ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

Simulation of Warehouse Automation using Genetic Algorithm

Manilal D. L.[1], Sachin Jose[2], Vidyadheesa D N[3]

Head of Department, Department of Computer Science, Model Engineering College, Kochi, Kerala, India[1]

U.G Student, Department of Computer Science, Model Engineering College, Kochi, Kerala, India[2, 3]

ABSTRACT: The world is moving towards increased automation and the advent of sophisticated machine learning algorithms coupled with the exponential rise in robotic capabilities has increased the rate at which automation is being adopted. At the same time games provide a rich and open environment to model decision making. In this paper we explore the automation of a warehouse using genetic algorithms which will be trained using the rich unstructured environment found in games. The concepts described in this paper to create and simulate a virtual warehouse in Unity with multiple parallel bots .
The benefits of genetic computation over traditional reinforcement learning techniques for utilising traditional first person games like tetris will also be discussed .

KEYWORDS:Tetris, Warehouse, Automation, Genetic Algorithm, A* Algorithm, Path Planning.

I.INTRODUCTION

Medium to mega warehouses are becoming a common site due to local e-commerce giants like Amazon,Alibaba and Flipkart. To most business warehouses are usually the cause of huge overheads and are usually run sub optimally making it prime for automation. While industrial automation is usually driven with the goal of increase in manufacturing productivity, much of the current motivation behind these automation techniques is driven by the need to enhance efficiency, increase the safety of the operational environment and quickly respond to the changing consumer demand for the new products. Machine learning has found various applications across industries. In this model of simulation, an efficient stacking algorithm is designed using genetic algorithms. The open game space provided by Tetris is utilised to design the stacking algorithm. Graph algorithms are utilised while determining locations to ensure conflicting items aren't placed close to each other.

There are mainly 2 components to this project, they are the

1) Stacking Algorithm

2) Robotics Algorithm

II. RELATED WORKS

This Simulation consists of using the rich game space to model decisions. Here we utilise tetris for the purpose of creating a sophisticated stacking algorithm. Automation is widely gaining popularity among different parts of the industry. In recent years, computer games have been developed rapidly and provide an ideal platform for Artificial Intelligence (AI) research. The purpose here was to research about various methods or strategies used by different types of games/robots AI to create better AI opponents/solve games. There were various types of reinforcement techniques used in developing this AI. One that stood out most of the time was Deep Q-learning(DQL). This technique was used in creating ViZDoom (Visual Doom) AI and also by mobile robots.There are games such as Chess, GO, Hex etc. which are turn-based games. Since these are not real-time games, algorithms such as -greedy, actor-critic(or A3C),Monte-

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

Carlo tree search(MCTS), particle swarm optimization and genetic algorithms are useful to solve problems of order with exponential time complexity..

III.WAREHOUSE AUTOMATION CONTROL SYSTEM OVERVIEW

The simulation consists of 4 warehouse shelves. There is a drop off station for the cargo, from where the cargo will be placed in nodes for pickup by bots. When the cargo reaches the drop off station it is assigned to one of the 4 nodes, if the nodes are busy it will be added to a queue. When the Drop-off station node contains cargo it flags the bot station for pickup. The botstation assigns a free bot to the Node. If no bot is free then it will push the request into the bot-station queue. The bot picksup the cargo and calculates the position of it is supposed to drop the cargo at.

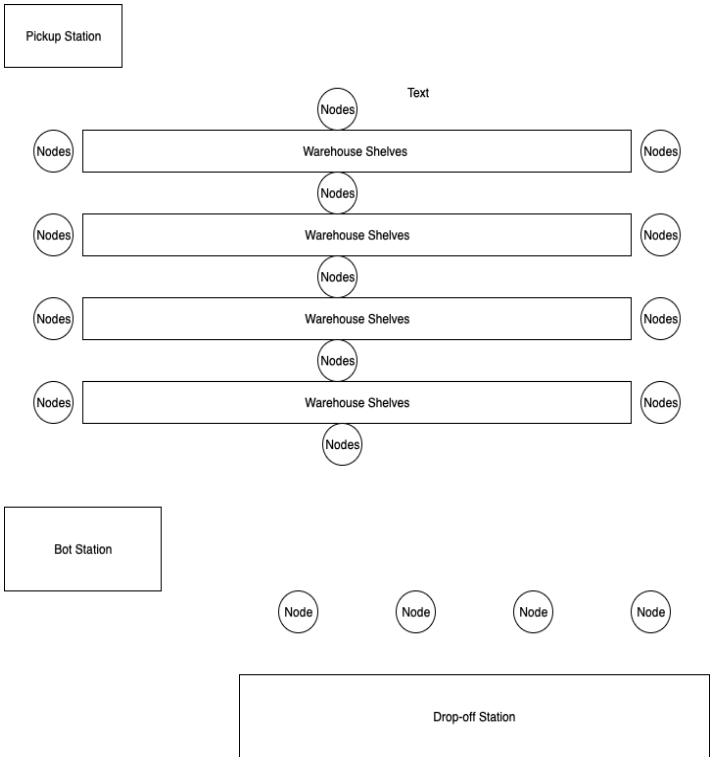

Fig. 1. Schematic of Warehouse in Simulation

Once it calculates the position it is supposed to place the cargo at, the bot then calculates the shortest available path it can take to reach the shelf. The bot places the cargo and returns to the bot station. The same process occurs during pickup. The bot-station receives a request from the pickup station. It calculates the path to the required shelf. Picks up the cargo and drops it off at the pick-up station.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

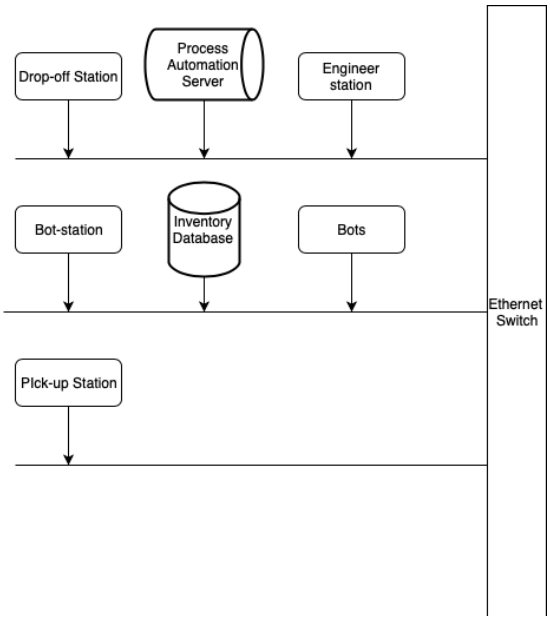

Fig. 2. Schematic of automation control system

IV.STACKING ALGORITHM

This module deals with the efficient placement of the cargo in the shelves. The simulated warehouse contains 4 shelves with cuboidal boxes of dimensions between 1 and 4 generated randomly. While placing the box, rotations of the box will also have to be considered to maximise the efficiency of placement. Tetris is one of the best selling video games of all time. It was created in the former Soviet Union and swept the west by storm in the 1980s. The game is a popular use of tetrominoes, the four-element case of polyominoes, which have been used in popular puzzles since at least 1907. A version of Tetris was made with Unity. The game which is a single player game was created in 2D. The tetrominos that were spawned at random fell sequentially at an increasing speed in a 20*10 grid.

The speed of fall of the tetrominos increases as it gets to the lower bottom or when the down arrow is pressed. Each Tetromino has 2 movements associated with it. A horizontal movement (horizontal shifts) and rotations which can be applied before the tetromino reaches the bottom or another tetromino. The game consists of 7 tetrominoes. They are the L J S Z I O tetrominoes. The L, J and T have 4 rotations each while the minos S, Z and I will have 2 rotations each, finally the O tetromino will have no rotations associated with it. After a player manages to fill a row in the grid that particular row is removed and the player is rewarded after completing a row. The speed of the game gradually increases until the player loses, ie. the pieces are higher than the grid.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

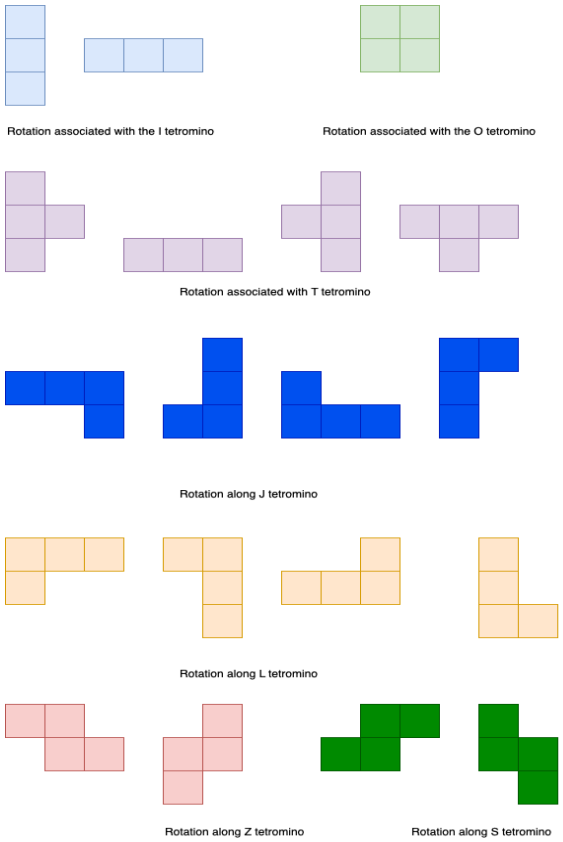

Fig. 3. Tetromino pieces and rotation

The general approach to developing AI for Tetris is a minimax based search and evaluation adapted for one-player games. It is assumed that the game always chooses its next move (the next falling piece to place) at random. Depending on the game implementation, the search process relies on a onepiece or two-piece evaluation function. The first case assumes that the player only knows the current state of the board as well as the current piece to place. The second one adds the information about the next piece to be played.

A deep overview of the existing computational AI controllers developed for Tetris can be found in [1]. The most notable contributions include a hand-coded evaluation function that includes a linear combination of five game board features whose weights are manually tuned by trial and error. Evolutionary techniques have been also presented for optimizing the weights in the evaluation functions according to the average number of completed lines of several play outs [2]. Other approaches apply reinforcement learning techniques or antcolony optimization instead. Neural Networks, widely used for implementing controllers for a varied set of games, have been proposed combined with reinforcement learning for creating a Tetris agent [3].

An evolutionary heuristic approach was chosen for designing the Tetris AI.

V.THE EVOLUTIONARY HEURISTIC

The AI designed is a one-step look ahead maximization heuristic used to calculate the optimal placement for a given piece. The AI uses the current game state and the next tetromino to place and calculates all legal placements for that

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

piece. The heuristic then approximates the quality of each of these simulated states, returning the placement that leads to the next best state. The AI utilises 4 features that were extracted from the game state.

1) Aggregate Height: This heuristic returns how high a grid is. To compute the aggregate height, the sum of the height of each column(the distance from the highest tile in each column to the bottom of the grid) is calculated. This value is to be minimised as a lower aggregate height means more pieces into the grid before hitting the top of the grid.
2) Completed Lines: This is probably the most intuitive heuristic among the four. It is simply the number of complete lines in a grid. This heuristic is to be maximised as clearing the lines is the goal of the AI.
3) Holes: A hole is defined as an empty space such that there is at least one tile in the same column above it. Due to the nature of tetris, a hole is harder to clear as all the lines above the game will have to be cleared before the hole can be reached. This heuristic is to be minimized.
4) Bumpiness: A deep ”well” in the tetris grid is undesirable. The presence of these wells indicate that lines that can be cleared easily aren’t cleared. If a well were to be covered, all the rows which the well spans will hard to clear. The bumpiness of a grid tells us the variation of its column heights. It is computed by summing up the absolute differences between all two adjacent columns. To ensure that the top of the grid is as monotone as possible, the AI will try to minimize this value.

These features are linearly combined in order to evaluate any given board game S as:

$$Score(S) = \lambda 1.c1(S) + \lambda 2c2(S) + \lambda 3c3(S) + \lambda 4c4(S)$$

where λ element of [0,1], $\forall$ i = 1,2,3,4 and λ1, λ2, λ3, λ4 are the weights for the features c1 c2 c3 c4, in the state S.

A real-valued GA is applied to obtain an optimal weights set that maximizes the controller performance. The GA individuals are vectors containing heuristic weights sets. Individuals fitness is obtained from the controller performance using individual’s heuristic weights set. The GA uses a population of 24 individuals, initially generated at random by sampling values in the range [0, 1]. Individuals are normalized so that all weights always sum 1. Tournament Selection, Weighted average crossover, mutation operator and delete n last replacement have been employed.

The fitness calculation is a non-deterministic process, since the same individual can perform very differently in subsequent play outs. For this reason, every individual is evaluated once at every generation. The latest fitness score is cumulative with the previous one by following an aging fitness function. This aging function sequentially increase the fitness accuracy evaluation after evaluation. Consequently, those individuals that underperform are easily revealed and discarded. The genetic algorithm stops when the population average fitness do not improve in 50 generations.

After this process the weights found at the end of this process to calculate the total score to place our boxes efficiently.

Algorithm 1 : Training Algorithm
procedure GENETICALGORITHM
t ← 0
Initialise the population P(t) with random weights
while*termination conditionis not met* **do**
begin
Evaluate fitness of each member of population P(t)
Select members from population P(t) based on fitness
Produce the offspring based off these genetic pairs
Replace based on the fitness
candidates of P(t) with these offspring
set time t:= t+1
end
end

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

VI.ROBOTICS ALGORITHM

The cargo is to be placed by small bots that can pickup the cargo from pickup stations and place them at the point of placement. The bots work parallel with each other and require path planning algorithms to prevent deadlocks and more importantly prevent collisions at all costs. The spaces in between shelves have paths for only 1 bot to pass through at a given amount of time. At a given time for N shelves we have N bots working at the warehouse.

This problem can be solved in 2 approaches, the first was an overly simplistic approach which can be utilised for small to medium size warehouses and the other utilises the A* algorithm for path planning and semaphore and mutex locks for synchronisation to prevent collision.

Algorithm 2:Stacking Algorithm
procedure STACKING
Result: Index at which box is placed
begin
t ← 0
i ← 0
Initialise rbox with box dimensions
while*t not grid length***do**
begin
calculate score on position t
add score to s[i]
Produce the offspring based off these genetic pairs
increment t
end
end
return index of greatest value of s
end

In the simplistic approach for path planning, N shelves are defined for N-1 bots. Nodes are defined at various positions in the warehouse. When a bot picks up cargo from the pick up point, it merely picks up the cargo and proceeds to the closest node that is free. From there it proceeds in a clockwise fashion to the closest node that is free until the desired node is reached. This approach produces no deadlock and has no collision associated with it.

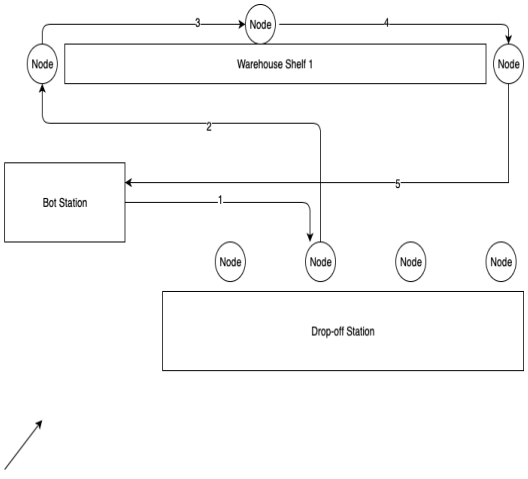

Fig. 4. Simplistic path planning

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

This approach is not scalable and is extremely inefficient to implement in real life. So the A* algorithm for path planning is implemented in the simulation.

Path planning is one of the most important studied problems in the field of autonomous robots. The autonomous robot should pass around obstacles from a given starting position to a given target position, touching none of them, i.e. the goal is to find a collision-free path from the starting to the target position. Research on path planning has generated many fundamentally different approaches to the solution of this problem, in which A* algorithm [7][8] is the one of the outstanding approaches have been developed for solving this problem. As in our schematics of our warehouse shown above, we define nodes around our warehouse for path planning. In order to prevent deadlock each bot utilises bounded waiting. Once a bot receives its destination node, it navigates its path

Algorithm 3:Path Planning
procedure PATH PLANNING
Result: Path the nodes have to take to reach the destination.
begin
Update graph
Initialise open list
Initialise closed list
Add start node to open lost
while *open list is not empty***do**
begin
find node with least f on the open list, call it g
pop g off the open list
Generate gs 8 successors and set their parents to g
for each successor **do**
if successor is the goal **then**
stop search
successor.g = q.g + dist b/w successor and q
successor.h = distance from goal to successor.
end
if*node e with same position as successor***then**
if *e in the OPEN list and a lower f than successor***then**
skip this successor
end
if e is in CLOSED list with f < successor **then**
Skip this successor
end
else
add e to the open list
end
end
push q on the closed list
end

using the current graph(ie only the free nodes). Once all the resources have been collected. The bot navigates the path. After passing each node, it sets its flag to Free. An example based on our schematic is shown below. In this (fig 5) example, the bot has to navigate between the green nodes. The red nodes are the nodes that aren't available. The red nodes are locked by another bot. (fig 6)This is the final path found by the bot in the graph. This path planning algorithm is both efficient and scalable. This method is also applicable to all types of warehouses and manages to virtually eliminate collision and deadlocks.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

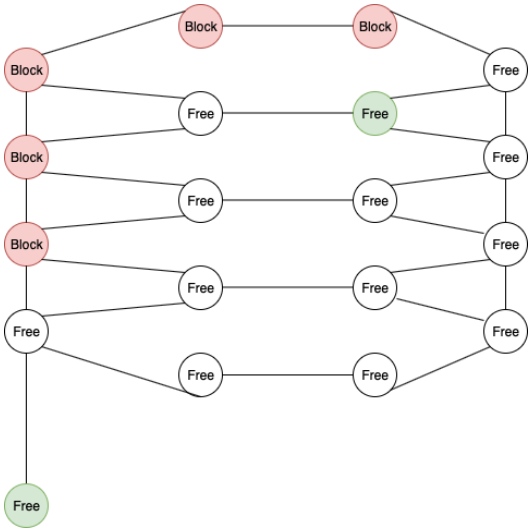

Fig. 5. Path planning initiation

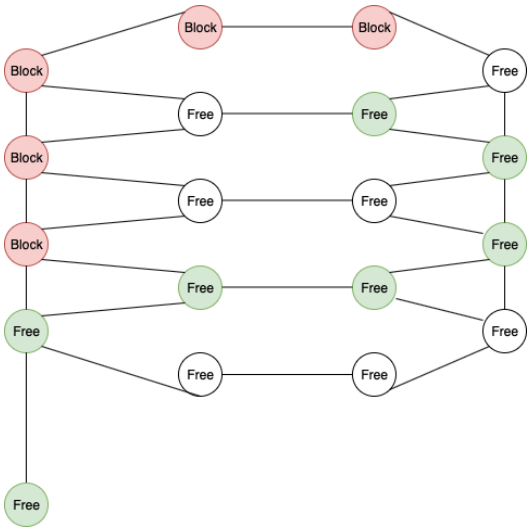

Fig. 5. Path finalised

VII.CONCLUSION

In this paper, an automation control system for warehouse automation was developed independently and it has shown promising results in simulations. It can be applied for various warehouses. There was an extensive survey on various stacking techniques to help improve the accuracy and speed of stacking. Various path planning approaches were also explored to improve the speed, accuracy and parallel working of bots. Fully automated warehouses can be buit off of this simulation. Fig. 5. Path planning initiation Fig. 6. Path finalised

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(A High Impact Factor, Monthly, Peer Reviewed Journal)

Website: www.ijircce.com

Vol. 7, Issue 5, May 2019

REFERENCES

[1] Thiery and B.Scherrer .Building controllers for tetris .IcgaJournal, 32(1):311, 2009.
[2] .Bohm,G.Ko kai and .Mand An evolutionary Approach to Tetris . In The Sixth Metaheuristics International Conference (MIC2005), page 5, 2005
[3] .Lundgaard and B. McKee. Reinforcement learning and neural networks for tetris. Technical report, Technical Report, University of Oklahoma, 2006.
[4] Jose M. Font, Daniel Manrique, Sergio Larrodera. Towards a Hybrid Neural and Evolutionary Heuristic Approach for Playing Tile-matching Puzzle Games
[5] Modeling Decisions in Games Using Reinforcement Learning HimanshuSingal, Palvi Aggarwal, VarunDutt
[6] Development and Application of Automation Control System to Plate Production Line. Jiao Zhi-Jie, He Chun-yu, Wang Jun, Zhao Zhong
[7] .Stentz, The Focussed D* Algorithm for Real- Time Replanning, Pro- 5 ceedings of the Interna- tional Joint Conference on Artificial Intelligence, pp. 16521659, 1995.
[8] . Maxim, G. Geoff and T. Sebastian, ARA*: Anytime A* search with provable bounds on sub- optimality, Proceedings of Conference on Neural Information Processing Systems (NIPS), Cam- bridge, MA, 2003. MIT