Project Proposal
CMPU-365
Matthew Imiolek, Amy O'Connell

Basic Turn-Based Pac-Man

Game Design:

Due to the complexity of standard Pac-Man, we have decided to simplify the game such that it works better in the context of the ideas presented in class, specifically minimax search with alpha-beta pruning. To do this we plan to make the game turn based, decrease the size of the grid, reduce the number of ghosts, and adjust the ghosts' AI to fit this new context.

Turn-Based:

In order to make the game turn-based, we plan to replace the constant movement elements of Pac-Man with alternating grid-based movement. In a given turn, Pac-man and each ghost will move one square. We plan to make all of the characters move at the same speed. Otherwise, it will become significantly more difficult, perhaps impossible, for Pac-man to either escape or be caught. For all characters, the only option will be to move to an adjacent grid index without a wall marking. A wall marking's only purpose will be to designate spaces where characters cannot go.

Size of the Grid:

When implemented in our turn-based model, traditional Pac-Man is essentially a 28 x 30 grid. We plan to reduce the size of the grid to 15 x 15 in order to shorten games. In the original game, walls blocking movement take up the equivalent of two grid indices, but we plan to use only one for the sake of simplicity. It appears the choice to use two was primarily based on graphical preferences, which we will disregard.

Ghosts:

We plan to begin by implementing a game model with one ghost, whose goal is to cause Pac-man to achieve the lowest score possible. This is the direct opponent in the minimax function. We believe there is a large chance Pac-man will still win most if not all games in this situation, so we plan to add a second, simpler ghost which will simply move towards Pac-man as quickly as possible. This ghost will be designed so that it can be turned off and on, depending on the user's preferences. After this is implemented to create a mildly competitive game, we will attempt to add a copy of the first ghost that works in tandem with the original ghost to further minimize Pac-man's score. At this point, depending on how much time we have left, we will add a fourth ghost that moves randomly. In this case we will likely convert our model to use the expectiminimax function to account for the randomness.

Score:

In order to create the value used in the minimax function, there will be two primary modifiers: pellets and distance from ghosts. Pellets are the dots found in a standard Pac-man game, which are picked up to increase the score. Once all pellets have been collected, the level ends. We will be keeping these ideas roughly the same; picking up pellets will still increase the score. Collecting all pellets will cause the game to end. Picking up a pellet will cause a moderate increase in score on a turn by turn basis. In order to compel Pac-man to finish the game as quickly as possible, turns in which he does not pick up a pellet will cause a minor decrease in score. Any move that allows Pac-man to collect all the pellets will cause a massive addition to the current score, reflecting that the game has been won. This will also show how long it took to win the game based on cumulative score fluctuations from whether or not a pellet was collected a given turn or not. The second aspect of move evaluation is Pac-man's resulting proximity to a ghost. We plan to have moves increasing proximity to a ghost cause a greater decrease in score than would be overcome by picking up a pellet. This ensures Pac-man will prioritize his safety, as we believe that, due to the turn-based nature of our game, moving away from a ghost will be harder than normal and being closer to a ghost will close off more winning outcomes than it might in a traditional game. In the case that Pac-man does manage to get further away, we will reward this with a large increase in score, as it should open up more potential moves. In the case that the distance between Pac-man and a ghost becomes zero, the game will be over and Pac-man will lose, prompting a large reduction in the score. However, if Pac-man managed to pick up more pellets than in another situation, this score will be higher. It should also be noted that this setup creates a situation where the ghosts might position themselves to keep Pac-man from moving towards the pellets, thereby decreasing his score if it is the best strategy for them at the given time.

Testing:

In order to test our game, we will create situations where we can figure out the optimal outcome ourselves, and then test to see if the AI comes up with those same outcomes. Essentially we will be making chess problems, but for Pac-Man. In order to do this we will manipulate the positions of Pac-man and the ghosts, the number of pellets on the level, the ghosts used, and the location of the walls. For example, we will create situations where Pac-man may get stuck in a dead end depending on the locations of the ghosts and the depth of the search. We will have a general board design for use in actually playing the game if the user wants, and to see how the completed game runs without interference.

Minimax Search:

In order to have our game pick the optimal moves we plan to use a minimax search similar to the one we made previously. The difference between this model and the previous one is that to

counter to Pac-man's move we plan to eventually have two ghosts acting in a single move, leading to more possibilities and a greater complexity in how the game progresses compared to that of the a single action response to every move, as seen in chess. We will also implement a ghost that moves without the influence of the minimax algorithm, adding a level of complexity to the information Pac-man has to account for not present in the minimax we used in chess. We have also chosen to move to expectiminimax in the case that we have time to implement the ghost which moves randomly. We believe that converting our existing minimax model to expectiminimax will be far more doable in whatever time we will have left rather than switching algorithms to Monte-Carlo tree search. In both of these situations the heuristic used will involve the scoring system explained above.

Implementation Outline:

We plan to break Pac-Man up into a number of stages, each doing specific actions. The first will be the game stage, which will set up the board, the characters, the score, and start the games turns. The second stage will be a turn, which, due to the nature of Pac-Man as a single player game, will be considered every action from the start of one Pac-Man move to the beginning of the next Pac-Man move. This will be broken up into two parts, the Pac-Man move and the ghost move. During the Pac-Man move Pac-Man will use the minimax algorithm to pick the optimal move to make to increase his score. The ghosts move will consist of two parts: ghost-algorithm and ghost-other. In ghost-algorithm, any ghosts using minimax to pick an optimal move to reduce Pac-Man's score will make their moves. In ghost-other, any ghosts that don't require minimax will go. All of the information will be stored in a game struct.

For implementing the actual parts of the game, we will start by making a working game consisting of a board with walls, pellets, a score, and Pac-man. At this point Pac-man will simply move around and eat up all the pellets to succeed. We will then add the first ghost, al-ghosty-one, which will be the antagonist in minimax. Then we will add the second ghost, target-ghosty, which will just try to move as close to Pac-man as possible. Then we will ad the third ghost, al-ghosty-two, which will join al-ghosty-one as an antagonist in minimax. At this point if we have time we will convert minimax to expectiminimax and add Clyde, the ghost who moves randomly (there is a ghost in the original Pac-Man with this move pattern, and his name is Clyde, so we will name this ghost after him).

Citations:

Pittman, Jamey. “The Pac-Man Dossier.” Gamasutra, 23 Feb. 2009, www.gamasutra.com/view/feature/132330/the_pacman_dossier.php

Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, pp. 163–171, 177-180

http://ai.berkeley.edu/multiagent.html - Python project used by Berkeley which uses Pac-Man and minimax (basis of idea to do something with Pac-Man and minimax)