AI Group Project : Pac-man

Team name : I don't know

Team ID : 17

Members : 王軒 0816095、林佑庭 0816136、寸少康 0816144

[How the game AI works]

Mainly used limited depth dfs for eating pellets and ghosts. For every possible action (up, down left, right) at a given state(position in terms of 16x16 grid coordinate), use dfs to evaluate the highest possible reward of the next 8 step of each action.

Then pick the action with the highest potential reward within the next 8 step and make a move, rinse and repeat.

The reason of limited depth is of course time complexity problem, thus we combined numerous methods and heuristic function to optimize our algorithm.

[Experiments and experiences]

1. What if all pallets are out of reach within 8 step:

We used a heuristic funtion that calculates the manhattan distance of all the possible props still on the playground and move our pacman in that general direction, once they are within reach, the algorithm switches back to normal dfs.

2. What if pacman has eaten a power pallet:

When a player is in power mode, eating a ghost gives +200 points, which is huge compared to 10 points of a normal pallet, thus it is crucial to winning the game. So we give the ghsot a very high priority when in power mode, but ignore the ghost in the middle of the map that is invincible.

3. What if there is a room-like structure of the map:

Typical dfs has a blind spot where you can’ t explore the state that you’ ve already visited. This is a problem when there is a room-like structure that the entrance and exit are the same state. According to our algorithm the room is then marked at a very low reward, so it will never go in and collect the pallets. To solve this I slightly tweeked the dfs algorithm so that if there are absolutely no option at all it will just pass on the value instead of overwriting it as extremely low reward.

4. What if the algorithm tries to reverse:

It took me a good amount of time to realise that this game doesn’ t allow reverse motion at all(I misunderstand the spec as if we stop before reversing then it’ s allowed). To solve this I make a pair to keep track of the speed which the pacman is currently going to determine which direction is not allowed. Finally I combined the results with wall detection, so essentailly the pacman sees the reverse direction as a wall.

5. Time pruning:

To prevent time-out, I added a variable to keep track of how many times my algorithm has timed out. It shortens the dfs’ s depth every time a time-out happens. Because of this I’ m able to go deeper then usual to test the limits of the computer.

6. Landmines:

At first I thought the landmines are useless, but after some trial and errors, I find it actually pretty useful and halarious at the same time. I used a very short dfs towards our back direction and check if there is any ghost or player. If the return is true then place one landmine. It will only place another landmine if there is no one behind and someone or some ghost comes into our radar again. Seems simple but works better then expected.

[Contribution of individual team members]

王軒 : Code and report of pacman.

林佑庭 : Provide thoughts and debugging.

寸少康 : Provide thoughts and code for some function.