Credit Hours System
CMPN402 Machine Intelligence

Cairo University
Faculty of Engineering

Machine Intelligence

“HexErs Team”

Submitted to: Dr. Nevine Darwish
Group ID: 1
Submission Date: 19 / 3 /2017

Table of Contents

Research

Problem Definition

We are required to design and implement an intelligent agent for playing the game of Hex on an 11*11 board using AI strategies and techniques.

Hex is a finite, perfect information game that belongs to the general category of connection games.
In 11×11 Hex, there are approximately 2.4×1056 possible legal positions; this compares to 4.6×1046 legal positions in chess.

Background information

Hex is a classic 2-player board game. It was originally invented by Piet Hein in 1942 and independently by John Nash in 1948. Since then, the game became a domain of Artificial Intelligence research.

The board takes a rhombus-shape and consists of n*n hexagon cells. Each player has several stones of some color and is assigned two opposing sides of the board with the same color. Players alternatively place stones on empty cells on the board. The one player that connects his two opposing sides of the board first with his stones wins the game.

Hex game cannot end with a draw which means there would always be a winner for the game. Consequently, by Nash's strategy-stealing argument, it could be proven that the first player in Hex game always has a winning strategy. That's why Hex the swap rule is often used in the game.

Previous work

Shannon's Hex machine

The first Hex playing machine was introduced in 1950 by the American Claude Shannon and E. F. Moore. It was essentially a resistance network with resistors representing edges and lightbulbs representing vertices. The move to be made corresponded to a certain specified saddle point in the network. Starting from here, researcher tried to develop computer algorithms that would be able to solve the Hex game emulating the Shannon network.

Hex playing automatons

Starting from 2000, and based on the various research into the game of Hex, the first Hex playing automatons started to appear. The first implementations used evaluation functions that emulated Shannon's network model along with alpha-beta pruning techniques. After the development of the Monte Carlo Go (another similar board game), strong Monte Carlo Hex players started to appear that were able to compete fiercely with the best alpha-beta Hex players.

Hexy

Hexy is a Hex playing program written by Vadim Anshelevich. It was gold medalist of the Computer Olympiads in 2000. Its approach is based on virtual connections.

MoHex

MoHex is a Monte Carlo UCT Hex player that was written in 2007 by Philip and Broderick with Ryan which are members in the Hex research group in Alberta university. In the UCT search tree it prunes moves via virtual connection and inferior cell information; it also handles solved states. In playouts, it uses only one local pattern (preserving a threatened bridge analogous to the miai move in Go). It uses the RAVE (all moves as first) heuristic and lock-free parallelization. MoHex won gold/gold/silver respectively at the '10/'09/'08 Computer Games Olympiads. Currently, MoHex is considered the strongest Hex playing program out there.

Proposed Search Tree Algorithms

Alpha-beta pruning algorithm

Alpha-beta pruning algorithm, which was developed to reduce the number of nodes to be evaluated bu the minimax algorithm, is commonly used in machine playing 2-player games such as Tic-tac-toe, Chess, Go, etc.

We studied Alpha-Beta pruning in class. Many enhancements were introduced to this algorithm help reduce the size of minimax trees. In the early Hex playing computer programs, Alpha-Beta search technique was widely used. Queenbee was one of the Hex playing programs using alpha-beta search. It was written by Jack van Rijswijck. Queenbee won silver at the London 2000 CGO.

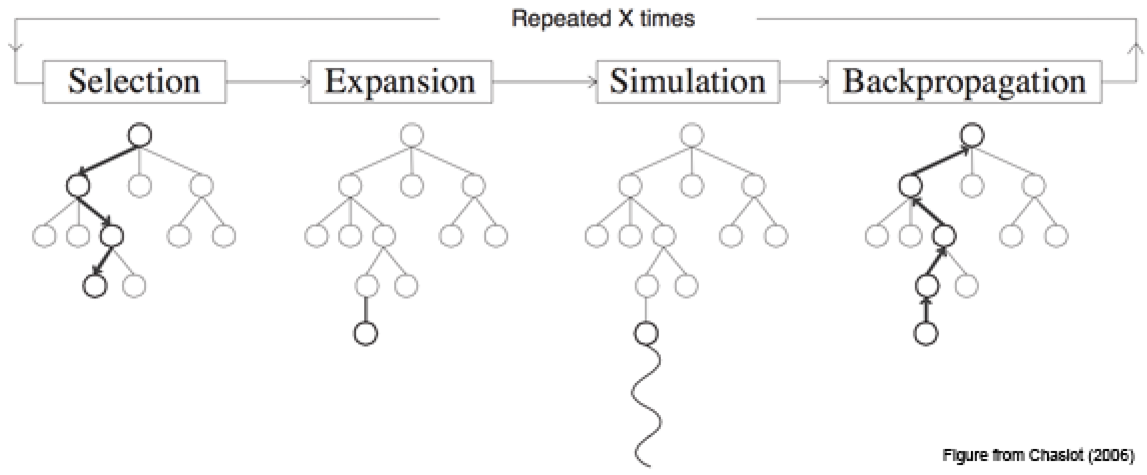
Monte Carlo tree search

Monte Carlo Tree Search (MCTS) is a method for making optimal decisions in artificial intelligence (AI) problems, typically move planning in combinatorial games. MCTS combines the generality of random simulation with the precision of tree search.

Starting from 2007 and after the revolution of Monte Carlo Go, Monte Carlo Hex players started to rise and were able to outperform the standard (alpha-beta) Hex programs.

Basic Algorithm

The basic MCTS algorithm is simplicity itself: a search tree is built, node by node, according to the outcomes of simulated playouts. The process can be broken down into the following steps:



1. **Selection**
   Starting at root node R, recursively select optimal child nodes (explained below) until a leaf node L is reached.
2. **Expansion**
   If L is a not a terminal node (i.e. it does not end the game) then create one or more child nodes and select one C.
3. **Simulation**
   Run a simulated playout from C until a result is achieved.
4. **Backpropagation**
   Update the current move sequence with the simulation result.

Each node must contain two important pieces of information: an estimated value based on simulation results and the number of times it has been visited.

Advantages

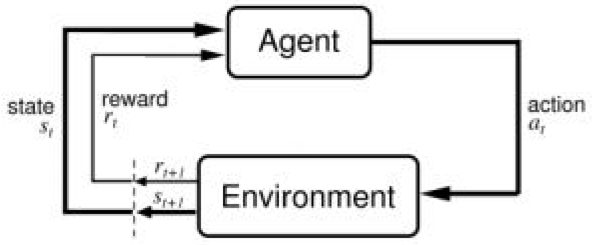
- Does not require any strategic or tactical knowledge about a given game (or other problem domain) to make reasonable move decisions.
- MCTS performs an asymmetric tree growth that adapts to the topology of the search space. The algorithm visits more interesting nodes more often, and focusses its search time in more relevant parts of the tree.
- The algorithm can be halted at any time to return the current best estimate. The search tree built thus far may be discarded or preserved for future reuse.
- The algorithm is extremely simple to implement

Disadvantages

- The MCTS algorithm, in its basic form, can fail to find reasonable moves for even games of medium complexity within a reasonable amount of time. This is mostly due to the sheer size of the combinatorial move space and the fact that key nodes may not be visited enough times to give reliable estimates.

Reinforcement learning:

Reinforcement Learning is a type of machine learning that allows you to create AI agents that learn from the environment by interacting with it. Just like how we learn to ride a bicycle, this kind of AI learns by trial and error.



Q learning

Q learning is a learning technique that can find an optimal state-action pair. Q learning does not need any initial utility values or probability values. The only parameters needed are the reward values for every possible game state.

Transition rule

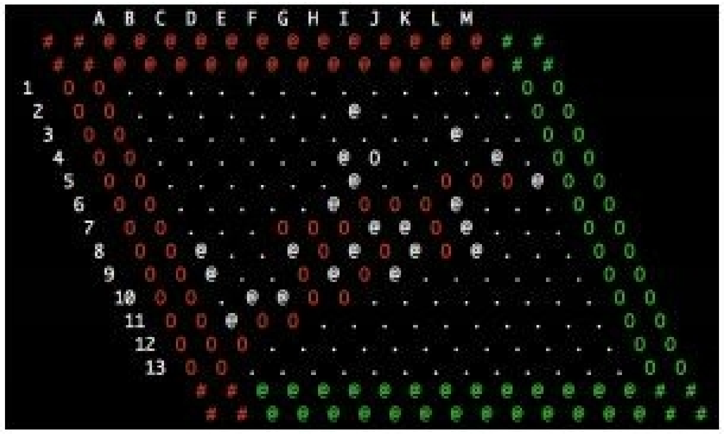
Q (state, action) = R (state, action) + Gamma * Max[Q(next state, all actions)]

Algorithm

1. Set the gamma parameter, and environment rewards in matrix R.
2. Initialize matrix Q to zero.
3. For each episode:
    - Select a random initial state.
    - Do While the goal state hasn't been reached.
    - Select one among all possible actions for the current state.
    - Using this possible action, consider going to the next state.
    - Get maximum Q value for this next state based on all possible actions.
    - Compute: Q (state, action) = R (state, action) + Gamma * Max [Q (next state, all actions)]
    - Set the next state as the current state.
    - End Do
4. End For

NeuroHex

NeuroHex is a deep learning Hex agent that was developed by DeepMinds Organization. Despite the large action and state space, the system trains a Q-network capable of strong play with no search.



```
         A B C D E F G H I J K L M
   # # @ @ @ @ @ @ @ @ @ @ @ @ @ # #
    # # @ @ @ @ @ @ @ @ @ @ @ @ @ # #
  1  0 0 . . . . . . . . . . . . . 0 0
   2  0 0 . . . . . . . @ . . . . . 0 0
    3  0 0 . . . . . . . . . . @ . . 0 0
     4  0 0 . . . . . . @ 0 . . . @ . 0 0
      5  0 0 . . . . . . @ . . 0 0 0 @ 0 0
       6  0 0 . . . . . @ 0 0 0 @ . . . 0 0
        7  0 0 . . . 0 0 0 @ @ 0 @ . . . 0 0
         8  0 0 @ . . @ 0 @ 0 @ 0 @ . . . 0 0
          9  0 0 @ . . 0 @ 0 @ . . . . . . 0 0
          10  0 0 . @ @ 0 0 . . . . . . . . 0 0
           11  0 0 @ 0 0 . . . . . . . . . . 0 0
            12  0 0 0 . . . . . . . . . . . . 0 0
             13  0 0 . . . . . . . . . . . . . 0 0
                  # # @ @ @ @ @ @ @ @ @ @ @ @ @ # #
                   # # @ @ @ @ @ @ @ @ @ @ @ @ @ # #
```

Results

After two weeks of Q-learning, NeuroHex achieves win-rates of 20.4% as first player and 2.1% as second player against a 1-second/move version of MoHex, the current ICGA Olympiad Hex champion. Data suggests further improvement might be possible with more training time.

Suggested tools

Fuego Libraries for agent logic module

Fuego is a collection of C++ libraries that facilitates the use of Monte Carlo tree search.

Unity

Unity is a s a cross-platform game engine that provides customizable and easy to use editor, graphical pipelines to DirectX and OpenGL, advanced physics engine. We recommend it to interface team.

Design

System Definition

Environment properties

- Environment: A Hex grid of size 11*11.
- Performance measure: the ability to bridge between connected cells in the grid.
- Sensors: clicks at GUI through mouse if playing against human agent or
- Actuators: coloring a selected cell.

Used Data Structures:

1. DSU: It is a graph algorithm that is very useful in situations in which you need to determine the connected components in a graph. Thus, it could be widely used in maintaining and constructing bridges between the played cells of the agent

2. Priority Queues: They will used to to sort the cells from up to down in case of connecting between horizontal borders and from left to right in case of connecting between vertical borders.

3. Stacks: It is used in DFS algorithm to check whether the current cell has a neighbor of the same color or not to check if there is a winning route or not.

4. Hash Sets: It can be used in implementing the function which is responsible for retrieving all the legal plays. Its usage purposes lies in the prevention of inserting any play/act more than once.

Agent Architecture

Agent consists of modules that communicate with each other through plugins and internal parameters:

1- Board module

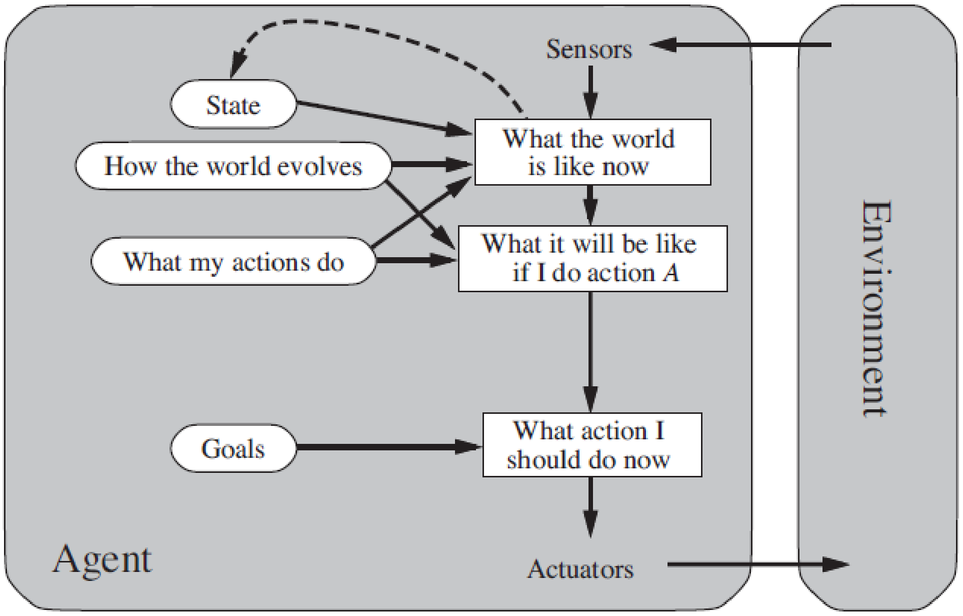
This module is responsible for the hex game logic. It contains the functions which manages the swapping rules, player turns, detection of winning states of any of the two players, updating the board with any step/play taken by the agent or the opponent.

2- Solver module

This module is responsible for preparing a list of good legal plays based on the well-known game heuristics and winning strategies. It then passes this list if legal plays to Monte Carlo (Agent Logic Module) to begin performing AI algorithms to come out with the decision of choosing which of this legal plays will most probably end the game with a winning state to out agent.

3- Agent logic module

This module is responsible for searching and planning for every upcoming moves. this agent is a *goal-based* agent which uses Monte Carlo search tree (MCST) as an algorithm for searching the states tree and applying Upper Confidence Bound (UCT) formula for balancing exploitation and exploration expansions through the search space.


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

4- Interface module:

This module is responsible for viewing the GUI for the user. If the agent is playing against human agent then this module will send the location of selected cell from human agent and send it to implementation module. It also receives from integration module the colors of each cell after every move.

5- Outer communication:

This module is responsible for the communication with opponent agent.

6- Control module:

This module is responsible for regulating actions to be taken between the three modules. It receives action from the other agent through outer communication module, send the updated data to agent logic, receives the best next move from agent logic, and send the last updated data as to interface module.

References

http://webdocs.cs.ualberta.ca/~hayward/hex/

http://maarup.net/thomas/hex/

http://www.dtic.upf.edu/~jonsson/dissertation.pdf

http://mnemstudio.org/path-finding-q-learning-tutorial.htm

http://webdocs.cs.ualberta.ca/~hayward/papers/mcts-hex.pdf

https://webdocs.cs.ualberta.ca/~hayward/papers/m2.pdf

https://www.cs.cornell.edu/courses/cs312/2002sp/lectures/rec21.htm

https://pdfs.semanticscholar.org/bb25/58b0f519ea921c4aff1197555153091f7177.pdf

https://www.cs.unm.edu/~aaron/downloads/qian_search.pdf

http://vanshel.com/Hexy/

http://www.cameronius.com/research/mcts/about/index.html

https://arxiv.org/pdf/1604.07097.pdf