TDT4136 Introduksjon til kunstig intelligens
Assignment 3

Odd André Owren

September 2019

1 Introduction

1.1 Minimax

The minimax-algorithm is an algorithm that is designed to minimize the maximum gain in e.g. a turn-based game. The value returned from the minimax-algorithm is the largest value a player can be sure to get when they know the actions of the other player(s). The way the algorithm does this is by "simulating" a given number of moves ahead, and expecting that you want to maximize your score, while the opposing player(s) want to minimize your score. This way the player can choose the move that has the highest predicted score, given the situation.

1.2 Alpha-Beta Pruning

Alpha-Beta Pruning (ABP) is an algorithm that could be considered an extension of the minimax-algorithm. ABP starts out with the same principle as minimax, one player wants to maximize their score, the other player(s) want to minimize the score. ABP implements the use of two extra variables, Alpha and Beta, which is used as thresholds to cut off branches, such that one can minimize how many branches one needs to explore to make a decision. Alpha is the highest value that a player is guaranteed to get while exploring the tree, while Beta is the lowest value a player is guaranteed to get. This way one can compare the scores in leaf nodes to either alpha or beta, depending whether one should minimize or maximize the score. If the score you want to maximize is higher than beta, then the value is returned, and the rest of the branch is cut off. This can be done since one knows that if the other values are higher, they will still not be chosen above the beta-value, and there is no use to return any value that is higher even when maximizing. When minimizing, the score is compared to alpha, and if it is lower, the branch is cut off in the same way as when maximizing.
This makes ABP faster than minimax, as it reduces the amount of branches that have to be explored before an action is chosen.

2 Implementation

2.1 Minimax

My implementation of the minimax-algorithm is split into two functions; maximizer and explorer. The maximizer-function is used to maximize the score for pacman, through the use of the explorer-function. The explorer-function runs through all agents on the board (ghosts and pacman) and expects ghosts to choose the action that minimizes pacman's score and pacman to maximize his own. When the given depth is reached, the state is evaluated and the score at that point is returned to the recursive stack. This leads to the action with highest expected score after depth number of moves for pacman to be chosen through minimax.

2.2 Alpha-Beta Pruning

My implementation of ABP is based on the same premise as minimax, but uses a minimizer-function instead of explorer. It also makes use of alpha and beta as parameters in the functions, allowing the values to be passed between the functions when exploring the action-tree. Again, the maximizer-function tries to find the maximized value for pacman. When a new best score is found in current state, alpha is updated if the score is higher. The score is also compared to beta, and if the score is higher than beta, the branch is cut off and the score is returned. The best score will not be chosen over beta at the next level anyways, so there is no need to maximize any further. In the minimizer-function, the opposite happens. When a new best score is found, beta is updated if necessary, and the branch is cut off if the best score is lower than alpha. In total, the processing saved from using ABP instead of minimax equals about the same as cutting 1 level from the depth of minimax. When the depth is reached, the action with the highest expected score after depth number of moves is returned.

3 Results

First off, both implementations passed the tests that were included in the task. The results are the file "testCases.txt" included in the delivered zip-file.
As far as the agent performing optimally when adjusting different depths of the algortihms is debatable. In multiple cases the pacman-agent would stand still and wait for a ghost to chase it, then do a feint and go back to standing still. This lead to it waiting a long time before doing anything else. In other runs, pacman would decide to die pretty quick of the bat, as that was a way for it to minimize the losses it got by waiting before doing a move. At that point, the agent had realized it wasn't going to win, and tried to end it as quickly as possible, since living on came with score-deduction over time.
Most of the times, pacman would win, or at least not die, which I consider to be a fair enough for a game that is quite a lot more complex than many other turn-based games. I have not researched how the ghosts in the game decide their

actions, but as it is seemingly more random than pacman's actions, it is harder to predict which actions pacman should do, and in most cases the predicted best action ends up not being the actual, since the ghosts chooses something else than predicted, but that is hard to account for when implementing minimax and ABP.