CHAPTER 2

Literature Survey & Scope of Research

2.1 Introduction
2.2 Theories and Terminologies
2.3 Different Types of Games
2.4 Game Platforms
2.5 Game Development Process
2.6 Challenges in latest games
2.7 Scope of Research
2.8 Summery
Chapter -2

Literature Survey & Scope of Research

2.1 Introduction

This chapter gives information about existing techniques and technologies available in 3D game development. It also explains existing pathfinding algorithms and discusses their limitations. In middle part of the chapter, discussion about different types of game is made. In later part it discusses the scope of research in this area.

2.2 Theories and Terminologies

During the research work, following technologies are used to simplify the job of analysis, design, and coding activities of 3D game development.

2.2.1 Graph Theory

[10] In discrete mathematics, graphs are collections of discrete objects and their relations. Graph can be visually represented by symbolizing its objects as vertexes/ nodes and relations as edges/ lines. Figure 2.1 shows an example of a graph.

![Graph Example](image)

Figure 2.1: A graph example[10].
Mathematically, graph is defined as pair of two sets V and E,

Where, V: a non-empty set of vertex/ nodes = \{ V1, V2, ..... Vn\}

E: a set of edges/lines connecting pairs of nodes = \{ e1, e2, .... , en\}

In short, G = (V, E).

A directed graph is a graph in which its edges are given direction. In mathematical definition, directed edge is a set of ordered pairs of vertexes, or E = \{ (Va, Vb)\}; Va, Vb∈ V.

Figure 2.2 shows an example of a directed graph.

Figure 2.2: A directed graph example.

Terminology in graph theory:

- Adjacent
A pair of vertexes in undirected graph G is adjacent if there is an edge connecting both vertexes. All adjacent vertexes of vertex u are called neighbors of u.

- **Incident**
  An edge e in undirected graph G is incident with vertex u and v if e connects u and v.

- **Isolated Vertex**
  A vertex v is isolated if there is no edge in graph that incident with v. Isolated vertex can also be defined as a vertex which is not adjacent to any other vertexes in the graph.

- **Degree**
  Degree of a vertex v in undirected graph G is the number of vertex adjacent to v.

- **Path**
  A path of length n from vertex v0 to vn in graph G is a sequence of n edges e1, e2, ..., en so that e1 = (v0, v1), e2 = (v1, v2), en = (vn-1, vn) are edges in graph G. In other words, a path is a sequence of edges that begins at a vertex of graph and travels from vertex to vertex along edges of the graph [4].

- **Circuit or Cycle**
A circuit or cycle is a path that starts and ends at the same vertex.

- Connected
  An undirected graph G is called connected graph if for each pair of vertexes u and v in graph G there is a path connecting u and v.

### 2.2.2 Pathfinding in 3D games

Game characters usually need to move around their level. Sometimes this movement is set in stone by the developers, such as patrol route that a guard can follow blindly or a small fenced region in which a dog can randomly wander around. Fixed routes are simple to implement, but can easily be fooled if an object is pushed in the way. Free wandering characters can appear aimless and can easily get stuck [35].

More complex characters don’t know in advance where they will need to move. A unit in a real time strategy game may be ordered to any point on the map by the player at any time; a patrolling guard in a stealth game may need to move to its nearest alarm point to call for reinforcements; and a platform game may require opponents to chase the player across a chasm using available platforms[35].

For each for these characters the AI must be able to calculate a suitable route through the game level to get from where it is now to its goal. We had like the route to be sensible and as short or rapid as possible (it doesn’t look smart if our character walks from the kitchen to the lounge via the attic)[35].

This is pathfinding, sometimes called path planning, and it is everywhere in game AI.
2.2.2.1 Search algorithm used in 3D Pathfinding

Graphs are heavily used in video games pathfinding; hence, it is not surprising that graph searching become an essential topic in game programming. Graph searching is one of the most essential algorithms in the game programming. Depth-First Search (DFS) and Breadth-First Search (BFS), and Dijkstra are most common used algorithm for implementing Pathfinding and AI in 3D games. Following sections discusses these algorithms.
2.2.2.1 Depth First Search (DFS) Algorithm

In this section we are going to discuss one of the important graph travel and technique. One of them important graph travels and technique that is depth first search. To proceed with that we also required stack data operations such as push and pop. On right hand side in Figure 2.3 you can see an empty stack which is used for keeping track of all the visited nodes.

To start with from the starting node A, so A will be push it in stack. In output sequence also A will be added. Node A should marked as visited.

Stack: A
Chapter 2: Literature Survey & Scope of Research

Output: A

Now according to depth first search technique, the top of the stack symbol should be checked. From this A node all unvisited nodes should be checked on A. Here we have B and S adjacent to node A. Following the alphabetical order B should be pushed in Stack. B will be also added in output sequence and marked as visited in a graph.

Stack: B A

Output: A B

Now, B has not any unvisited adjacent so B will be pop up from the stack.

Hence, again A will be top on the stack and it has only one unvisited node which is S. so node S will be pushed on top of the stack. At the same time it will be also added to the output sequence. Also mark S will be marked as visited.

Stack: S A

Output: A B S

Now node S is on top of the stack. Node C and node G are unvisited adjacent to the node S. alphabetically node C should be push on the stack as well as added to the output sequence. Mark node C as visited.

Stack: C S A

Output: A B S

Node C has D, E and F as adjacent unvisited node. Among them D should be push in stack according to alphabetical order, add it to the sequence and mark node D as visited.
Stack: D C S A

Output: A B S C D

Node D being on top of the stack, has node C is the only adjacent but visited so there is no other go. So node D will be pop it up from the stack. It will be also marked as visited node.

Stack: C S A

Output: A B S C D

Now, Node C stand on top of the stack, node E and node F are the unvisited adjacent of node C. among them node E will be push in stack because of the alphabetical order, it will be added to the output sequence and it will be marked as visited.

Stack: E C S A

Output: A B S C D E

Now, node E has node H as unvisited adjacent. So it will be push in stack. It will be also added in output sequence and marked as visited.

Stack: H E C S A

Output: A B S C D E H

Node H has only one (node G) as unvisited adjacent. So it will be marked as visited, push in stack and added in output sequence.

Stack: G H E C S A

Output: A B S C D E H G

The last node which is left with G should be added on top of the stack, mark as visited and added into the output sequence.

Stack: F G H E C S A
Output: A B S C D E H

Now F has the two marked adjacent to node C and node G. so it should be pop from the stack. Similar case with node H, node E, node C, node S and node A. So H, E, C, S and A will be pop from the stack. Hence stack will become empty. This operation will signal the algorithm to stop. This is how the Depth First Search algorithm works.

Stack:

Output: A B S C D E H G F

In DFS, suppose the process starts from node A than, it will respectively visit A B S C D E H G F.

Figure 2.4 and 2.5 shows pseudo code of DFS in normal way and recursively respectively.

```plaintext
create empty stack S
u ← starting node
push u to S
mark u as visited
while S is not empty
    top ← pop(S)
    if top fulfills the search condition
        //some expression
        //stop function
    for each v neighbor of u
        if v is unvisited
            mark v as visited
            push v to S
```

Figure 2.4: DFS Pseudo code [11]
DFS(u):
    if u fulfills the search condition
        // some expression
        // stop function
    mark u as visited
    for each v neighbor of u
        if v is unvisited
            DFS(v)
    mark u as unvisited

Figure 2.5: Recursively defined DFS [11].
2.2.2.1.2 Breadth First Search (BFS) Algorithm

Breadth First Search (BFS) Algorithm works by visiting a node, enqueue all adjacent nodes to a queue and process them by dequeueing them. In other words, the process will visit every node in the same depth; let’s say depth n, before visit nodes in depth n+1\[10\]. It is also known as best first search algorithm.

\[52\] BFS node visiting process can be visualized as in Figure 2.6 BFS can be defined as follows:

![Breadth First Search (BFS) Algorithm](image)

Figure 2.6: Breadth First Search (BFS) Algorithm
Figure 2.7 shows pseudo code for implementation of BFS algorithm.

```
create empty queue Q
u ← starting node
enqueue u to Q
mark u as visited
while Q is not empty
  top ← dequeue(Q)
  if top fulfills the search condition
    //some expression
    //stop function
  for each v neighbor of u
    if v is unvisited
      mark v as visited
      enqueue v to Q
```

Figure 2.7: BFS pseudo code[10]

**Implementing DFS and BFS in games**

The following section discusses implementation of DFS and BFS in games for particular case.

**Case 1: Minesweeper Empty Tile Click**

In minesweeper [see figure 2.8], if player clicks an empty tile (tile that do not have number or bomb on it) all other empty tiles adjacent to it will also be opened [10].
Case 2: In turn based tactic / strategy games[see Figure 2.9], characters can move for a certain distance of tiles. If player selects a character, the game shows which tiles that are available to be set on. Tiles that are outside of character’s maximum distance, or have
obstacle or other character on will not be shown as available[10]. Notice that in Figure available tiles are shown as blue tiles. Far tiles and tiles that have characters or obstacles on are not colored blue.

The main objective of the coloring algorithm is to visit all tiles that are available in range, and color them. Tiles that are not in range or unavailable will not be visited [52].

![Image](image.png)

**Figure 2.9:** Turn based tactic / strategy games

At the first sight, both DFS and BFS seem can be implemented in this problem by limiting its range of checking. However, because of the range limitation, BFS is more suitable to be implemented than DFS and BFS visits all nodes in the same depth before visiting any nodes in the next depth. DFS, on the other hand, it may produce incorrect results because of the ranges limitation [10].
Figure 2.10 shows incorrect results in limiting DFS range.

![Figure 2.10: Incorrect Results in Limiting DFS Range](image)

In Figure 2.10, DFS visiting process is represents running clock wise. It visits top, right, bottom and finally left neighbor. DFS visiting directions are shown as arrows, the colors represent depth / range; red means depth 1, blue 2 and green 3. Since DFS look to the depth first, it may result a non-optimum solution of node depth range\[30\]; Tile\[4,3\] range from “start” should be 1, but it is counted as 3. Because DFS searching is limited to a number, in this case, there are some nodes that are never visited, but actually they should be visited. In such node is shown in Tile [5,3].

Depth first and breadth first search both have some advantages. Which is the best depends on properties of the problem you are
solving. For tree search at least, depth first search tends to require less memory, as you only need to record nodes on the `current' path. If there are lots of solutions, but all at a comparable `depth' in the tree, then you may hit on a solution having only explored a very small part of the tree. On the other hand, that may not be the best solution. Also, depth first search may get stuck exploring long (and potentially infinite) blind alleys, when there is in fact a solution path of only one or two steps. (We could prevent this by setting a depth limit to our search, but then it wouldn't be exhaustive.). So depth first is good when there are many possible solutions, and you only want one (and you don't care which one). It may be less appropriate when there is only one solution, or if, you want the shortest one[10, 11].

Breadth first search may use more memory, but will not get stuck in blind alleys, and will always find the shortest path first (or at least, the path that involves the least number of steps). It may be more appropriate when exploring very large search spaces where there is an expected solution which takes a relatively small number of steps, or when you are interested in all the solutions (perhaps up to some depth limit)[10,11].

Dijkstra allows assigning distances other than 1 for each step. For example, in routing the distances (or weights) could be assigned by speed, cost, preference, etc. The algorithm then gives you the shortest path from your source to every node in the traversed graph.

Meanwhile BFS basically just expands the search by one “step” (link, edge, whatever you want to call it in your application) on every iteration, which happens to have the effect of finding the smallest number of steps it takes to get to any given node from your source (“root”).
Breadth First Search (BFS) is a non-informed search for trees. It involves visiting the nodes closest to the root before spreading out.

BFS uses memory like a hog. Since it visits the nodes closest to the root before it visits their children, so store the children in a queue. First at each iteration, it takes out a child, check to see if it is the solution. If it is, exits quickly. If not, put children in the queue and repeat.

Hence, BFS put all the children of all the nodes it traverses, the queue can get quite big, especially if the branching factor is high.

BFS will perform well if the search space is small. It performs best if the goal state lies in upper left-hand side of the tree. But it will perform relatively poorly relative to the dept first search algorithm if the goal state lies in the bottom of the tree. BFS will always find the shortest path if the weight on the links are uniform.

The following section explains the Dijkstra algorithm in detail.

2.2.2.1.3 Dijkstra algorithm

Dijkstra's algorithm, conceived by computer scientist Edsger Dijkstra in 1956 and published in 1959. [1][2] It is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms [13, 14].

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping
the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra’s algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).

Dijkstra’s original algorithm does not use a min-priority queue and runs in time $O(|V|^2)$ (where $|V|$ is the number of vertices). [36] The implementation based on a min-priority queue implemented by a Fibonacci heap and running in $O(|E|+|V|\log|V|)$ (where $|E|$ is the number of edges) is due to. This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights.

**Dijkstra Algorithm in detail**

Let the node at which we are starting be called the initial node. Let the distance of node $Y$ be the distance from the initial node to $Y$. Dijkstra’s algorithm will assign some initial distance values and will try to improve them step by step[13, 14].

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes unvisited. Set the initial node as current. Create a set of the unvisited nodes called the *unvisited set* consisting of all the nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node $A$ is
marked with a distance of 6, and the edge connecting it with a neighbor \( B \) has length 2, then the distance to \( B \) (through \( A \)) will be \( 6 + 2 = 8 \). If \( B \) was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

4. When its done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.

5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop and the algorithm has finished.

6. Select the unvisited node that is marked with the smallest tentative distance, and set it as the new "current node" then go back to step 3.

**Dijkstra Description**

Suppose you would like to find the shortest path between two intersections on a city map, a starting point and a destination. The order is conceptually simple: to start, mark the distance to every intersection on the map with infinity. This is done not to imply there is an infinite distance, but to note that intersection has not yet been visited; some variants of this method simply leave the intersection unlabeled. Now, at each iteration, select a current intersection. For the first iteration, the current intersection will be the starting point and the distance to it (the intersection’s label) will be zero. For subsequent iterations (after the first), the current intersection will be the closest unvisited intersection to the starting point[13, 14].
From the current intersection, update the distance to every unvisited intersection that is directly connected to it. This is done by determining the sum of the distance between an unvisited intersection and the value of the current intersection, and relabeling the unvisited intersection with this value if it is less than its current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths. To facilitate shortest path identification, in pencil, mark the road with an arrow pointing to the relabeled intersection if you label/relabel it, and erase all others pointing to it. After you have updated the distances to each neighboring intersection, mark the current intersection as visited and select the unvisited intersection with lowest distance (from the starting point) – or the lowest label—as the current intersection. Nodes marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned [13, 14].

Continue this process of updating the neighboring intersections with the shortest distances, then marking the current intersection as visited and moving onto the closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as visited (as is the case with any visited intersection) you have determined the shortest path to it, from the starting point, and can trace your way back, following the arrows in reverse [13,14].

Of note is the fact that this algorithm makes no attempt to direct "exploration" towards the destination as one might expect. Rather, the sole consideration in determining the next "current" intersection is its distance from the starting point. This algorithm, therefore "expands outward" from the starting point, interactively considering every node that is closer in terms of shortest path distance until it reaches the destination. When understood in this way, it is clear how the
algorithm necessarily finds the shortest path, however, it may also reveal one of the algorithm's weaknesses: its relative slowness in some topologies.

The following section represents pseudo code of the Dijkstra algorithm.

**Assumption:**

In the following algorithm, the code u:= vertex in Q with min dist[u], searches for the vertex u in the vertex set Q that has the least dist[u] value. Length(u, v) returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes u and v. The variable alt on line 17 is the length of the path from the root node to the neighbor node v if it was to go through u. If this path is shorter than the current shortest path recorded for v than that current path is replaced with this alt path. The previous array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source [13, 14, 17].
Pseudo code of Dijkstra [13,14]

1  function Dijkstra(Graph, source):
2      dist[source] := 0  // Distance from source to source
3      for each vertex \( v \) in Graph:  // Initializations
4          if \( v \) \( \neq \) source
5              dist[v] := infinity  // Unknown distance function from source to \( v \)
6              previous[v] := undefined  // Previous node in optimal path from source
7          end if
8      add \( v \) to \( Q \)  // All nodes initially in \( Q \)
9  end for
10
11  while \( Q \) is not empty:  // The main loop
12      \( u \) := vertex in \( Q \) with min dist[\( u \)]  // Source node in first case
13      remove \( u \) from \( Q \)
14
15      for each neighbor \( v \) of \( u \):  // where \( v \) has not yet been removed from \( Q \).
16          alt := dist[\( u \)] + length(\( u \), \( v \))
17          if alt<dist[\( v \)]:  // A shorter path to \( v \) has been found
18              dist[\( v \)] := alt
19              previous[\( v \)] := \( u \)
20          end if
21  end for
22  end while
23  return dist[], previous[]
24 end function
If we are only interested in a shortest path between vertices source and target, we can terminate the search if \( u = target \). It can read the shortest path from source to target by reverse iteration [13,14]:

1. \( S := \text{empty sequence} \)
2. \( u := target \)
3. \( \textbf{while} \) previous[\( u \)] is defined: // Construct the shortest path with a stack \( S \)
   4. insert \( u \) at the beginning of \( S \) // Push the vertex into the stack
   5. \( u := \) previous[\( u \)] // Traverse from target to source
6. \( \textbf{end while} \)

Here, sequence \( S \) is the list of vertices constituting one of the shortest paths from source to target, or the empty sequence if no path exists.

A more general problem would be to find all the shortest paths between source and target (there might be several different ones of the same length). Then instead of storing only a single node in each entry of previous [52] we would store all nodes satisfying the relaxation condition. For example, if both \( r \) and source connect to target and both of them lie on different shortest paths through target (because the edge cost is the same in both cases), then we would add both \( r \) and source to previous [52]. When the algorithm completes, previous [52] data structure will actually describe a graph that is a subset of the original graph with some edges removed. Its key property will be that if the algorithm was run with some starting node, then every path from that node to any other node in the new graph will be the shortest path between those nodes in the original graph, and all paths of that length from the original graph will be present in the new graph. Then
to actually find all these shortest paths between two given nodes we would use a path finding algorithm on the new graph, such as depth-first search [14].

**Disadvantages of Dijkstra’s Algorithm**

The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources. Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path [13, 14, 15].

**2.3 Different types of games**

[18,19,20,21] Video game types (genres) are used to categorize video games based on their gameplay interaction rather than visual or narrative differences.[1] A video game genre is defined by a set of gameplay challenges. They are classified independent of their setting or game-world content, unlike other works of fiction such as films or books. For example, an action game is still an action game, regardless of whether it takes place in a fantasy world or outer space.[2] Within game studies there is a lack of consensus in reaching accepted formal definitions for game genres, some being more observed than others. Like any typical taxonomy, a video game genre requires certain constants. Most video games feature obstacles to overcome, so video game genres can be defined where obstacles are completed in substantially similar ways [13, 14, 15].

Following is a listing of commonly used video game genres with brief descriptions and examples of each. This list is by no means complete or comprehensive. Chris Crawford notes that “the state of computer game design is changing quickly. We would therefore expect the taxonomy presented here to become obsolete or inadequate in a short
As with nearly all varieties of genre classification, the matter of any individual video game's specific genre is open to personal interpretation. Moreover, it is important to be able to "think of each individual game as belonging to several genres at once."[1]

**Types of Video game genre**

### 2.3.1 Action

An action game requires players to use quick reflexes, accuracy, and timing to overcome obstacles. It is perhaps the most basic of gaming genres, and certainly one of the broadest. Action games tend to have gameplay with emphasis on combat. There are many subgenres of action games, such as fighting games and first-person shooters[18,19].

#### 2.3.1.1 Ball and paddle

The predecessor of all console game genres, a ball-and-paddle game was the first game implemented on a home console (Pong). Later renditions have included Breakout, which was a driving influence behind the Apple II computer. A version of Breakout called Block Buster was also packaged with the first handheld console with swappable cartridges, the Microvision [18, 19].

#### 2.3.1.2 Beat 'em up and hack and slash

Beat 'em up and hack and slash games have an emphasis on one-on-many close quarters combat, beating large numbers of computer-controlled enemies.[4][5] Gameplay involves the player fighting through a series of increasingly difficult levels. The sole distinction between these two genres are that beat 'em ups feature hand-to-hand combat, and hack and slash games feature melee weaponry, particularly bladed weapons. Both genres feature little to no use of
firearms or projectile combat. This genre became popular in 1987 with the release of Double Dragon, leading to a large number of similar games. The fighting style is usually simpler than for versus fighting games. In recent times, the genre has largely merged with that of action-adventure, with side-scrolling levels giving way to more open three-dimensional areas, and melee combat co-existing with shooting and puzzle elements[18,19].

2.3.1.3 Traditional Fighting game

Fighting games emphasize one-on-one combat between two characters, one of which may be computer controlled.[6][7] These games are usually played by linking together long chains of button presses on the controller to use physical attacks to fight. Many of the movements employed by the characters are usually dramatic and occasionally physically impossible. Combat is always one-on-one.[6,18,19] This genre first appeared in 1976 with the release of Sega's Heavyweight Boxing and later became a phenomenon, particularly in the arcades, with the release of Street Fighter II. Later, in 1992, the Mortal Kombat series debuted and brought with it new features for future fighting games, features such as a dedicated block button; the performance of a "finishing move" on a defeated opponent; and in-game secrets such as hidden and otherwise unplayable characters[[18,19].

2.3.1.4 Mascot Fighting game

Mascot Fighting games, often called 'party brawlers', are fighting games usually developed to showcase a particular brand's line up of intellectual properties. Unlike traditional fighting games, mascot fighting games allow for up to four characters on the screen at a given time, up to three of which may be computer controlled. These games
are played similarly to traditional fighting games, with the exception that stages in mascot fighting games are usually bigger than those in traditional fighting games and may or may not have platforms, allowing for vertical combat. Another feature exclusive to mascot fighting games is the ability for items to spawn on stage, giving a lesser skilled players assistance against tough opponents. A stage hazard, which is an attack at an area of the stage that causes damage to players that are unfortunate enough to get caught in it, may appear that players will have to look out for, unless the feature is turned off for the match. Notable releases in this genre are Super Smash Bros, Cartoon Network: Punch Time Explosion, and PlayStation All-Stars Battle Royale [18,19].

2.3.1.5 MOBA

Multiplayer online battle arena (MOBA), also known as action real-time strategy (ARTS) or Hero Brawler, is a sub-genre of the real-time strategy (RTS) genre of video games, in which often two teams of players compete with each other in discrete games, with each player controlling a single character through an RTS-style interface. It differs from traditional RTS games in that there is no unit construction and players control just one character. In this sense, it is a fusion of action games and real-time strategy games. The genre emphasizes cooperative team-play; players select and control one "hero", a powerful unit with various abilities and advantages to form a team’s overall strategy. The objective is to destroy the opponents' main structure with the assistance of periodically spawned computer-controlled units that march towards the enemy's main structure via paths referred to as "lanes". Notable examples include Defense of the Ancients, League of Legends, and Smite [18,19].
2.3.1.6 Shooter

2.3.1.6.1 First-person shooter

First-person shooter video games, commonly known as FPSs, emphasize shooting and combat from the perspective of the character controlled by the player. This perspective is meant to give the player the feeling of "being there", and allows the player to focus on aiming. Most FPSs are very fast-paced and require quick reflexes on high difficulty levels. The fast-paced and 3D elements required to create an effective looking FPS made the genre technologically unattainable for most consumer hardware systems until the early 1990s. Wolfenstein 3D was the first widely known FPS, and Doom was the first major breakthrough in graphics; it used a number of clever techniques to make the game run fast enough to play on consumer-grade machines. Since the release of Doom, most FPS games now have a multi-player feature to allow competition between multiple players. Games such as Team Fortress, Halo, Killzone, Metroid Prime, Unreal Tournament, Quake, Half-Life, Call of Duty, TimeSplitters and Battlefield are in the ever-expanding first-person shooter genre[18,19].

2.3.1.6.2 Massively multiplayer online first person shooter

Massively multiplayer online first person shooter games (MMOFPS) are a genre of massively multiplayer online games that combines first-person shooter gameplay with a virtual world in which a large number of players can interact over the Internet. Whereas standard FPS games limit the number of players able to compete in a multiplayer match (generally the maximum is 64, due to server capacity), hundreds of players can battle each other on the same server in an MMOFPS. An example of a MMOFPS is PlanetSide 2[19].
2.3.1.6.3 Light gun shooter

Light gun shooters are a genre of shooter genre designed for use with a pointing device for computers and a control device for arcade and home consoles.[11][12][13] The first light guns appeared in the 1930s, following the development of light-sensing vacuum tubes.

2.3.1.6.4 Third-person shooter

Third-person shooter video games, known as TPSs or 3PSs, emphasize shooting and combat from a camera perspective in which the player character is seen at a distance. This perspective gives the player a wider view of their surroundings as opposed to the limited viewpoint of first-person shooters. [18,19,22].

2.3.2 Action-adventure

Action-adventure games combine elements of their two component genres, typically featuring long-term obstacles that must be overcome using a tool or item as leverage (which is collected earlier), as well as many smaller obstacles almost constantly in the way, that require elements of action games to overcome. Action-adventure games tend to focus on exploration and usually involve item gathering, simple puzzle solving, and combat. "Action-adventure" has become a label which is sometimes attached to games which do not fit neatly into another well known genre[22, 52].

The first action-adventure game was the Atari 2600 game Adventure (1979).

2.3.2.1 Stealth game

Stealth games are a somewhat recent sub-genre, sometimes referred to as "sneakers" or "creepers" to contrast with the action-oriented
"shooter" sub-genre. These games tend to emphasize subterfuge and precision strikes over the more overt mayhem of shooters[22, 52].

2.3.2.2 Survival horror

Survival horror games focus on fear and attempt to scare the player via traditional horror fiction elements such as atmospherics, death, the undead, blood and gore. One crucial gameplay element in many of these games is the low quantity of ammunition, or number of breakable melee weapons. A notable example is Silent Hill [22,52].

2.3.3 Adventure

Unlike adventure films, adventure games are not defined by story or content. Rather, adventure describes a manner of gameplay without reflex challenges or action. They normally require the player to solve various puzzles by interacting with people or the environment, most often in a non-confrontational way. It is considered a "purist" genre and tends to exclude anything which includes action elements beyond a mini game[18,20,22,52].

2.3.3.1 Real-time 3D adventures

Around this time, real-time 3D adventure games appeared. These included Nightfall in 1998, realMyst in 2000, and Uru: Ages BeyondMyst in 2003. They augmented traditional adventure gameplay with some of the attributes more commonly associated with action games. For example, freedom of motion and physics based behavior[22].

2.3.3.2 Graphic adventures

Graphic adventure games emerged as graphics became more common. Adventure games began to supplement and later on replace textual
descriptions with visuals (for example, a picture of the current location) [22].

2.3.4 Role-playing

Role-playing video games draw their gameplay from traditional role-playing games like Dungeons & Dragons. Most of these games cast the player in the role of one or more "adventurers" who specialize in specific skill sets (such as melee combat or casting magic spells) while progressing through a predetermined storyline. [18, 22]

2.3.4.1 Role-playing Choices

Some RPGs give the player several choices in how their story goes and such. Typically the player can effect if a character dies or whether they kill an enemy or simply knock them out non-lethally, one example being Dishonored. These are very popular to gamers because they have to deal with the consequences of their own choices, rather than the games developers, whenever they fail to save someone or don't get the ending they desired. This gives a much more interactive experience between gamers and gameplay thus also explaining their popularity. One notable example is the Mass Effect series. [22]

2.3.4.2 Use of fantasy in RPGs

Due to RPG origins with Dungeons and Dragons and other pen and paper role-playing games, the most popular setting for RPGs by far is a fantasy world, usually with heavy medieval European influences with Diablo series (by Blizzard), Final Fantasy series, Elder Scrolls series and Baldur's Gate series (all different kinds of RPGs) all sharing a basic fantasy setting. However exceptions do exist, with some more notable ones being the east Asian Jade Empire setting, and the science fiction settings of Knights of the Old Republic and Mass Effect by Bioware. The Fallout series is set in a post-apocalyptic retro-
futuristic America in which nuclear war destroyed a world in which culture had never advanced beyond that of the 1950s[22].

2.3.4.3 MMORPGs

Massively multiplayer online role-playing games, or MMORPGs, feature the usual RPG objectives of completing quests and strengthening one's player character, but involve up to hundreds of players interacting with each other on the same persistent world in real-time. Fantasy MMORPGs like The Lord of the Rings Online and Shadows of Angmar are remain the most popular type of MMOG. [22]

2.3.5 Simulation

2.3.5.1 Construction and management simulation

Construction and management simulations (or CMSs) are a type of simulation game which task players to build, expand or manage fictional communities or projects with limited resources.

In city-building games the player acts as overall planner or leader to meet the needs and wants of game characters by initiating structures for food, shelter, health, spiritual care, economic growth, etc. Success is achieved when the city budget makes a growing profit and citizens experience an upgraded lifestyle in housing, health, and goods. While military development is often included, the emphasis is on economic strength. Perhaps the most known game of this type is SimCity, which is still popular and has had great influence on later city-building games. SimCity, however, also belongs to the God Games genre since it gives the player god-like abilities in manipulating the world. Caesar was a long-running series in this genre, with the original game spawning three sequels[22].
Chapter 2: Literature Survey & Scope of Research

2.3.6 Strategy

Strategy video games focus on gameplay requiring careful and skillful thinking and planning in order to achieve victory and the action scales from world domination to squad-based tactics.

2.2.6.1 Artillery game

Artillery is the generic name for early two or three-player (usually turn-based) computer games involving tanks fighting each other in combat or similar derivative games. Artillery games were among the earliest computer games developed and can be considered an extension of the original use of computers, which were once used for military-based calculations such as plotting the trajectories of rockets. Artillery games are a type of strategy game, though they have also been described as a "shooting game."

Scorched 3D is an artillery game[18,22].

2.2.6.2 Real-time strategy (RTS)

The moniker "real-time strategy" (RTS), usually applied only to certain computer strategy games, indicates that the action in the game is continuous, and players will have to make their decisions and actions within the backdrop of a constantly changing game state. Some notable games include the Warcraft series, Age of Empires series, Dawn of War, Command and Conquer and Dune II (essentially the first RTS game)[22].

2.3.7 Sports

Sports are games that play competitively one team, containing or controlled by you, and another team that opposes you. This opposing team(s) can be controlled by other real life people or artificial intelligence [22].
2.3.7.1 Racing

One competes against time or opponent using some means of transportation. Most popular sub-genre is racing simulators[22].

2.3.7.2 Sports game

Sports games emulate the playing of traditional physical sports. Some emphasize actually playing the sport, while others emphasize the strategy behind the sport (such as Championship Manager). Others satirize the sport for comic effect (such as Arch Rivals). One of the best-selling series in this genre is the FIFA (video game series) series. This genre emerged early in the history of video games (e.g., Pong) and remains popular today [22].

2.3.8 Other notable genres

2.3.8.1 Music game

Music games most commonly challenge the player to follow sequences of movement or develop specific rhythms. Recently, music games such as Guitar Hero, Rock Band and Sing Star have achieved huge popularity among casual gamers [22].

2.3.8.2 Educational game

Educational games, as the name implies, attempt to teach the user using the game as a vehicle. Most of these types of games target young users from the ages of about three years to mid-teens; past the mid-teens, subjects become so complex (e.g. Calculus) that teaching via a game is impractical. Numerous sub-genres exist, in fields such as math or typing [22].
Chapter 2: Literature Survey & Scope of Research

2.4 Game Platforms

This section explains game platforms in details.

2.4.1 Mobile game

A mobile game is a video game played on a feature phone, smartphone, PDA, tablet computer, portable media player or calculator. This does include games played on dedicated handheld video game systems such as Nintendo 3DS or PlayStation Vita.[18,22]

The first game on a mobile phone was a Tetris game on the Hagenuk MT-2000 device from 1994 [2][3].

2.4.2 Arcade game

An arcade game (or coin-op) is a coin-operated entertainment machine, usually installed in public businesses, such as restaurants, bars, and particularly amusement arcades. Most arcade games are video games, pinball machines, electro-mechanical games, redemption games, and merchandisers (such as claw cranes)[22].

The golden age of arcade video games lasted from the late 1970s to the mid-1990s. While arcade games were still relatively popular during the late 1990s, the entertainment medium saw a continuous decline in popularity in the Western hemisphere when home-based video game consoles made the transition from 2D graphics to 3D graphics. Despite this, arcades remain popular in many parts of Asia as late as the early 2010s [22].

The term "arcade game" is also, in recent times, used to refer to a video game that was designed to play similarly to an arcade game with frantic, addictive gameplay [52].
2.4.3 Console game

A console game is a form of interactive multimedia used for entertainment. The game consists of manipulable images (and usually sounds) generated by a video game console and displayed on a television or similar audio-video system. The game itself is usually controlled and manipulated using a handheld device connected to the console, called a controller. The controller generally contains a number of buttons and directional controls, (such as analog joysticks), each of which has been assigned a purpose for interacting with and controlling the images on the screen. The display, speakers, console, and controls of a console can also be incorporated into one small object known as a handheld game [22].

Modern game multimedia usually comes in the form of an disc, which can be inserted into the game console. Recent advances have allowed games and game demos to be downloaded directly to the console via the Internet. Simpler consoles, however, may only have a fixed selection of built-in games. Cartridges were previously common storage devices for video game data, but due to technological advances, most video games are now stored on CDs, or higher capacity DVDs, or the Blu-ray Disc used by the PlayStation 3. The only games stored on cartridges now are the ones for the Nintendo DS, Nintendo 3DS, Playstation Vita, Neo Geo X and they are called "cards"[22].

2.4.4 PC game

PC games, also known as computer games, are video games played on a general-purpose personal computer rather than a dedicated video game console or arcade machine. Their defining characteristics
include a lack of any centralized controlling authority and greater capacity in input, processing, and output [18, 22]. Eg. Bedroom coder.

2.4.5 Handheld video game

A handheld video game is a video game designed for a handheld device. In the past, this primarily meant handheld game consoles such as Nintendo's Game Boy line. In more recent history, mobile games have become popular in calculators, personal digital assistants (PDA), mobile phones, digital audio players (e.g., MP3), and other similar portable gadgets [18,52].

2.5 Game Development process

Game development is a software development process, as a video game is software with art, audio, and gameplay. Formal software development methods are often overlooked. [22]Games with poor development methodology are likely to run over budget and time estimates, as well as contain a large number of bugs. Planning is important for individual and group projects alike[22,52].

Overall game development is not suited for typical software life cycle methods, such as the waterfall model [7].

Methods employed for game development are agile development and Personal Software Process(PSP) [7]. Some important game development process are as bellow.

Pre-production

Pre-production[22] or design phase[18] is a planning phase of the project focused on idea and concept development and production of initial design documents [18,22].

Game design document
Before a full-scale production can begin, the development team produces the game design document incorporating all or most of the material from the initial pitch [18,22].

**Prototype**

Placeholder graphics are characteristic of early game prototypes.

Writing prototypes of gameplay ideas and features is an important activity that allows programmers and game designers to experiment with different algorithms and usability scenarios for a game. A great deal of prototyping may take place during pre-production before the design document is complete and may, in fact, help determine what features the design specifies. Prototyping may also take place during active development to test new ideas as the game emerges [18,22].

**Production**

Production is the main stage of development, when assets and source code for the game are produced[18].

Mainstream production is usually defined as the period of time when the project is fully staffed[18]. Programmers write new source code, artists develop game assets, such as, sprites or 3D models. Sound engineers develop sound effects and composers develop music for the game. Level designers create levels, and writers write dialogue for cutscenes and NPCs. Game designers continue to develop the game’s design throughout production [52].

**Design**

Game design is an essential and collaborative[19] process of designing the content and rules of a game, requiring artistic and technical competence as well as writing skills. During development, the game
Chapter 2: Literature Survey & Scope of Research

designer implements and modifies the game design to reflect the current vision of the game [22].

**Programming**

The programming of the game is handled by one or more game programmers. They develop prototypes to test ideas, many of which may never make it into the final game. The programmers incorporate new features demanded by the game design and fix any bugs introduced during the development process. Even if an off-the-shelf game engine is used, a great deal of programming is required to customize almost every game[22].

**Level creation**

From a time standpoint, the game’s first level takes the longest to develop. As level designers and artists use the tools for level building, they request features and changes to the in-house tools that allow for quicker and higher quality development. Newly introduced features may cause old levels to become obsolete, so the levels developed early on may be repeatedly developed and discarded[22,52].

**Art production**

It’s usually done by art team. Which includes Sketch Artist and others.

**Audio production**

Game audio may be separated into three categories—sound effects, music, and voice-over [22].

**Testing**

At the end of the project, quality assurance plays a significant role. Testers start work once anything is playable. This may be one level or subset of the game software that can be used to any reasonable
extent. Early on, testing a game occupies a relatively small amount of time. Testers may work on several games at once. As development draws to a close, a single game usually employs many testers full-time (and often with overtime) [22].

**Milestones**

Commercial game development projects may be required to meet milestones set by publisher. Milestones mark major events during game development and are used to track game's progress. Such milestones may be, for example, first playable alpha or beta game versions. Project milestones depend on the developer schedules [22].

**Code freeze**

Code freeze is the stage when new code is no longer added to the game and only bugs are being corrected. Code freeze occurs three to four months before code release [29,30].

**Beta**

Beta is feature and asset complete version of the game, when only bugs are being fixed

**Code release**

Code release is the stage when all bugs are fixed and game is ready to be shipped or submitted for console manufacturer review. This version is tested against QA test plan. First code release candidate is usually ready three to four weeks before code release [22,23,24].

**Gold master**

Gold master is the final game's build that is used as a master for production of the game.

**Crunch time**
Chapter 2: Literature Survey & Scope of Research

Overtime is expected in the games industry [25,26] Particularly, crunch time unpaid overtime requested by many companies to meet project deadlines and milestones that negatively affects game developers[22].

**Post-production**

After the game goes gold and ships, some developers will give team members comp time (perhaps up to a week or two) to compensate for the overtime put in to complete the game [24].

**Maintenance**

In recent times popularity of online console games has grown, and online capable video game consoles and online services such as Xbox Live for the Xbox have developed. Developers can maintain their software through downloadable patches. These changes would not have been possible in the past without the widespread availability of the Internet [22, 23].

**2.6 Challenges in latest games**

As discussed earlier many pathfinding algorithm are available but each of them has limitations.

**Challenges**

**2.6.1 Disadvantages of Dijkstra’s Algorithm**

- The major disadvantage of this algorithm is the fact that it does a blind search by consuming a lot of time waste of necessary resources [46, 47].
- Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path [46, 47].
2.6.2 Disadvantages of BFS Algorithm

- Breadth First Search (BFS) is a non-informed search for trees. It involves visiting the nodes closest to the root before spreading out [49, 50].
- BFS uses memory like a hog. Since we want to visit the nodes closest to the root before we visit their children, we must store the children in a queue. At each iteration, we take out a child, check to see if she is the solution. If she is, exit quickly. If not, put her children in the queue and repeat [49].
- BFS must put all the children of all the nodes it traverses, the queue can get quite big, especially if the branching factor is high [48,50].
- BFS will perform well if the search space is small. It performs best if the goal state lies in upper left-hand side of the tree. But it will perform relatively poorly relative to the dept first search algorithm if the goal state lies in the bottom of the tree. BFS will always find the shortest path if the weight on the links are uniform [49, 50].

2.7 Scope of research

Since game AI is centered on appearance of intelligence and good gameplay, its approach is very different from that of traditional AI. A common goal today is to make the game AI more human, or at least appear so.

One of the greatest challenges in the design of realistic Artificial Intelligence (AI) in computer games is agent movement and Pathfinding strategies. Pathfinding strategies are usually employed as the core of any AI movement system. Pathfinding strategies have the
responsibility of finding a path from any coordinate in the game world to another. Systems such as this take in a starting point and a destination; they then find a series of points that together comprise a path to the destination. Pathfinding in 3D games usually employs some sort of precomputed data structure to guide the movement. At its simplest, this could be just a list of locations within the game that the agent is allowed move to. Pathfinding inevitably leads to a drain on CPU resources especially if the algorithm wastes valuable time searching for a path that turns out not to exist. The Techniques for implementation of steer behavior (NPCs, Combat system) in the game is also core of AI system in the 3D game development.

A non-player character (NPC), sometimes known as a non-person character or non-playable character in a game is any character that is not controlled by a player. In electronic games, this usually means a character controlled by the computer through artificial intelligence. In traditional tabletop role-playing games the term applies to characters controlled by the game master or referee, rather than another player. E.g. Flocks, animals, crowds etc. NPC behavior in computer games is usually scripted and automatic, triggered by certain actions or dialogue with the player characters. Video game combat AI: The many contemporary video games fall under the category of action, first person shooter, or adventure. In most of these types of games there is some level of combat that takes place. The AI's ability to be efficient in combat is important in these genres. A common goal today is to make the AI more human, or at least appear so.

2.8 Summery

In the first part of this chapter introduction of graph theory and search algorithm like DFS, BFS and Dijkstra for 3D pathfinding has been discussed. In the middle part, types of games and game platforms has
been discussed. Later part of this chapter discusses steps for game development and limitation of existing pathfinding algorithm. In the last scope of research has been identified.