CHAPTER 1

INTRODUCTION

Chain Reaction is a highly popular game. It's most interesting feature is that any static evaluation for board positions is highly volatile and can change dramatically as the result of one single move. This causes serious problems for conventional game-tree search methods. In this work, we explore an innovative approach using Automata Theory to determine advantageous moves and analyse the game progression for any future heuristic development. In computer science, artificial intelligence, and mathematical optimization, a heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed.

The objective of the game is to remove all of the opponent's pieces from the board by causing chain reactions of explosions. A general scenario of the game is shown in Figure 1.1 and described below.

- The board is initially empty, and each player has a number of pieces of a specific colour.
- Each square on the board is assigned a 'capacity' value depending on the number of neighbours it has.
- Each player then takes turns in placing pieces of their respective colours on the board. This can be done either on an empty square or one on which that player's piece is present.
- Every time a move is played, a check for explosions must be performed according to the capacity of the square on which it is placed. If the piece explodes, its fragments get added to each of its neighbouring squares. After this the check for explosions must be done recursively till the board reaches a stable state.
- The game ends when the board is composed of pieces of a single colour.

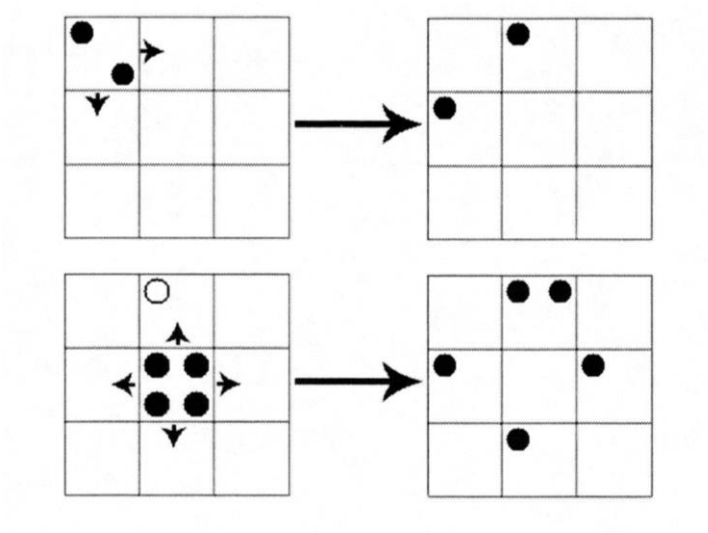

Figure 1.1 A typical game scenario

There are two separate approaches to defining the automata for the game. The first approach is defining the states for a particular piece. These set of states that the piece undergoes is position dependent. The sample set of states are: initial, grouped and exploded. However, due to the logic of the game, a particular piece's state depends on its neighbours' states also. This is where the second approach of the game comes into play. Here, we define the state of the game itself. This is dependent on the states of it's composite pieces and on every move of the player i.e. the input. The game ends when the game reaches the final state.

1.1 Report Organisation

Chapter 2 contains the Literature Survey giving brief summary on the game and how automata are used in general in mobile and computer games. We have referred to previous works on how automata are crucial in game development. Chapter 3 defines the problem statement. Chapter 4 defines the methodology and the Finite state Automata that is developed for the game. Chapter 5 describes the requirements and design and how the game is implemented. Chapter 6 consists of screenshots of the implemented game. Finally, Chapter 7 includes conclusion and future work.

CHAPTER 2

LITERATURE SURVEY

2.1 Previous Studies

Computational and Automata Theory is an exhaustive field. Deterministic Finite Automata, or DFAs, have a rich background in terms of the mathematical theory underlying their development and use. Examining real-world applications of DFAs helps to gain an appreciation of the usefulness of this theoretical concept. DFA uses include protocol analysis, text parsing, video game character behaviour, security analysis, CPU control units, natural language processing, and speech recognition. Additionally, many simple or often complex mechanical devices are frequently designed and implemented using DFAs, such as elevators, vending machines, and traffic-sensitive traffic lights. DFAs naturally lend themselves to concisely representing any system which must maintain an internal definition of state. This makes it an optimal method for analysing game states and behaviour. It could be used in the various stages of design of a new arcade or board game or to formally depict the various possibilities of a game. It can also be used for creating the Artificial Intelligence system for a game. Although graph based analysis methods have been studied previously in the form of minmax algorithms, they have not been thought of in terms of automata. In the present paper, the game chain reaction is being studied and its various states analysed. This analysis is important in a multiplayer game to efficiently predict which player will win the game or in a uniplayer game to design the heuristics of the AI.

Several previous works have led to the inception of the idea for the paper. S. Turocy[1] explains the need for formal modelling of a situation as a game requires the decision-maker to enumerate explicitly the players and their strategic options, and to consider their preferences. John Nash[2] demonstrated that equilibrium points exist in all finite games. At this point all players choose actions which are best for them given their opponents' choices. [3] E. Anitha defines the use of automata in electronic and android games. Much of the game theory is concerned with finite, discrete games which have a finite number of players, moves, events and outcomes etc. Many researchers from multidisciplinary fields have worked and applied the game theory in construction

engineering and management domains. In [4] Dafyd Jenkins shows how the game under study, i.e. Chain Reaction can be analysed using Monte Carlo Analysis and Minimax algorithms. Although the method of analysis is strikingly different, the motive behind the study is similar and thus has been taken into consideration. The study mentioned focuses primarily on the high volatility present in the game. This means that in many positions it is difficult to evaluate a given game position since on the outset though it may seem like Player A is winning, in a single move with subsequent Player B may win. This important characteristic of the game is what makes it interesting as a game and to analyse.

2.2 Outcome of Literature Review

It has been observed that the chain reaction game has not yet been looked at in terms of a finite state machine. However being inherently stateful, this characteristic can be studied so that future studies for the Artificial Intelligence of the game can be conducted. It could be compared to the existing algorithms for AI for any improvement or other heuristic strategies if possible.

CHAPTER 3

PROBLEM DEFINITION

The most important issue to be addressed while analysing the game is it's volatility. Any static evaluation for board positions is highly volatile and can change dramatically as the result of one single move. This causes serious problems for conventional game-tree search methods. Whilst Chain Reaction has very simple rules and game play, it has an “easy to learn; hard to master” complexity to it. It has an extremely high branch factor which means that there are too many possible states of the board which makes it difficult to analyse. It also does not have a decreasing complexity as the game progresses (unlike Chess where the complexity decreases as pieces are eliminated). The complexity only generally decreases during the extremely short end-game of Chain Reaction (usually one or two moves).

This Project attempts to analyse the possible state transitions for the game heuristic of the game. It may be used to efficiently predict the winner of the game. Alternately, it can also be used to build the AI for a uniplayer version of the game. Also, this could be the basis for designing a new board or arcade game derived or inspired from the current.

CHAPTER 4

METHODOLOGY

4.1 Finite State Automata for the Game

The following grids will determine the states and the state transitions in the heuristic of the game.

- Start State : No ball on the board
- Grid contains ball < critical mass of the grid
- Grid with ball = critical mass of grid
- Explosion of 1 colour ball into a grid of another colour ball

Let the NFA be defined as: $N= (Q, \Sigma, \delta, Q_o, F)$

We now consider the above conditions to define the possible states in a two player game.

States: Q

1. Start state: No Ball on the board
2. S10: Player A with 0 ball on grid with critical mass=2
3. S11: Player A with 1 ball on grid with critical mass=2
4. S20: Player A with 0 ball on grid with critical mass=3
5. S21: Player A with 1 ball on grid with critical mass=3
6. S22: Player A with 2 balls on grid with critical mass=3
7. S30: Player A with 0 ball on grid with critical mass=4
8. S31: Player A with 1 ball on grid with critical mass=4
9. S32: Player A with 2 balls on grid with critical mass=4
10. S33: Player A with 3 balls on grid with critical mass=4
11. S'10: Player B with 0 ball on grid with critical mass=2
12. S'11: Player B with 1 ball on grid with critical mass=2
13. S'20: Player B with 0 ball on grid with critical mass=3
14. S'21: Player B with 1 ball on grid with critical mass=3

15. S'22: Player B with 2 balls on grid with critical mass=3
16. S'30: Player B with 0 ball on grid with critical mass=4
17. S'31: Player B with 1 ball on grid with critical mass=4
18. S'32: Player B with 2 balls on grid with critical mass=4
19. S'33: Player B with 3 balls on grid with critical mass=4

Alphabets of Transition: $\sum$

Transitions in state can occur only with the addition of a ball by a player or due to explosion from another grid. Thus the finite state machine cane be modelled with alphabets a and b.

'S' defines addition of a ball by player a.

'S" defines addition of a ball by player b.

Transitions due to explosions from other grids can be modelled using epsilon transitions.

Transition Functions:

The transition functions are described as shown in the epsilon-NFA in the Figure 4.1.

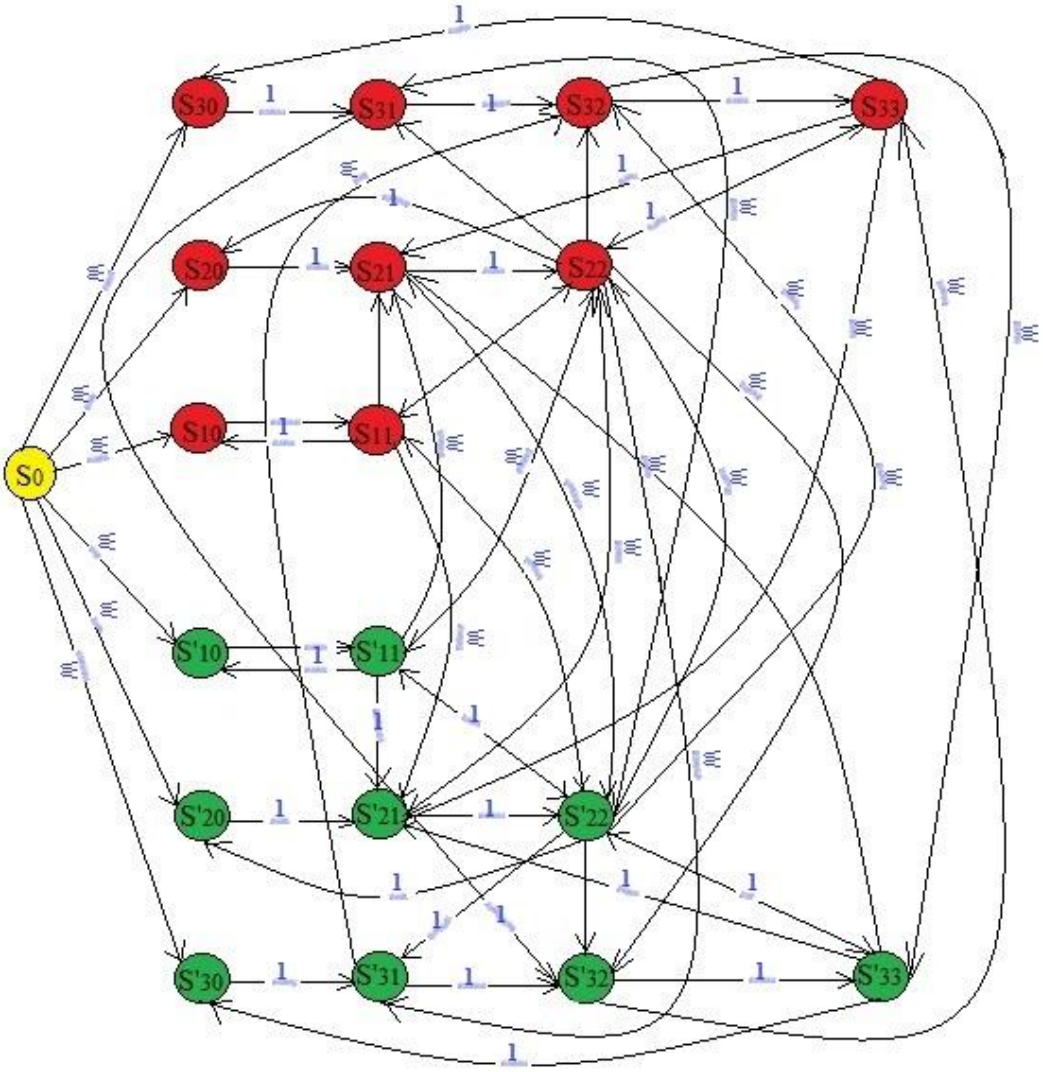

Figure 4.1: Epsilon NFA for the game

Start State: Q_o

The epsilon NFA has a start state when grid has no ball placed. The overall start state for the game is when each grid has no ball.

Final State: F

The epsilon-NFA defines the heuristic of the game. A heuristic cannot have an end in itself. Thus the NFA has no final state.

Condition for Game Over: Even though the state machine model of the heuristic does not have any defined end state, the game ends when all the grids on the board belong to states of a single player i.e. either AbB or AaB.

CHAPTER 5

REQUIREMENTS AND DESIGN

5.1 Requirements

- Operating system: 64bit Windows Linux environment with a Browser.
- RAM: 4.0 GB.

5.2 Design

The algorithm designed for the game first takes input from the player and based on the input builds the matrix for the game. Initially the matrix is empty. When Player 1 adds a ball in the grid, it is stored in a temporary variable. It checks the position of the ball on the matrix and the number of balls already present in the square. Based on this recursion occurs for adjacent positions. This is checked for all positions on the board.

Algorithm 1: Chain Reaction game using Automata.
Input: The grid size in terms of rows and columns.
Output: The grid with respective number of rows and columns.

```
1.  Recur()
2.  void Recur( position)
3.      Add 1 ball to current position
4.      Store in temp
5.      if position is corner and temp > 1
6.          Recur() for 2 adjacent positions
7.      if position is on an edge and temp> 2
8.          Recur() for 3 adjacent positions
9.      if  position is in the central portion and temp>3
10.         Recur() for 4 adjacent positions
11.     if all atoms of same colour
12.         Print winning message
```

CHAPTER 6

IMPLEMENTATION

The Graphical user interface of the game includes a square board with equal grids per row and column. With each move of the player, a piece must be added to the board in the correct position. In addition to this, recurring explosions must be displayed when necessary. Thus, these graphical components have been implemented in the project using Hypertext Mark-up Language (HTML). Additional styling for better user experience has been done using Cascading Style Sheets.

The core features of the game i.e. the grid size, the grouping of cells according to position and player colours have been stored in a JavaScript file. It also takes care of computing the explosion checks, drawing and animation involved after every move. Thus the complete implementation of the Finite State Automata for the heuristic of the game would be implemented using functions coded in JavaScript.

Function state1() checks if the square is corner square and returns 1 if it corner square and 0 otherwise. Function state2() checks if the square is edge square and returns 1 if it edge square and 0 otherwise. Function state3() checks if the square is interior square and returns 1 if it interior square and 0 otherwise. Function stateexplode() checks if any of the corner, edge or interior states have balls equal to critical mass. Function startt() starts the game for a player. An active player takes turns to play according to the alive_user array. Function AnimateGrid() animates each square with the required number of balls. Function rebuild() builds the entire grid after a user inputs a ball and state transitions occur. Function gameover() is used for checking if all the squares recurs within the state of the same player.

An example game has been played for understanding of the algorithm. The Figure 7.1 shows the status of the board mid game. Here splitting of interior atoms has been shown. Figure 7.2 shows effect of this move on the neighbouring balls. Figure 7.3 shows the status of the board just before end game has been reached. Finally, Figure 7.4 shows the end of game with the appropriate message.

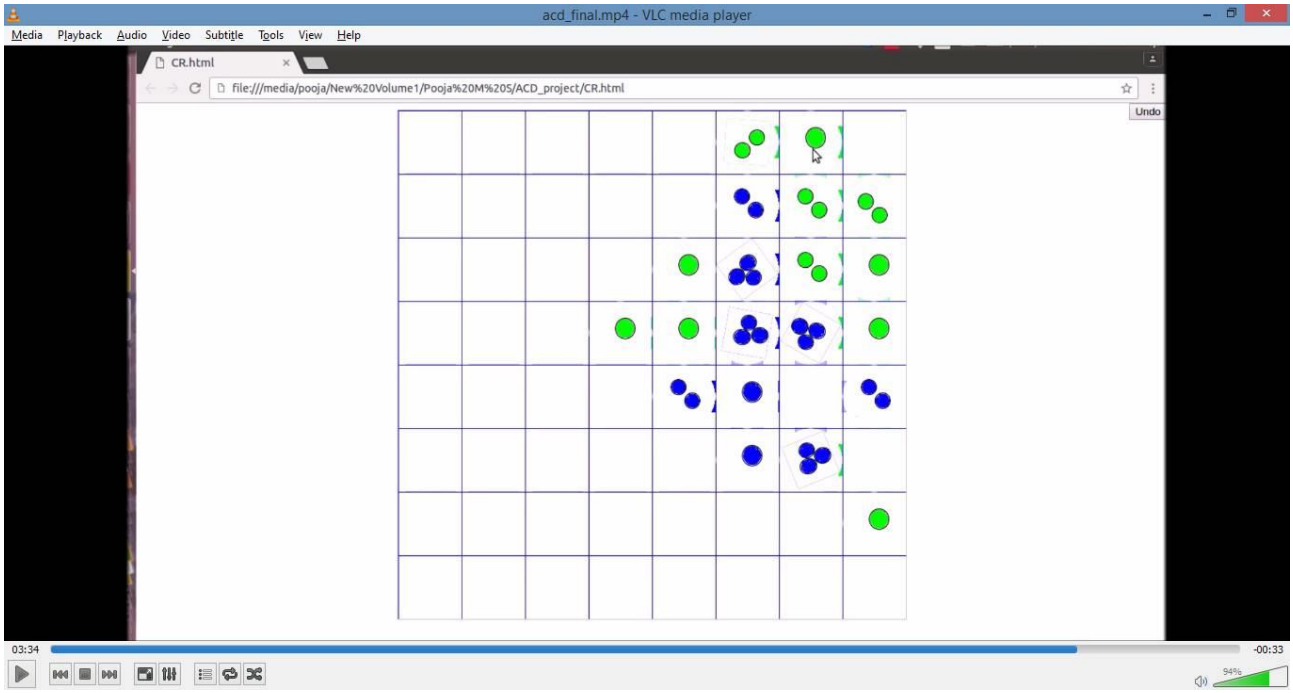

Figure 7.1: Screen shot showing splitting of an interior blue group of balls.

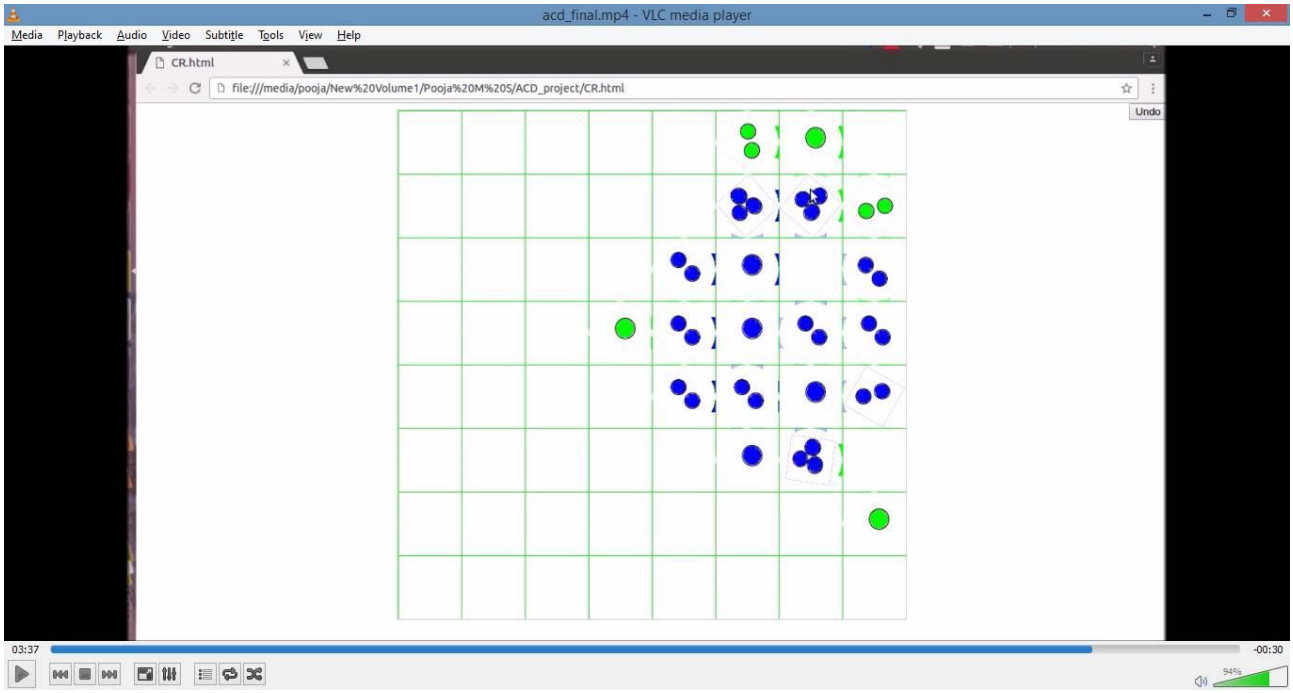

Figure 7.2: Screen shot showing effect of move shown in Figure 7.1 on neighbouring green balls.

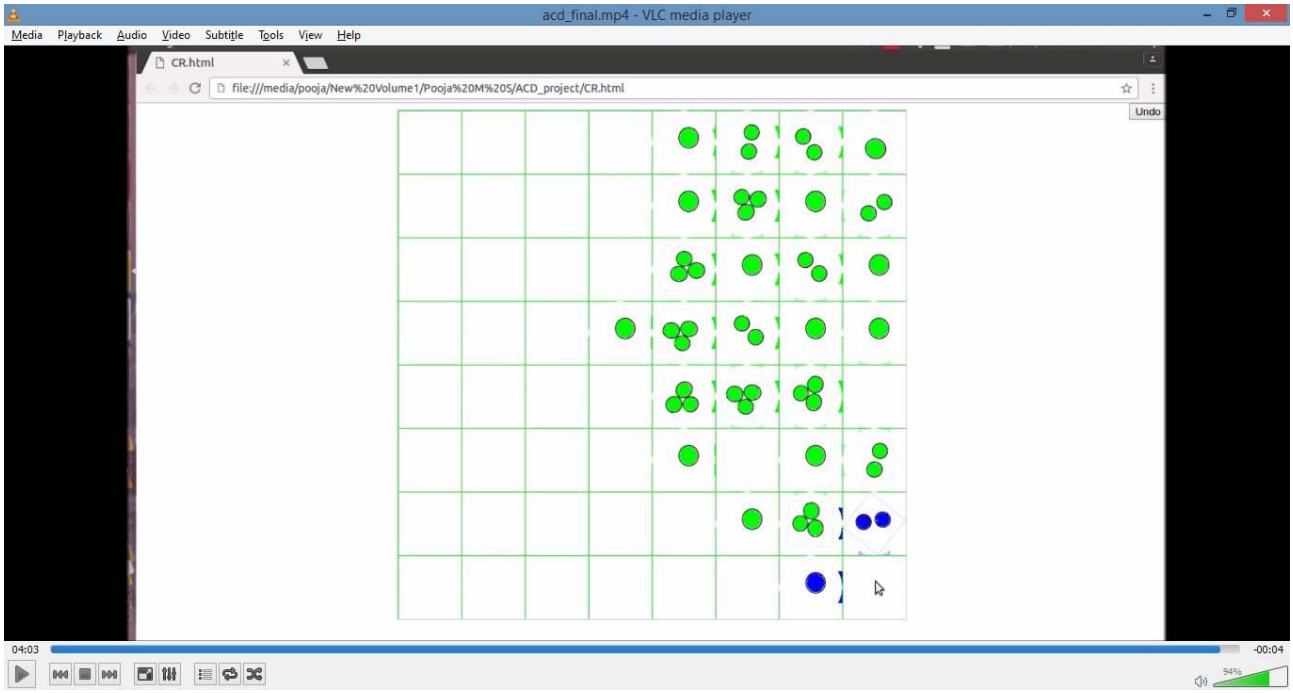

Figure 7.3: Screen shot showing the board just before the end of game.

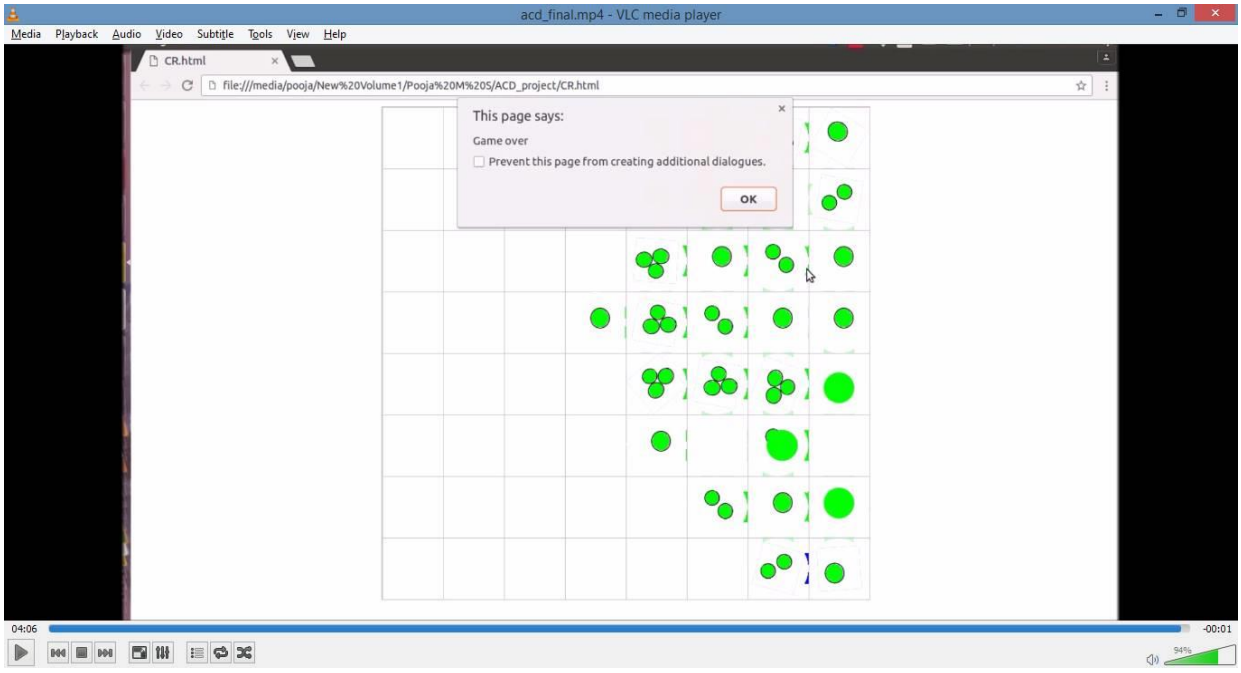

Figure 7.4: Screen shot showing the board on end of game.

CHAPTER 7

RESULT

An NFA has been developed for the Chain Reaction game. This approach is quite different from the ones that have been used previously, namely the decision tree approaches like Minmax and Monte Carlo analysis. Though the method used in Minmax greatly reduces the iteration space and is arguably faster, the method used in the current work is much simpler. This fresh approach will facilitate the development of an AI for a uniplayer game since the game has been developed for two players in the present work.

CHAPTER 8

CONCLUSION AND FUTURE WORK

The progression of the game itself occurs in a series of states. This inherent linking with the automaton concept makes it suitable for studying and modelling the state transitions involved. The future work for the Project can be taken up in several ways. An algorithm predicting the outcome of the game or rather which player shall win can be designed. A heuristic for the AI of the uniplayer version of the game can be thought of. A different arcade or board game derived from the concept can also be implemented.

REFERENCES

[1] T. L. B. v. S. Turocy, "Game Theory," Encyclopedia of Information Systems, Academic Press, 2002.

[2] J. C. H. R. S. J. W. W. E. v. D. J. F. N. P. H. Harold W. Kuhn, "The Work of John Nash in Game Theory," in journal of economic theory, 1994.

[3].E Anith,"Application of Automata for electronic and Android games, International Journal of Computing Algorithm",2014.

[4] Dafyd Jenkins and Colin Frank, "Highly Volatile Game Tree Search in Chain Reaction" http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4100131&tag=1 ,May 2006.