Modern AI - Individual Hand-in

Luka Locniskar

1 Introduction

GAMES and AI are intertwined fields [1]. This paper describes three approaches to designing and developing an agent for the Ms. Pac Man game. The three approaches are:

- Behavior Tree
- QLearning
- Monte Carlo Tree Search

Ms. Pac Man is a game where a yellow character navigates through a maze while collecting pills and avoiding ghosts. The game is won when all the pills are collected and is lost when the character collides with a ghost.

This is what makes the game an interesting challenge for AI development. The agent is working towards two goals at the same time and those goals are at odds with each other at times. There is also a meta goal which is reaching the highest score. That is achieved through the smart usage of the so called magic pill which, when eaten, makes the ghosts edible. On eating the ghosts a high score reward is given to the player.

The goal of this exercise was two fold. The first being to learn the described algorithms through their implementation. The other is creating an agent which could accumulate a high score and reach a high level.

2 Approach

2.1 Behavior Tree

The behaviour tree approach can hardly be classified as Artificial Intelligence. It is merely a specific conceptual approach to developing behaviour that may seem intelligent if done correctly. It is a tree of hierarchical nodes [3] where nodes are categorised as either leafs, decorator or composite nodes. Composite (Selector, Sequencer) and decorator (Repeater, Negator) nodes control the flow of the algorithm, leaf nodes are actual behaviours or commands. This abstraction allows the designer to split the behavior of an agent into tasks of various size and complexity at appropriate levels.

The behaviour tree for Ms.PacMan was designed with two higher level concepts as the core of the behavior. These are pill collection and ghost avoidance. These two behaviors are represented by Sequencers. The root node is a Selector node which prioritises ghost avoidance over pill collection as survival is deemed more important than increasing score. All the leaf nodes are:

- Check if in danger: which finds the closest ghost to Ms Pac-Man and if the distance is smaller than a certain value it returns true

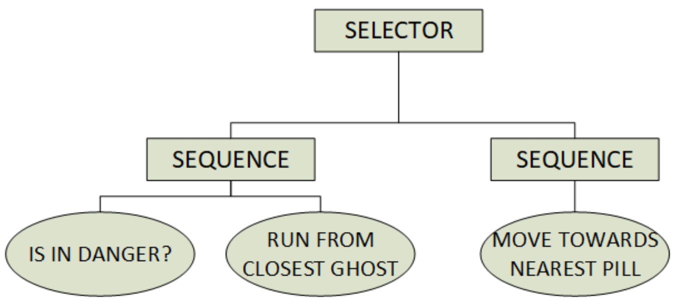

Fig. 1. Behavior Tree

- Run from ghost: makes Ms Pac-Man move in the opposite direction of the closest ghost
- Panic run: makes Ms Pac-Man run in the opposite direction of the closest ghost at a certain time but keeps her running away from that ghost for a certain amount of time
- Move towards closest pill: moves Ms Pac-Man towards the closest available pill

2.2 QLearning

QLearning falls under the family of Reinforcement Learning algorithms. It is a machine learning approach inspired by behaviorist psychology [1]. The core concept is rewarding the agent if it reaches a state we deem positive and punishing it if it reaches a state we deem negative. In implementation we differentiate between utility-based, Q-learning and reflex agents. [2]

This approach is modeled as a Markov decision process, [1] where we have a set of possible states and a set of all possible actions. Besides that we have a probability of transition function, which defines the probability of getting to state s2 from state s1 using action a and a reward function which provides a numeric representation for the reward/punishment of entering state s2 from state s1 using action a.

An important concept with any RL algorithm is state representation and evaluation. This concept makes implementing QLearning for ms. pacman quite difficult. The possible states are defined by:

- Ms Pac-Mans position
- Ghosts:
  - Positions
  - Are they edible
- Monte Carlo Tree Search

  - Positions (constant)
  - Are they available
  - Are they magic

The permutations of all these possibilities make the possible state space very large. Since the agent trained by QLearning cannot generalise it only knows the best possible actions for each state it has encountered during training. In the case of Ms Pac-Man this would mean that for a good agent it would need to be trained for an extensive duration with extensive resources. Since this was not a possibility, the approach taken was an abstraction of the game state in an effort to reduce their amount. The abstracted state representation contained the following information:

- Are ghosts close: boolean
- The positions of ghosts closer than a certain distance to Ms Pac-Man. Their positions are in Ms Pac-Mans coordinate frame (relative to her)
- The direction of the closest available pill: [Left, Right, Up, Down]

2.2.1 Monte Carlo Tree Search

Monte Carlo Tree Search is a method for finding optimal decisions in a given domain. [4]. Generally, algorithms that search through the space of possibilities and evaluate them by quality are limited by the complexity and possible moves of a game. Monte Carlo tries to mitigate this flaw by only searching through a subset of decisions and building the search tree iteratively and asymmetrically [4]. The algorithm used for Ms Pac-Man is upper confidence bounds for trees or UCT, which balances exploration of not expanded nodes and exploitation of best nodes well. The algorithm is split into distinct parts:

- Selection
- Expansion
- Simulation
- Backup

During selection a node is chosen based on the tree policy. That node is then expanded, which means all of the possible actions that can be taken from that node (state) are added to it as children. A default policy is then run from the new child. A default policy is the simulation of the game from the state, represented by a node, to a terminal game state or to a certain depth. When the terminal state/depth is reached, the state is evaluated with a score which is then backed up to the root node.

The structure of the algorithm represents a few issue when used for a Ms. Pac Man agent using the provided framework. A move can be made each time step (50ms) so the tree becomes big quite fast. If one takes into account the possible moves of all the ghosts at each time step the branching factor becomes overwhelming. 50ms is not enough time to rebuild a tree and explore it thoroughly enough to make good decisions, but reusing old trees is difficult as the state reached in the next time step can differ from the one in the tree. This is especially true if only Ms Pac-Mans moves are taken into account when branching. In that situation it is necessary to predict the moves of the ghosts which might not be accurate.

MCTS for Ms Pac-Man is implemented by taking into account only the moves made by Ms Pac-Man while the ghost moves are predicted. The built tree is maintained from the beginning but the current nodes state is always replaced with the actual state of the game. This will make the MCTS biased the longer it runs. When running the default policy the backup is started when Ms Pac-Man dies or when it has progressed (reached the depth) of 20 moves due to time limitation.

3 ALGORITHM PARAMETERS

3.1 Behavior Tree

One could define the parameters of a behaviour tree as the structure of the tree itself. However, the structure for the behavior tree used for the Ms Pac-Man agent is static. It is pre-defined and unchanged during run time. Though the tree was built using expert knowledge and insight about the game.

Something less static and more prone to change are the parameters used for some of the leafs of the tree. These are:

- Check if in danger; min distance.

The min distance parameter defines what is the minimum distance between Ms Pac-Man and the closest danger for the agent to be considered in danger and will start running away from that ghost. This parameter was changed and adjusted through experimentation. It has shown to perform better when the min distance was lower (around 5). When the distance was high, the agent started fleeing too soon which proved to be be a bad strategy as it took away from the other goal of the agent (collecting pills) and because the ghosts are often distributed around the map quite evenly which resulted in the agent fleeing from one ghost right into the arms of the other.

- Panic run; run duration.

The run duration parameter defines for how long the agent runs away from the ghost that made it run. This parameter was similarly optimised through experimentation. The optimal value was very short as the long run duration produced an agent which was too often oblivious to anything but the ghost. The leaf itself was also excluded from the tree as it produced worse results.

3.2 QLearning

The main parameters of the QLearning agent are the learning rate, discount factor, exploration chance, rewards and the minimum ghost distance.

"The learning rate determines at what extent newly acquired information overrides old information" [2]. It can change while learning but with the Ms Pac-Man agent it stayed constant. The learning rate is set at a constant 1.5, which was arrived to through experimentation.

"The discount factor determines the importance of future rewards" [2]. While a value which would be too low would make the agent too short sighted, a too high value might make it discard short term punishment for long term rewards. Though this can be mitigated through properly set rewards. It is set at a constant 0.9 to ensure the agent would chase ghosts when they are edible, even if they are far away.

The exploration chance defines with what probability the ghost will do a random action (exploration) or chose an action proven to be the best at that state (exploration). The exploration chance decreases linearly from 100 per cent at the begging of training to 0 per cent at the very end.

The rewards determine to what and how much the agent reacts to. The rewards given to the ghost were determined through expert knowledge but not adjusted at any time. The positive rewards are pill=1, magicPill=2, ghostEaten=3 and the negative reward is wasEaten=-10. The negative reward was set intentionally very high to prioritise ghost avoidance over pill collection.

Minimum ghost distance is a very crucial parameter in the QLearning agent. It determines how close the ghost must be to be considered part of the state representation. Since the states representation is an abstraction of the actual game state it is necessary to choose only the crucial information. The distance is set to a constant 30, which was arrived to through experimentation.

3.3 Monte Carlo Tree Search

The only adjustable parameter in the Monte Carlo Tree Search agent is the Cp constant. The Cp constant is the exploration term which defines how the tree policy balances exploration and exploitation. It is set to a constant 0.7, approximately 1 divided by the square root of 2. The reason for this value is because it ”was shown by Kocsis and Szepesvari to satisfy the Hoeffding inequality with rewards in the range [0, 1].” [4]

4 Performance measure and Discussion

The performance of the algorithms was measured when playing against the starter ghosts. Each type of agent was run 10 times, playing against the ghosts. The score was then averaged and these are the results:

Behavior Tree	2289
QLearning	232
Monte Carlo TS	147

None of the average results are particularly good. The reasons for these scores and how to possibly improve them are discussed individually.

4.1 Behavior Tree

The Behavior tree agent scored the most of all three agents. This is due to having much more direct control of the bahavior of the agent. All of the intelligence of the agent is outsourced to the designer. Ms Pac-Man is a game that seems simple, but is in its core quite complex, as far as decision making goes. Even though the Behavior Tree used as an agent is very simple, it preforms well because the actions are limited and pre defined.

It also became apparent that improving on this simple behavior is quite difficult. Some more complex leaves were developed and added to the tree but barely improved the average score or in fact made it worse. A leaf that made the agent chase the ghosts when they were edible actually put it in more danger, as it ended up closer to the ghosts at all time and made the ghosts respawn faster.

The agent is quite unsophisticated as it mostly reacts to it’s immediate surroundings and the current state. It does very little to try to predict the near or far future. This is most likely the best course for future improvement. Taking into account the consequences of it’s actions is also something that could be added to improve performance. As the agent only looks for it’s current move it does not take into account what this move entails.

4.2 QLearning

The QLearning agent unsurprisingly under preformed. This is because QLearning works quite poorly when the number of possible states is large. This was to be mitigated by the simplistic abstraction of the state representation. But this did not work well enough. This problem would likely be mitigated by running the training of the agent more times. As it currently stands, the agent is trained on 10.000 game attempts.

What likely inhibits its performance is the fact that it mostly learns from a set of states that occur in the first part of the game. The game is stopped when the agent gets eaten by a ghost and is then reset to the initial position. While the movement of the ghosts is affected by some randomness and the agent (especially at the beginning of training) prefers exploration over exploitation the states at the beginning of the game do seem overly represented in the training phase.

One thing that should be improved in the future is changing the function of transition from exploration to exploitation from a linear to a sigmoid function. The agent should also be trained for a much longer duration. But the main thing that would improve its performance is a better state representation.

4.3 Monte Carlo Tree Search

The Monte Carlo agent surprisingly under performed. Seeing the performance of the Monte Carlo agent built by T. Pepels [5] its performance was in fact quite underwhelming. This is most likely due to many reasons.

One obvious reason is the state representation. While the agent made by T.Pepels [5] is based on an abstraction of the true state, where the maze is represented by a graph of junctions and connection, the agent here works on a literal state representation.

Another reason is poor optimisation and design of the algorithm. While the algorithm could be running in a separate thread, building the tree and evaluating the possible moves it currently does not. The 50ms given for building and evaluating the tree allows the current implementation to run the main loop of the algorithm a mere 4 times on average. This does not produce the best or even good decision.

State evaluation (heuristic) could also be improved. Giving a state a very low score when the agent died and the actual score of the game when it didn’t is a very simplistic way of interpreting the game state. One could also measure the closeness of pills and the distance from ghosts.

When the default policy is run, it exits either when the agent has died or it reached a depth of 20. This seems to be an inadequate depth to properly evaluate the best next decision.

5 Conclusion

While the Behavior tree performed the best in the given circumstances, MCTS seems to have the highest potential if done correctly. Both MCTS and QLearning could be improved with better state representation, evaluation (heuristic) and plenty of fine tuning. In the available time, the direct control of the Behavior Tree seemed to overcome the complexity of the other two approaches.

References

[1] Georgios N. Yannakakis and Julian Togelius *Artificial Intelligence and Games* Springer, 2018

[2] Stuart Russell and Peter Norvig *Artificial Intelligence and Games, third edition* Pearson Education Limited, 2009

[3] Chris Simpson *Behavior trees for AI: How they work* Gamastura, 2014. Accessed: 2018 http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

[4] Cameron Browne et al. *A Survey of Monte Carlo Tree Search Methods* IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 4, NO. 1, MARCH 2012

[5] Tom Pepels, Mark H. M. Winands and Marc Lanctot *Real-Time Monte Carlo Tree Search in Ms Pac-Man* IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 6, NO. 3, SEPTEMBER 2014