MATH 384/484 Final Project

Computer Program That Learns to Play Angry Birds

by
Oguzcan Adabuk

In this project I implemented a program that learns to play Angry Birds. In order to create a replica of the original Angry Birds game, I used the game engine Unity3d. Unity3d has become a very popular game engine and it is free for developers who make less than $100,000 per year from their published titles.

I used C# programming language to write all my scripts for all game related code as well as the main learning (optimization) algorithm that enables the program to improve its game play performance.

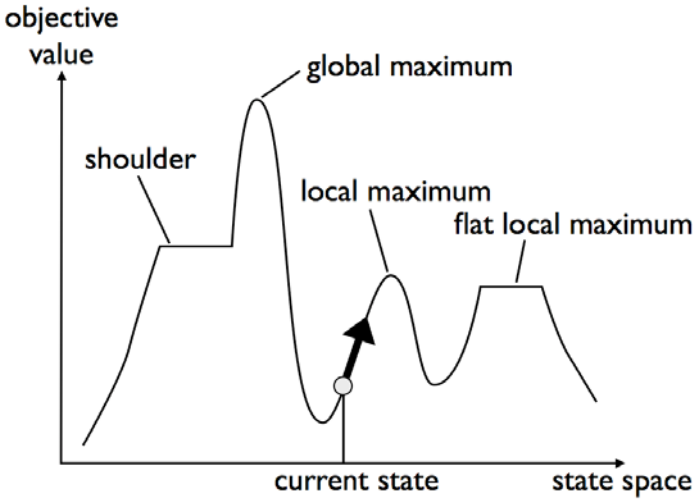
I was inspired by the Introduction to Artificial Intelligence course I teach at Northeastern Illinois University.

In this project I used genetic algorithm to improve computer player's performance. Genetic algorithm is an optimization algorithm that is inspired by biological concepts such as evolution, survival of the fittest, genetic crossover and mutation.

First step in the implementation is to represent states of a problem. To do this a proper data structure must be used. Each different configuration is called a solution but not all solutions are of the same quality. In order to determine solution quality a fitness function must be implemented.

After selecting a proper data structure and representing states, a population of solutions should be created. A population consists of many individual solutions. In general, these individual solutions are created randomly, assigned random configurations. These individuals make up the initial population. When creating an initial population, the size chosen is very important. A very large size may result in a very big converging time and intense use of computational resources. On the other hand, a small size will result in lack of genetic diversity of the population and this will limit the algorithm's capability of finding an optimal solution.

After the population is created we implement a fitness function. Fitness function determines the level of fitness for each individual solution. The general idea is that individuals with higher fitness level are favored for reproduction to produce more fit offspring until an optimal solution (a perfect individual) is found. Although genetic algorithm is not complete (doesn't guarantee a solution), it will optimize the population and the population at the end will have higher average fitness level than the initial population. Fitness function depends on the nature of the problem. For example, in the n-queens problem, fitness function assigns a score based on how many queens are attacking each other. In this case, fitness function is calculated based on several parameters: How close a bird landed to the pig structure, how much damage is done to the structure and how many pigs are killed. Each of these parameters have a different weight. Finding optimal weights for the fitness function is very critical, for example, if the weight of doing structural damage is more than the weight of killing pigs, then the program will try to destroy as many blocks as possible instead of accomplishing the main goal of this game, which is killing pigs.



Next step is to create a mating model where parents are picked from the population based on certain criteria and coupled to create new offspring. There are several ways of parent selection. Even though the goal should be to pick parents that perform well and have high fitness function, avoiding picking the best individuals repeatedly is crucial because this will result in lack of genetic diversity which will prevent the algorithm from finding an optimal solution. Choosing parents with a probability proportional to their fitness levels is one way to prevent this. Another solution is using a tournament selection where multiple individuals are randomly picked and the individual with the highest score in that group picked as a parent. In this project I used the tournament selection since I couldn't come up with an efficient way to determine what would be a top (optimal) score, and the difficulties associated with proportioning probabilities without a top score. Looking at the result I can say that the tournament selection worked well.

After parents are picked, crossover operators are applied. In crossover operation several genes are selected from both parents to makeup the child. I randomly choose which gene comes from which parent. I observed that usually the fitness of the new child is somewhere between the better parent and the worse parent. Some occasions it is better than both parents, in rare occasions it is worse than both parents.

After the child is created I again use the tournament selection but this time I use it to place the child into the population at the same time preserving the population size.

Mutation operator comes in handy for keeping the genetic diversity. Sometimes certain individuals start dominating a population, they end up passing their genes to a majority and this causes an early convergence. To prevent this situation I used mutation with a %5 probability. A gene of an new offspring is randomly changed with chance of %5.

Above you read an outline of genetic algorithm. In this section I will explain how I specifically implemented the algorithm for the Angry Birds playing computer program.

In order to get the best performance I had to implement genetic algorithm with two phases. First phase optimizes each individual bird. The second phase tries to find the best combination of different birds.

First phase has three parameters that are stored in an array. These parameters are shooting angle (0-90), shooting speed(0-300) and the powerup activation time (0-3 seconds).

Each configuration of birds are scored by running game and letting the computer play. After several generations computer is able to find birds that perform greatly. Usually 15-20 generations is sufficient with a population size of 50. This phase finds the optimal combinations of shooting angle, speed and powerup activation times. However Angry Birds is a sequential environment. Results of shooting a bird will effect the next bird. Combinations of different birds can be crucial to having a more complete solutions. For example, the first bird can destroy an important piece of block that covers a pig and the second bird can come in and kill that pig through the path that was opened by the first bird. Or one bird can do a lot of damage to the structure killing most pigs, the next one can aim to kill an isolated pig. Therefore in the second phase I used what I called tuples. Tuples are also represented with integer arrays. Each tuple has three elements in them, and these elements represent different birds. For example the first element of a tuple can store a specific bird which has its own array and parameters such as angle, speed, powerup time. This way the program finds the optimal sequence of birds with different parameters.