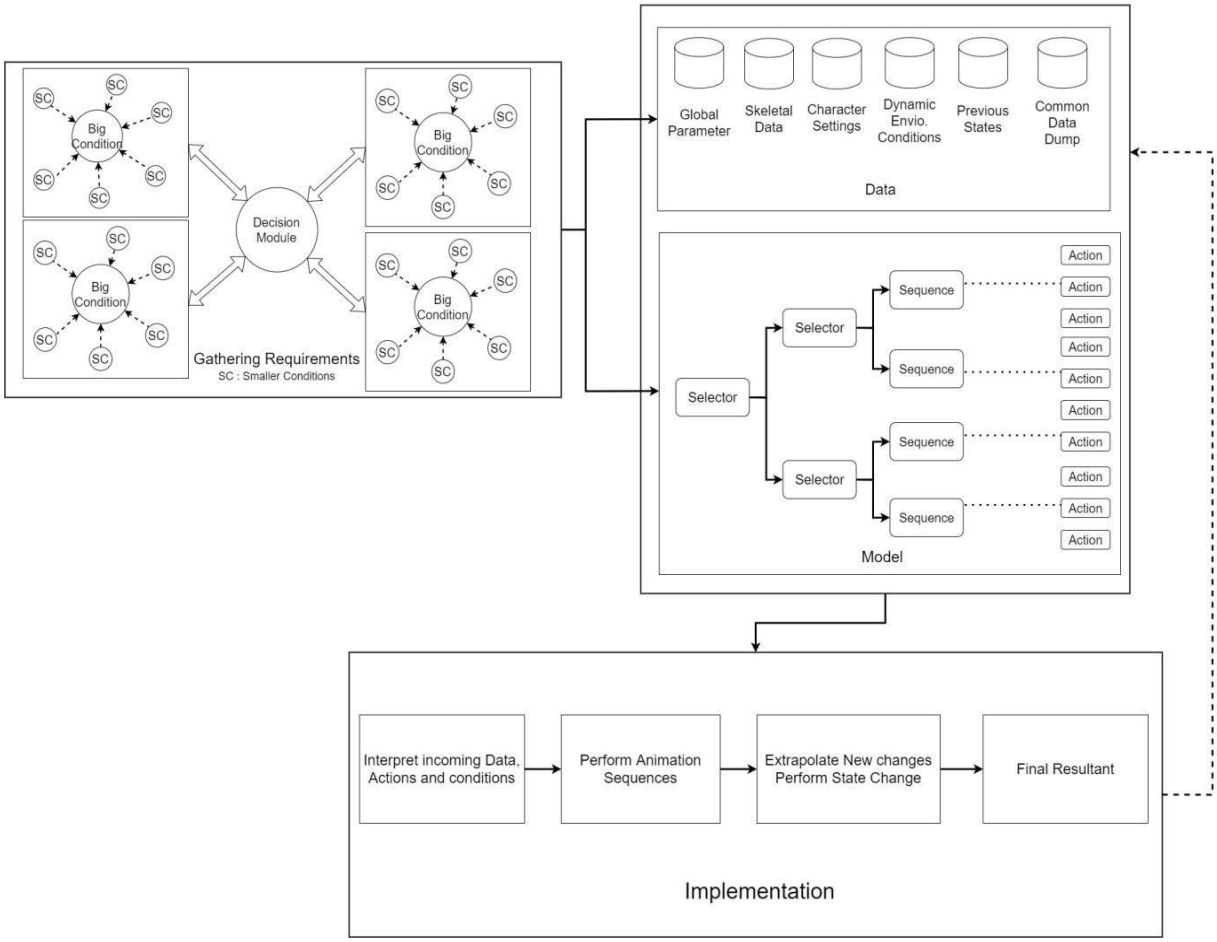

Fig. 1: The Proposed System.

acter experience levels, the amount of stamina they have, their levels in different skills and

their weapons. This helps to calculate a multitude of variables such as damage dealt, the distance they can sprint, if they are allowed to access particular skills, weapons and areas in the simulated environment.

- Dynamic Environment Conditions: Provides the environment details such as type of environment, level of adversaries in the area that can help limit a characters entry into the area, weather conditions which can also play a major role in the propagation of type of elements at any given time.
- Previous States: This includes all the previous interactions faced by a particular character or the series of characters.
- Common Data Dump: Data values of any conditionals in the past and any new data values that come in upon the execution of the newer conditionals.

3) Model - Behaviour Trees

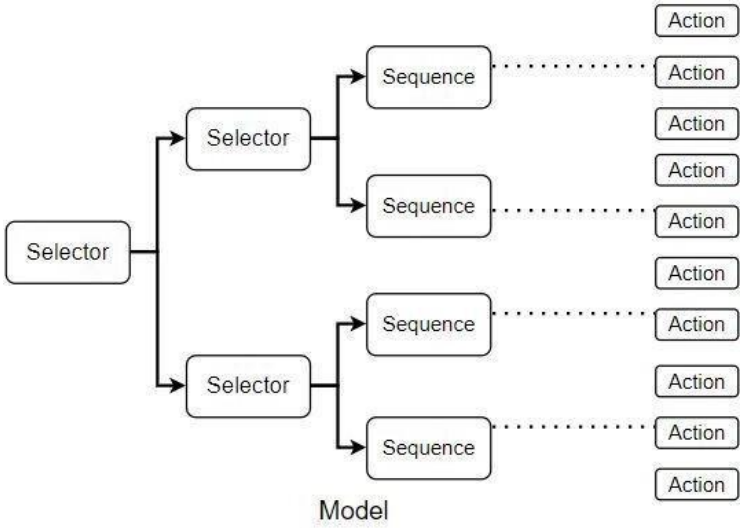

Fig. 5: Model Design Module of the Proposed System.

The behaviour tree will be applicable to all logic governing bodies in the environment. Not only will they act smart but this will also help provide a certain sense of unpredictability in our simulated environment. This entails all the important decisions and also leads to certain actions that are to be performed when particular sequences and selectors are validated. Each sequence is allotted a weight or a price that is to be extracted to complete the sequence. The idea behind the behaviour tree is to optimize the route taken for each sequence where the weights exercised are maximized or minimized as per the requirements of that particular scenario.

4) Implementation

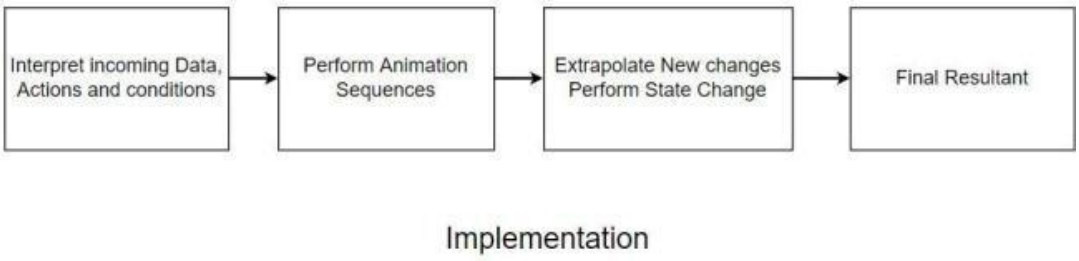

Fig. 6: Implementation Module of the Proposed System.

The Implementation module focuses on making sure the actors in the environment understand the different actions that need to be performed and deliver the desired output. This module consist of 4 key elements which are:

- ✓ Interpret incoming data, actions and conditions: Here the Game Data and Action sequence are presented to the actor. The actor will interpret the data and correspond each action to its corresponding parameter. For example the bone parameters on the shoulder are set to perform an action
- ✓ Perform Animation sequence: Upon setting each of the parameters, the actors perform the desired animation sequence. For example, as the shoulder bone is set the animation sequence is performed and the arm is rotated 90 degrees upwards.
- ✓ New changes and State change: Once the set of actions are performed the actor understands and extrapolates the changes in the parameters and per- forms the corresponding state change. For example after the animation sequence is performed the new positions for the arm are studied and the changes in the parameters and state change are presented.
- ✓ Final Result: After the State change, the ac- tors can perform these actions repeatedly as they compare the initial and current state and create a data template to reduce error rate to a minimum. For example, the actor can now perform the action moving an arm 90 degrees using the studied parameter changes.

The Implementation Module is looped with the data and model modules to create a seamless training algorithm for the actor to perform several actions, understand the differences and as the game progresses become smarter and tougher to beat.

IV. LIMITATIONS

1. The proposed system heavily relies on multiple condi-tionals that converge to make a

major decision. This makes the entire process very compute intensive and em- ploys multiple parallel decisions to reach a conclusion. Adding a huge multitude of conditionals in any scenariowill slow down the effective speed of operations being performed.

2. The model design used in any scenario is limited bythe constraints of the particular model. The choice of the model design plays a huge role to the point where different models can lead to a different level of efficiency. Our choice of using a behaviour tree dependson its support in our choice of rendering engine but thereare other models that can be used in these scenarios which would be more efficient.
3. Each path taken in the incentivized model has a partic- ular cost. This can directly or indirectly be influenced by the gathering requirements phase. The initial phase if delayed due to a conditional can adversely impact the whole system. One way to counter this would be to execute decisions in the model with the most recent information or weights.

V. CONCLUSION

The significance of Artificial Intelligence in video games cannot be underestimated by any means. The various different techniques for employing AI that are covered in this paper reaffirm the importance. We have seen that the ease of under- standing offered by Behavior Trees make it an extremely popu- lar method among the game developer community, highlighted by Unreal Engine's support for primarily using BTs to create game AI. Unreal Engine has become one of the most widely used game engines owing to its ability to create outstand- ing AAA titles and graphically intensive games. This paper further reviewed different pathfinding methods such as grids, quadtrees, nav-mesh as well as intelligent agent systems that can be used in modern games. Based on the literature review conducted, a system is proposed based on the linear system for AI agents. This proposed system is decentralized, where the NPCs are capable of taking different individual decisions. It involves gathering information from smaller conditionals for performing a certain action, followed by updating the relevant data in the environment. Finally the behavior tree determines the corresponding course of action and the actor's state is changed, after which the actor can compare the results in the future, resulting in a training algorithm for the actor. Further research and implementations for the proposed system would contribute to making it a generally feasible AI system which can be used in large scale games and RPGs.

VI. ACKNOWLEDGEMENT

We would like to thank prof Sanjay Deshmukh for useeful discussions and his constant guidance as well as support.

REFERENCES

1. V. Kushnir, B. Koman en at "CREATING AI FOR GAMES WITH UNREAL ENGINE 4" in C. 113–119 Electronics and information technologies, 2018.
2. Marcotte R., Hamilton H.J en at "Behavior Trees for Modelling Artificial Intelligence in Games: A Tutorial" in Comput Game J., 2017; 6: 171–184.
3. Sekhavat, Yoones en at "Behavior Trees for Computer Games" in the International Journal on Artificial Intelligence Tools. 2610.1142/S0218213017300010.
4. Johansson, Anja & Dell'Acqua, Pierangelo en at "Emotional behavior trees" in 2012 IEEE Conference on Computational Intelligence and Games, CIG, 2012; 355-362.
5. Martin Cerny, Tomas Plch, Matej Marko, Jakub Gemrot, Petr Ondracek, Cyril Brom et al "Using Behavior Objects to Manage Complexity in Virtual Worlds" in IEEE Transactions on Computational Intelligence and AI in Games /TCIAIG.2016.2528499.
6. Development of a Car Racing Simulator Game using Artificial Intel- ligence Techniques; Marvin T. Chan, Christine W. Chan, and Craig Gelowitz; Software Systems Engineering Program, Faculty of Engineer- ing and Applied Science, University of Regina, Regina, SK, Canada S4S 0A2.
7. Waltham, Michael & Moodley, Deshendran. An Analysis of Artificial Intelligence Techniques in Multiplayer Online Battle Arena Game Environments, 2016; 1-7. 10.1145/2987491.2987513.
8. Decentralized and Emergent NPC Coordination using Behavior Trees. Tiago Nunes Henriques, Instituto Superior Técnico, Taguspark Campus Av. Prof. Dr. An´ıbal Cavaco Silva, Oeiras, Portugal.
9. Cui, Xiao & Shi, Hao. "A*-based Pathfinding in Modern Computer Games", 2010; 11.
10. Abd Algfoor, Zeyad & Sunar, Mohd Shahrizal & Kolivand, Hoshang. "A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games", International Journal of Computer Games Technology, 2015; 1-11. 10.1155/2015/736138.
11. "Patrolling AI Systems in Video Games", S. Balapriya, N. Srinivasan from IJRTE, 8(4).
12. "Development of Pedestrian Artificial Intelligence Utilizing unreal En- gine 4 Graphic Engine", Fares Abu-Abed, Alexey Khabarov from IJRTE, 8(1).
13. Comparison of 3D games' efficiency with use of CRYENGINE and Unity game engines;

Hubert Zhukovsky, Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland.

14. HISTORY AND COMPARATIVE STUDY OF MODERN GAME EN- GINES; Partha Sarathi Paul, Surajit Goon, Abhishek Bhattacharya; Birla Institute of Technology, Kolkata Campus, Kol-107,WB, India, Institute of Engineering & Management, Salt Lake, Kol-91,WB, India.