**Artificial Intelligence in Games**

**Session 3**

1. **Evolutionary Algorithm (EA)**:
   1. EA is an optimization algorithm that considers complete solutions to a problem.
   2. Requires an objective function (also referred to as utility, evaluation, or fitness function) that can assign a numeric value to a solution, which can then be maximized or minimized.
   3. An optimization algorithm can be seen as a process that searches, in the space of solutions, for those with the highest (or lowest) value of that utility.
   4. *Hill Climber*: local search algorithm that moves continuously upward (increasing) until the best solution is attained. Workflow:
      1. *Initialization*: create a solution *s* by choosing a random point in the search space and evaluate its fitness *f(s)*.
      2. *Expansion*: generate and evaluate all possible neighbours of *s*. All possible single modifications that can be done to *s* to obtain an *s’* that differs from *s* by at most a certain given distance.
      3. *Iterate*: if at least 1 neighbour’s fitness *f(s’)* is better than *f(s)*, replace *s* with *s’* that has the best *f(s’)* observed, and return to step 2.
      4. *Return*: if none of the neighbours’ fitness *f(s’)* is better than *f(s)*, exit the algorithm and return *s*.

The limitations of deterministic hill climbing techniques include likeliness to get stuck in local optima due to their greedy acceptance of neighbouring moves.  
  
If we have a graph that is like a sine wave with different peaks and troughs, where the y-axis is the objective function and the x-axis in the state space. Local maximum and minimum problems occur when the algorithm only looks at the immediate surroundings (objective functions) on the graph whilst disregarding the whole state space, where there might be more maximums and minimums.

* 1. *Stochastic Hill Climber*: Stochastic Hill Climbing algorithm iterates the process of randomly selecting a neighbour for a candidate solution and only accept it if it results in an improvement.
     1. *Initialization*: create a solution *s* by choosing a random point in the search space and evaluate its fitness *f(s)*.
     2. *Mutation*: by using mutation, generate *s’*, one of the possible neighbours of *s*, and evaluate it to obtain *f(s’)*.
     3. *Iterate*: if *f(s’)* is equal or better than *f(s)*, replace *s* with *s’* and return to step 2.
     4. *Return*: if the computational budget is over, or *f(s)* is good enough, exit.

1. **Typical EA:**
   1. Initialize the population of N individuals, t = 0. 2.
   2. For each generation t, until convergence or max T:
      1. Evaluate the population Pt (i.e. evaluate all its individuals)
      2. Promote the best E individual(s) from Pt to the next generation’s population (Pt+1)
      3. For each N – E position in Pt+1:
         1. Select two parents from population Pt
         2. Create new individual(s) from them with crossover
         3. Mutate the individual(s)
         4. Insert the new individual in the next generation Pt+1
      4. Advance to the next generation (Pt → Pt+1).
   3. Return best individual from the final population Pt.
2. **Crossover**: combining genes from 2 (or more) individuals to create a new one (or more).
   1. Uniform: taking a gene from each parent, at random.
   2. Single point: choose a position in the individual at random. Then, take the first half from one parent, and the second half from another.
   3. N-point: choose N positions at random and repeat as above.
3. **Elitism**: select the best E individuals from the population and promote them to the next generation directly. Typically, E takes a small value to avoid a narrow convergence of the population.
4. **Rolling Horizon Evolutionary Algorithms (RHEA)**: RHEA is a family of algorithms that use evolution in real-time to recommend an action on each turn for the player to make.   
     
   RHEA is based on any Evolutionary Algorithm, repeated at every game tick:
   1. Each individual:
      1. Represents an action plan (sequence of actions).
      2. Is evaluated by executing the plan using the Forward Model, starting from the current state.
      3. Fitness: value returned by a heuristic function applied to the state reached after executing the action plan.

Plays the first action from the best individual found at the end of the evolutionary process.

RHEA is also a Statistical Forward Planning algorithm.

* 1. *Population seeding*: create seeded (add heuristic knowledge) initial population (set of sub-optimal solutions which are provided as inputs to a genetic algorithm and from which an optimal solution evolves).  
       
     Heuristic knowledge about the game.

Other methods:

• One Step Look Ahead (OSLA):

• Each gene is initialized with the best action at each step as determined by OSLA

• Monte Carlo Tree Search (MCTS)

• Initialize with the best sequence of actions found in the tree.

• Both provide a single individual: mutated (P-1) times to fill the starting population

* 1. *Bandit Based-Mutation*:
     1. A fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation and may become better understood as time passes or by allocating resources to the choice.
     2. Not preferred; vanilla RHEA works better.
  2. *Game Tree (first-level play)*:
     1. A type of recursive search function that examines all possible moves of a strategy game, and their results, to ascertain the optimal move.
     2. Generally good, improves vanilla RHEA.
  3. *Shift buffer*:
     1. Store evaluation process at each game step and add a random action at the end.
     2. Best enhancement, higher scores in most games.
  4. *Monte Carlo (MC) simulation*:
     1. Add a MC at the end of each evaluation.
     2. Always improve vanilla RHEA.

1. **Evolutionary MCTS**: Evolutionary MCTS (eMCTS) combines the tree search of MCTS with the genome-based approach of evolutionary algorithms.
   1. Each node in the tree is not a state: it’s a sequence of actions.
   2. Each link to a child does not correspond to an action, but to a mutation of the sequence of actions held in the parent.
   3. No Simulation step is executed: leaf nodes evaluated instead.

How is the root node generated: A greedy approach that determines a good starting point: OSLA.

What to do with illegal sequences: Repair mechanism also using OSLA.