2019

SNAILS GAME

Artificial Intelligence for Games

Naveed Hakim 15026393

Hamza Safdar 15026413

Khizar Farooq 15026422

Muhammad Usman 15026435

Submitted to: Dr. Junaid Akhtar

27 March, 2019

1. Synopsis:

We intend to develop a two player artificially intelligent game named as Snails. Snails is a two-player game with 8*8 board. Players have to cover maximum area in order to win the game. However, we built AI-Based game that is played between intelligent human being and artificially intelligent agent. Our goal was to make our agent intelligent as the agent may play against human with equal intelligence. To achieve this goal, minimax algorithm along with Heuristic functions is implemented that gives satisfactory results.

2. User Manual:

2.1. Game Overview:

Snails is a desktop game that is played between two players, one of the players can be artificially intelligent agent. Snails is a two-player game with 8*8 board. Each player has a snail that can be moved across the board and leaves it trace at its path. Main objective of both players is to cover maximum area over the board in order to win.

2.2. Rules of Game:

- General Game rules are given below:
- From his current position, player can step up, down, right or left in any unoccupied box.
- By occupying one empty box, player will get score of one.
- If any player have series of occupied boxes in row or column, he can jump to last box of that series from his current position towards both ends. However, move cannot be done in any middle box of that series.
- Player cannot move across diagonals.
- By doing these moves, goal is to maximize number of boxes occupied by player.
- Filling all boxes is terminating condition.

2.3. How to Start New Game:

New game can be started by just opening Snails.py that is provided in package.

2.4. User Interface

2.5. Start Playing:

First turn is of human player. You have to click on the adjacent box of blue snail and it will move to that box only if it is valid according to above mentioned rules. Red snail is of AI agent and after each human move; it will take its turn automatically.

2.6. Wining Condition:

Game will end if any of user takes more than 32 boxes. The one with more than 32 boxes occupied is declared winner.

3. Background

Literature describes that people are too much interested in playing games that are human vs computer and interests are increasing in developing artificially intelligent games that

may play with human being like another human being. Users demand that artificial agent should not be built upon some specific sequence of moves because that will be boring for them because user can crack it and the charm of game is lost. Agent should give as challenging competition as any other intelligent human being may give to someone.

To meet these requirements following techniques were studied and used.

3.1. Minimax Algorithm:

Minmax algorithm is mostly used for decision making in games. This algorithm works on a basic and simple rule that is of maximizing one's own score and minimizing opponent scores to win the game.

Minmax first tries to do wining move if it is not possible it tries to lead game to draw or loose otherwise. When a human player plays his move and turn for agent comes, min max algorithm is applied to find best move by virtually performing all possible moves to certain defined depth at the backend.

Search trees are built level by level. One level covers all possible moves of agent and next level covers all possible moves of opponent by changing turn. Scores are calculated on each move for each node of tree. In the end, we select the move whose branch returns maximum points for agent and minimum points for opponent.

3.2. Limitations of Min Max Algorithm for Snails game:

There are some limitations of the algorithm when it is applied for snail's game.

- As this game have an 8*8 board, number of maximum possible states exceed drastically that takes huge computation power.
- In snail's game, there is a rule that if any player have series of occupied boxes in row or column; he can jump to last box of that series from his current position towards both ends. Once his snail is at one end he can again move to other end and this can lead agent into same iterations and it can make some type of infinite loop that will not only lead agent towards losing the game but also can take lots of computation power.
- Although, it gets current best move that brings victory closer but still it cannot guarantee that agent will win always.

To handle these issues following steps have been taken:

3.3 Depth of search Tree:

Depth of search tree has been included that limits tree to go until leaf for taking max or min score. It limits tree until a certain depth (according to computing power that we may provide) that minimizes cost of algorithm.

3.4 Heuristics:

In order to make agent a bit smarter like human being heuristics are added along with search tree. This heuristic will work like human mind. It tries to cover as much area as it can. Now agent turn is dependent upon search tree and heuristic.

In heuristic this agent will see how much columns and rows are available for it to occupy. it will not work to occupy box rather, it will try to occupy maximum columns and rows to restrict the area of human. It will do this by counting available number of rows and columns and move in that respective direction.

4. Basic Architecture:

4.1. Two Player Game Design

In first phase of development, Game restricts both human, machine to abide by the rules, and calculates scores for both. In next phase, AI agent tries to confine other player in specific area and tries to block it path towards maximum of the area of board. When complete board is filled, score is calculated and player with maximum occupied boxes is declared as winner.

At start, an 8*8 grid is printed. This gird contains values of an array named as board that is initialized by zeros. However, at first and last indexes (row, cols) of board, both players are set respectively. Human player is set at first index and Ai Agent takes start from other side of board that is last index of board. To follow best game design, this board and grid are kept portable and sizes can be varied from 8*8 to any other square.

Player 1 can start the game by doing his move. In addition, trace of his path is kept by saving 11 in array for all the boxes that are path of snail 1. After human player, agent can do his turn and '22' is saved in to keep record of his trail.

Character	Representation
0	Empty grid
1	Human snail's head
2	Agent snail's head
11	Human covered area traces
22	Agent covered area trace

4.2. Basic Game Implementation:

4.2.1 Initialization:

'init' method is used to initialize necessary values for game. This method takes screen width, height, board title and board size. In addition, it initializes turn, score, and board then it creates a board state object that contains board and position of head of two players for further use.

Moreover, Images of snails are placed at their initial positions that are start and end of board in setup method.

4.2.2 Implementation of rules:

After initialization, game is started by human being and he plays first move by clicking on his snail and then by clicking in empty box at its right, left, up or down. This phase is consist of one major methods named as 'Check_In_Game_Matrix'.

Check_In_Game_Matrix:

Trigger: This method is called whenever there is a mouse click detected.

Arguments: This method takes x and y position of mouse click. That are basically row and column coordinates where mouse pointer is clicked.

Functionality:

If turn is 1 : 'Check_In_Game_Matrix' is responsible for taking coordinates from screen and sending them to 'MakeMove'. 'MakeMove' further checks games rules (section 2.2) and is responsible for making valid moves only. After making valid move, turn is changes

to two and score is increased by one for human player. After turn change, 'Check_In_Game_Matrix' is called again.

If turn is two: Turn two means that it is agent's turn and no mouse click is needed , system has to produce best move by using artificial intelligence(Section 5) implemented and then MakeMove.

Post Action: After method completes it, work once, on each valid move, value in board is first changes into one and then changes to 11 for human and vice versa is for AI agent to keep track of heads and traces.

Win or Lose: When complete board is filled, score is calculated and player with maximum occupied boxes is declared as winner.

5. Design and Implementation of AI agent:

AI agent is responsible of generating best possible move for artificially intelligent agent on his turn. Here we were having certain options for AI agent. One of them was to make agent do random move but that was obviously not good idea. Other was to teach agent some rules and let it move using that rules all the time But intelligent human being can figure patterns very quickly and certain hard coded rules were more easier for human players to guess next move of agent that would have made game quite boring . As goal was to make agent as intelligent as any other human being we decided to check all possible moves till the end of game virtually and chose the move that lead to maximum score according to out virtual calculation. This virtual calculation was done by using minmax algorithm (Section 3). To make AI agent better heuristics are added to game (Section 2.5). Design and Implementation details are given below:

5.2. AI Design:

In AI we have implemented minimax algorithm and added some heuristic. When turn of agent arrives, from current state all possible moves are made virtually. In this way one level of tree is made. After that, turn is changes and for each child of upper level, new children are generated that make all possible children of opponent move. Again, turn is changes, same process is repeated until one of the players wins or depth of tree is arrived. For each child score is calculated. Child with maximum score for agent is chosen for agent's turn and minimum score is returned for human player's turn.

5.3. AI implementation:

Generate Children:

Trigger: This function is called in 'Check_In_Game_Matrix' whenever turn is changed to two.

Arguments: Function takes boardstate as argument that is basically object of game state containing current board, positions of heads of agent and human and current turn.

Functionality: Generate children is responsible for generating all possible children of current state for any specific turn and returns all possible children with valid moves only. This is a helper function for minimax algorithm, as we have to check children of both turns virtually for calculating best possible move. Those children are returned in form of an array of arrays that contains all possible children.

Post Action: Score of each child is calculated until depth of tree through 'getscore' method. getscore method takes depth of tree and one child each time that are returned by generatechildren.

Getscore:

getscore takes state and depth in argument. It calls generateChildren for each of the children to generate children of that specific child in order to make a search tree. And decrements depth by one on each level. When depth touches zero this function calls 'evaluateboard 'and 'huristicsearch' functions. In the end it returns child with maximum score if turn is 2 and child with minimum score if turn is 1.

Evaluate Board:

Trigger: This function is called at depth zero of tree in getscore.

Arguments: state of game

Function: Evaluate board is responsible of calculating score of the state that is passed to it. It will iterate to all the index of the passed state and if it found '2' or '22' (agent filled boxes) in any index it will add one in the score and it will subtract one from the score and this score will eventually lead to the selection of intelligent move for agent.

Heuristic Search:

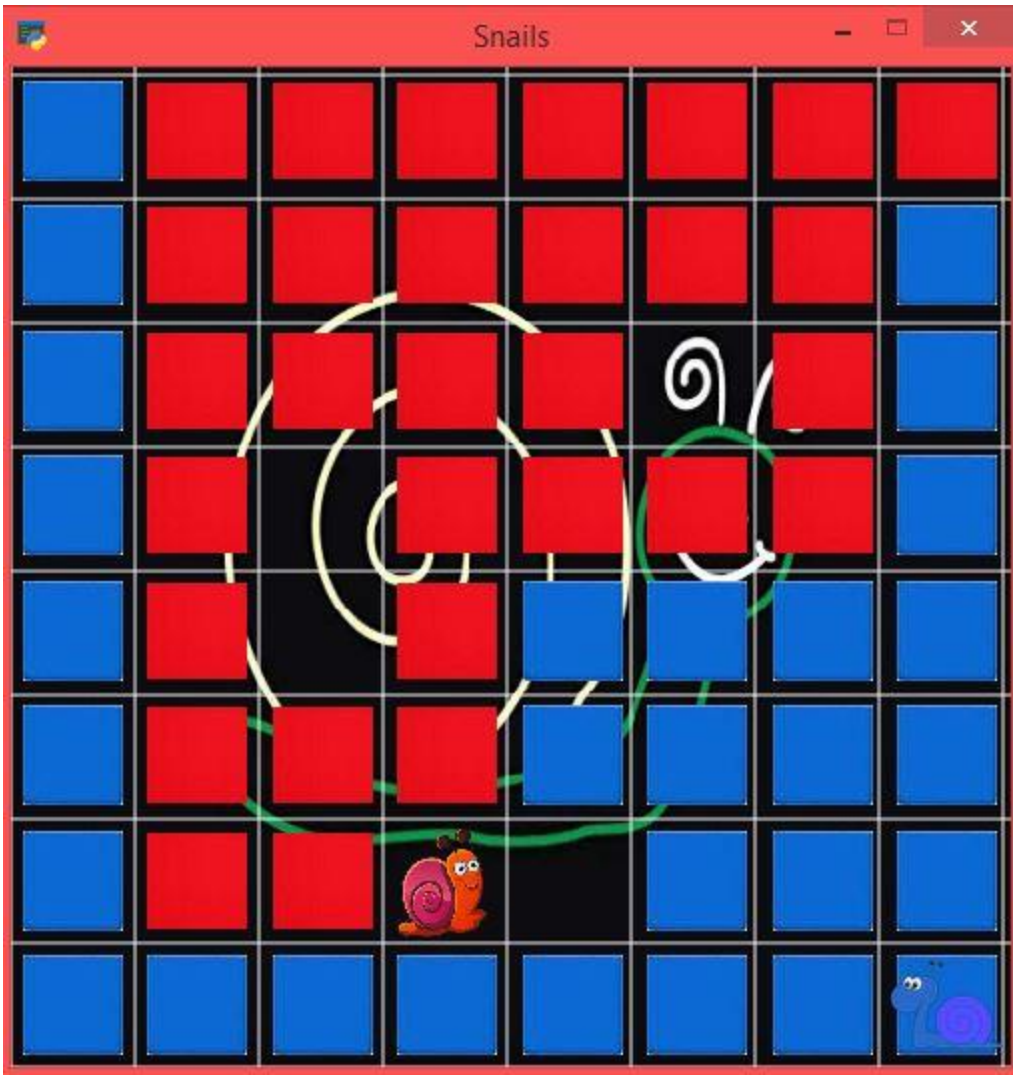
Trigger: called on tree depth zero in getscore.

Arguments: State of game

Function: It will take passed state game and tries to occupy maximum number of columns and rows to reduce the territory of human. For this in the state that is passed it will count the number of available columns and rows after this it will take half of calculated number and tries to occupy that number by moving in row and column. It didn't only move in column because if it only occupy columns then in the reaction of this move there is chances that human can cover all the below boxes and close the path for bot to come down. So, to overcome this agent will also move in rows.

6. Performance:

Multiple times testing was done to check performance of AI agent. It is noted that AI agent competes well with human and tries to beet it. Most of the time it beats human, however, even if it may not beat it does compete well.


Snails


Snails
GAME OVER
Agent Win !