Gotta Catch’em All Project Report

Team Member :	Ali Hassan T-13 28-11797
Team Member :	Omar Ihab T-13 28-3443
Team Member :	Abdelrahman Mahmoud T-08 28-10500

Date :	19/10/2016

Introduction

The purpose of the project is to create a search agent that is looking for Pokémons to collect inside a maze. This agent has to find all the Pokémons and then go to an end point before a certain time represented in an egg hatching time. At the beginning the agent starts at random location and so as the Pokémons, end point and the egg hatching time. The maze structure also is randomized every time the agents starts from the beginning. The job for this search agent is to find the shortest path in the maze between his starting point and the end point going through all the Pokémons in the maze assuming that the agent knows where are the Pokémons and where is the end point. The agent is asked also to create different search mechanisms to reach his goal and then display the difference between them in cost represented in number of steps and number of visited nodes in each mechanism. The search mechanisms are Breadth-first search, Depth-first search, Iterative deepening search, Uniform-cost search, Greedy search with at least three heuristics and A $*$ search with at least three admissible heuristics.

Tools used

We used haxe as the language to implement this project. Mainly because it's variety of cool features and it's nice syntax. We also used haxe-flixel to be able to present the problem visually. We used polygonal-ds libarary which contains some useful data structures.

- The problem:
    - The target of the searching was to find the shortest path where the agent can collect all pokemons and reach the goal and have his carried egg hatched.

- implementation of the search-tree node

- We implemented Node as a generic class. It takes the type of it's state. The reason we went with generic types is that we wanted to use Nodes in static functions plus it increases the code extendability and reliability.

    - Implementation : [state: T, parent: Node<T>, pathCost:Float = 0 , ?operator:Operator, depth:Int = 1]

State -> the state this node present
Node<T>: the parent Node .. should be of the same type
pathCost : the path cost till reaching this point. Default value is 0
Operator : the operator used to get to this node. Can be false in case of root
Depth : the depth of this node .. the default value is 1

- **implementation of the search problem ADT.**
    - The abstract problem class was implemented in class "Problem.hx". It's a generic class that takes the state class as it's type. We went this way because we needed the search functions to be static and we prefer generic classes over interfaces.
    - Implementation

    - search<T:State>(problem:Problem<T>, strategy:Strategy, visualize:Bool):Array<Operator>
    - The generic search algorithm .. it propogate the parameters to the search funcion hsearch but it's used to manage the iterative deepening strategy
    - hsearch<T:State>(problem:Problem<T>, strategy:Strategy, visualize:Bool, ?depth:Int= 0):Array<Operator>
      Where all the searching happens
    - public static function makeNode <T:State>(state: T, ?parent:Node<T>, ?pathCost:Float = 0, ?operator:Operator): Node<T>
      Creates a node

    - public static function expand <T:State>(problem:Problem<T>, node:Node<T>, operators:Array<Operator>, strategy:Strategy): Array<Node<T>>
      Expands a new level of the tree by applying all operators and sorting them based on the strategy

    - public static function isNotLoop <T:State> (node:Node<T>, state:T): Bool
      Used to prevent loops in the search tree by checking the new node will all it's ancestors

- **implementation of the Gotta Catch'em All problem**.
    - The problem implemented in the class `maze.hx` extends problem<MazeState> Because we are using out `mazeState` class.
    - Our maze class consisted of the following
        - pokemonsLocations:Array<Int> => array of pokemon locations in the map. We used int to store the locations for optimization. We also added functions to convert point to index and index to point
        - initialHatchingDistance:Int => the initial hatching distance
        - agentPos:Int => the initial position of the agent
        - agentDirection:Direction => the initial position of the agent
        - exitPos:Int => the initial position of the door
        - widthInTiles:Int => maze width in tiles
        - heightInTiles:Int => maze width in tiles

- **Implementation of the search strategies**
    - In search strategies we stored them all in an enum so they can be both readable and easy to check on. The sorting of the expanded nodes was done in the expand() function based on how low the value of the path cost and here how it went :
    - BreadthFirst : we add +1 to the parent cost so levels will be explored first
    - DepthFirst : we add -1 to the parent cost so leaves will be explored first
    - UniformCost; same as uniform cost since we considered all operators to be of the   same cost
    - Gready(id:Int) the path cost will equal the corresponding heuristic function result
    - AStar(id:Int) : the path cost will equal the corresponding heuristic function result plus the parent path cost

    - IterativeDeapening; was a special case since it had it had an additional argument .. so searching was done in while true loop with increased depth each time we run it.

- **Implementation of** of the heuristic functions:
    - We used 3 heuristic functions in both the greedy and the Astar strategies they can be found in the maze class. Here they are :
        - **manhatenHeuristics** : returns the distance to nearest pokemon or returns distance to door if all pokemons are acquired.
          Semantic Makes agent favors nearest pokemons direction.
          Admissibility: the agent needs to walk at least that distance to finish
          Advantages and disadvantages : very few computations but week heuristic value
        - **clusteringHeuristics** : returns the distance to the pokemon which has the minimum summation of distances to all pokemon or returns distance to doors if all pokemons are acquired.
          Semantic : Makes agent favors clusters of pokemons
          Admissibility : it assumes no obstacles
          Advantages and disadvantages : more computations and much better heuristic value depending on the type of the map.
        - **extendedManhatenHeuristics**: returns the optimal distance to goal state by choosing the distance to the nearest pokemon to agent and then choosing the distance to the nearest pokemon to it until all pokemons are acquired
          Semantic : Makes agent have far sight on how the whole path should go
          Admissibility : it assumes no obstacles are there which is not always the case. So the heuristic value is less than or equal the path.

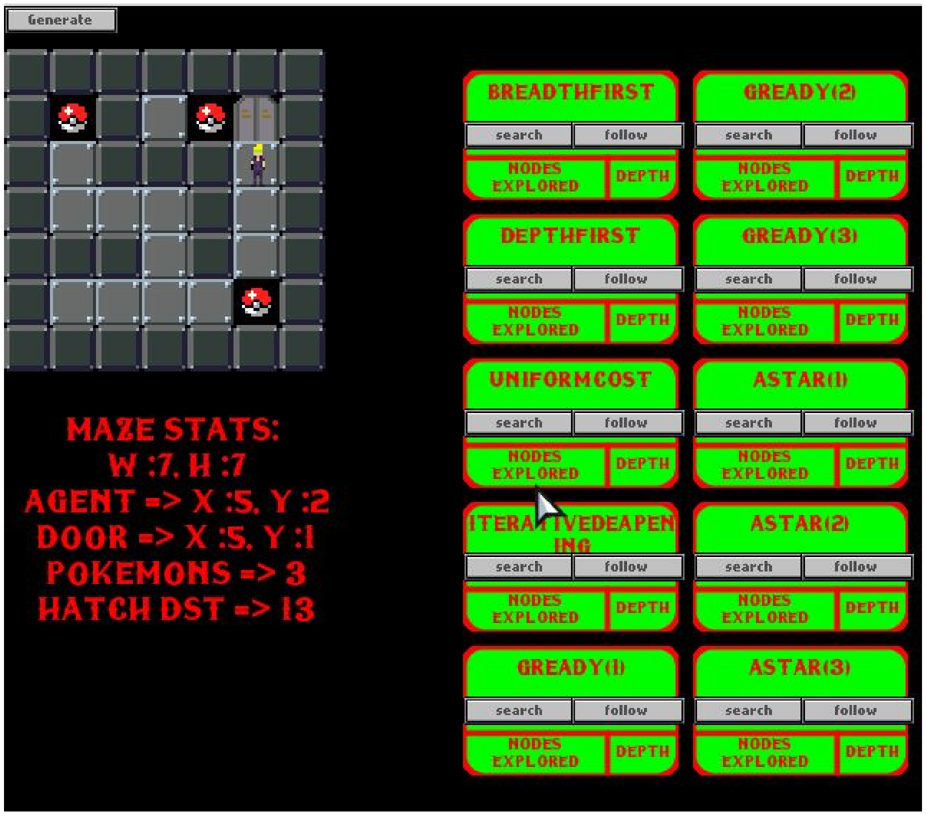
- **Observations:**
    - **Breadth first**: complete and optimal since all operators have the same cost. But it expand way too many useless nodes
    - **Depth first:** since we are using a perfect maze .. it usually produced a quick result so it can be labeled complete [almost]. But it's not optimal also in our settings it usually expands less nodes than depth first.
    - **Uniform cost :** same as breadth first since all operators have the same cost.
    - Iterative deeping : optimal and complete and explores more nods than depth first since it loops over nodes over and over every time it resets
    - Greedy : not optimal and not complete .. but explore nodes much less than breadth first . in heuristic h3 < h2 < h1 in exploring nodes
    - Astart : optimal and complete .. but explore nodes much less than breadth first but more than greedy. in heuristic h3 < h2 < h1 in exploring nodes

  Improvements
    - We could add the direction to the heuristic
    - We could change the operators to north,west, south, east and give them diffrent weights based on the current state which shall improve performance.
    - We could use another maze generation algorithm that doesn't produce perfect mazes since perfect maze have only way between each two points

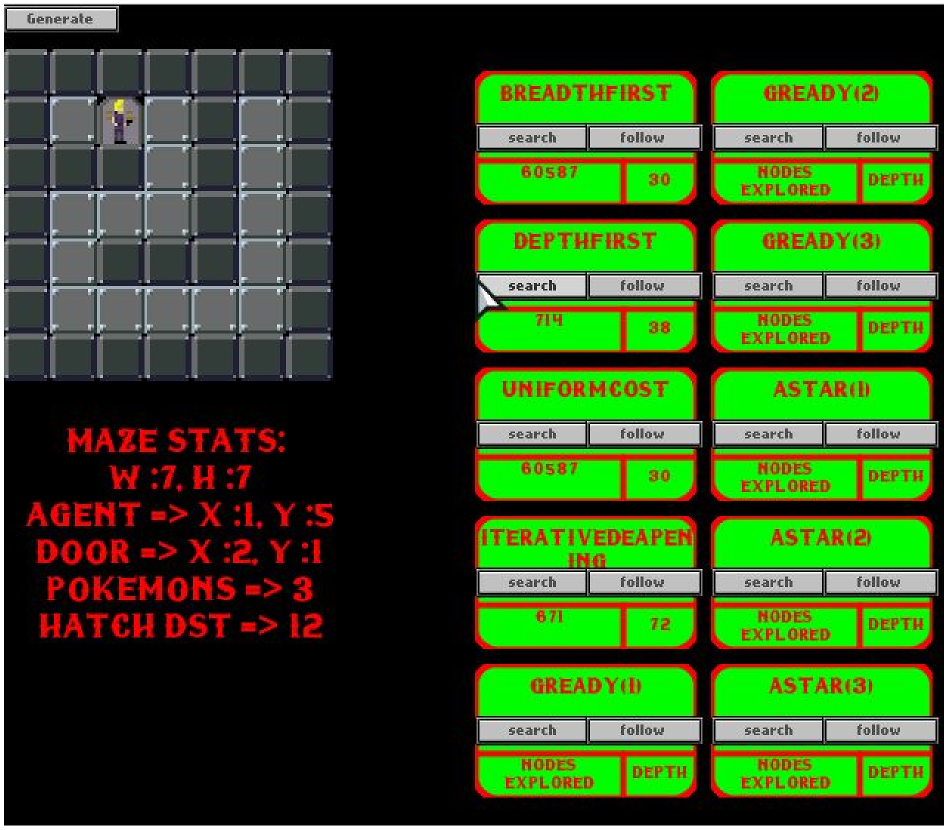
- **Source used :**
    - **Maze generation :** http://www.emanueleferonato.com/2015/06/30/pure-javascript-perfect-tile-maze-generation-with-a-bit-of-magic-thanks-to-phaser/

- **GUI**
    - **The game can be played using gui**
      Here are

Screenshots



MAZE STATS:
W :7, H :7
AGENT => X :5, Y :2
DOOR => X :5, Y :1
POKEMONS => 3
HATCH DST => 13



BREADTHFIRST: 60587 30
DEPTHFIRST: 714 38
UNIFORMCOST: 60587 30
ITERATIVEDEAPENING: 671 72

MAZE STATS:
W :7, H :7
AGENT => X :1, Y :5
DOOR => X :2, Y :1
POKEMONS => 3
HATCH DST => 12