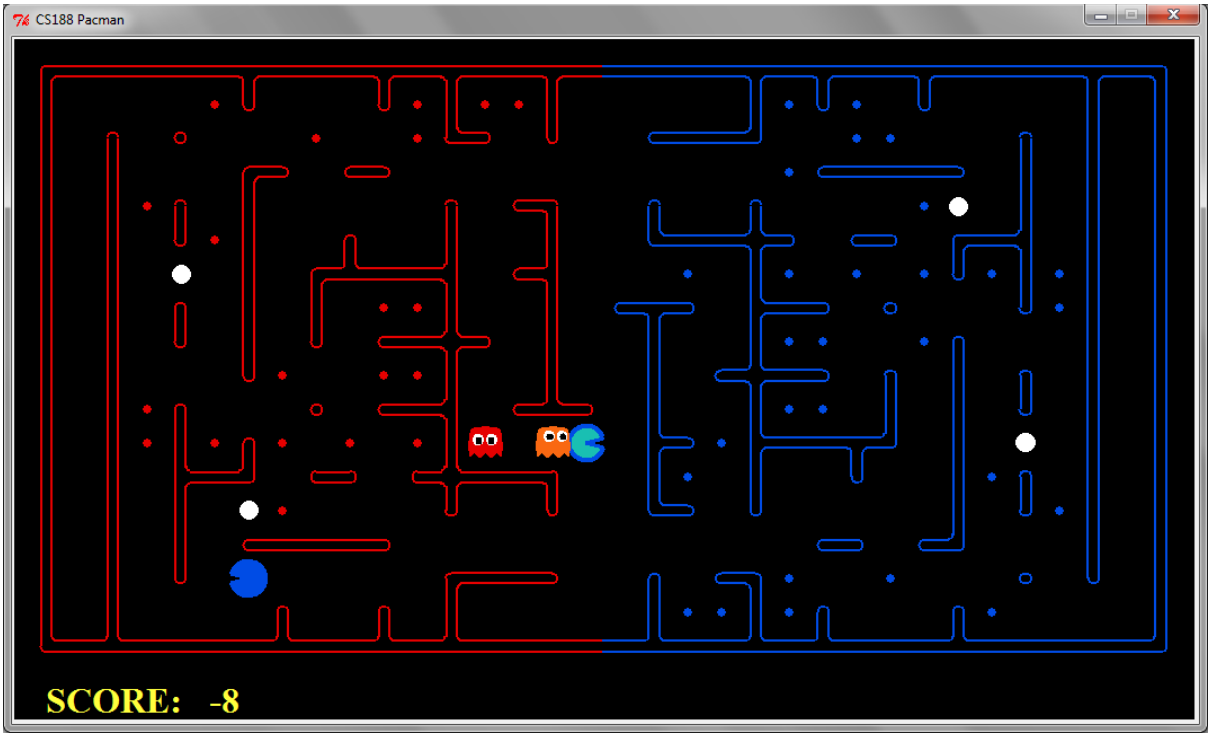
PACMAN : CAPTURE THE FLAG



Rohith R Vallu

Pankaj R Yadav

Nikhil Kumar Ramesh Jain

1. Introduction

This report details the different techniques used to control agents to play a multi-player variant of Pac-man. Pac-man : Capture The Flag (CS -188 Contest) was originally developed by John DeNero and Dan Klein as a part of Pac-man AI Projects at UC Berkeley. This version of Pac-man has completely different rules compared to the original Pac-man.

The game has been completely redesigned to implement the new rules of the game. The Pac-man map now consists of two halves: blue and red. Each half consists of its own food and the objective of each agent is to eat as much of its opponents food while at the same time preventing the opponent from eating its own food.

A multi-agent based environment gives us an opportunity to explore different AI techniques that can be implemented. Agents belonging to the same team need to coordinate their movements so that they can attack or defend while shifting between being ghosts or Pac-man. We have experimented with different AI techniques to implement the offensive/defensive tasks for each agent. This report will detail the techniques used, the benefits and the shortcomings of each technique and the results of how the agents fared during play-tests.

Along with several default maps, the framework also included a maze generation algorithm to generate new maps. This algorithm takes a seed as input and determines the complete layout of the map based on that seed (Togelius et.al 2011). Since a maze generator was already present in the framework, we contemplated how we could improve upon the algorithm to generate different maps than those being generated. We detail the changes made to the generator and the motivation for each change.

2. Approach

To design the agents, we needed to identify the different scenarios that an agent might encounter during the game. Agents can only observe an opponent's position and direction if they or their teammate is within five squares (Manhattan distance)(4). The maps are designed to be exactly symmetrical, so the opposing teams will usually encounter each other at the middle of the map. From here on the approach each agent takes differs based on the role assigned to each agent.

The agents described here follow a utility based approach to decision making. An agent will evaluate each option, calculate floating point values that describes how attractive the option is to the given agent's role and the best possible action will be executed.

2.1 Utility Agent

The first agent we designed was based on both defensive as well as offensive strategies. At the beginning of the game, defence is given priority. Agents that have entered its base will be chased. It effectively clears out the first wave of enemy agents before proceeding to go after food. An Utility agent almost always comes out on top during the first battle in the middle of the map.

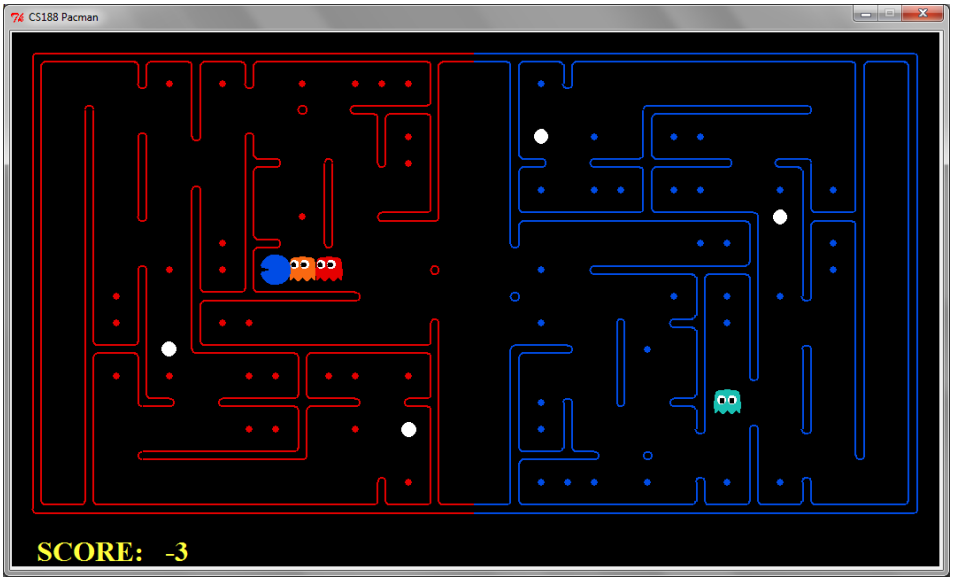


Fig 1: Utility Agents focus on getting rid of the first wave of enemies.
Source: Actual game-play.

Weights were assigned to each feature of this agent through iterative design process and rigorous testing. Higher priority was given to attacking Pac-man that were within its base. Even when it was actively eating the enemy food, the Utility Agent will come back to defend its own base when it detects an enemy ghost in its own half.

When paired with purely offensive agent (like the Offensive Reflex Agent) the Utility Agent does very well and can effectively increase the team's score by playing both defensive and offensive roles.

2.2 Adrenaline Agent

The Adrenaline Agent was designed to be slightly more offensive than the Utility agent. The difference here has to incorporate cooperation between two Adrenaline agents. Initially we tried including a feature 'distance-to-my-teammate', which would help spread the agents out in different directions while attacking the enemies base (thereby spreading out the attack and making it harder for the enemy to hunt the agents down). But this proved difficult to implement since one of the agents would often stay rooted to a spot (while trying to distance itself from its teammate) or go back to its starting position. We instead assigned specific sections to each agent at the beginning so they would split somewhere near the middle of the map to their respective sections.

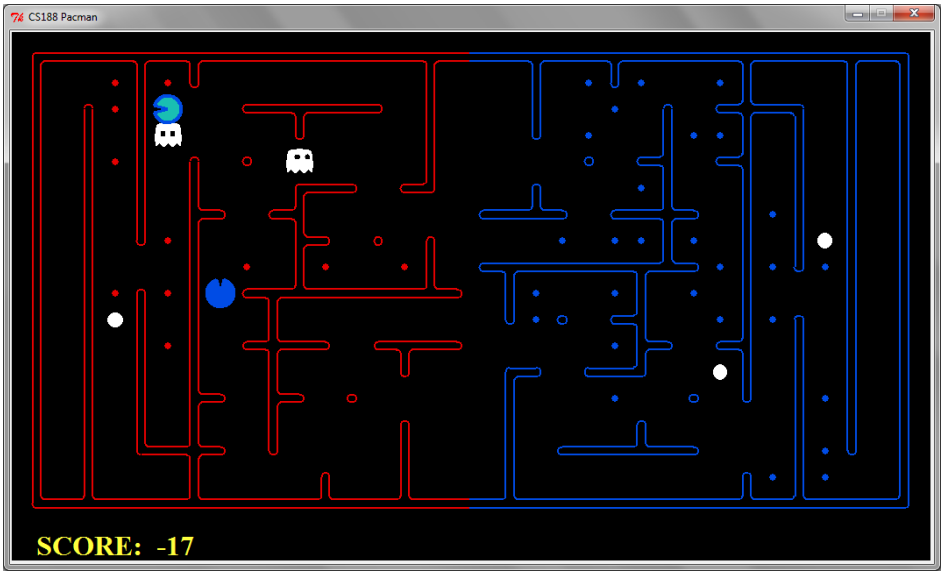


Fig 2: Adrenaline Agents have a more offensive role.
Source: Actual game-play.

The Adrenaline agent also acts differently when being chased by ghosts. This agent has an added feature for power pills- it actively seeks out power pills actively goes after scared ghosts. This feature has more weight when it detects that ghosts and the power pill are both nearby, therefore making it decidedly more dangerous than the Utility agent. During our play-tests we found that most player found an opposition of two Adrenaline agents very hard to beat. Against two defensive agents (like the Defensive reflex agents that were provided with the framework) the Adrenaline agents won every single time.

We tried to assign weights for both agents using Q-learning but it proved ineffective. Firstly, it didn't seem likely that the agents would learn cooperative behaviour through Q- learning. Even without the cooperative behaviour, the agents didn't seem to be making progress with large number of iterations(even with an increased learning rate) and we had to abandon that approach. Instead we relied on our intuition to come up with weights that would be appropriate for each feature and relied on testing to come up with the right balance for each the behaviour of each agent.

2.3 PCG

The framework included many default maps to test the agents. Moreover, it also included a maze generator algorithm that took random seeds as input to generate various maps. This by itself gave rise to a lot of variety to the levels. We changed the algorithm to create mazes which have caves or rooms with single openings, and more gaps in the middle. The caves were designed to create scenarios where the ghosts would have an opportunity to trap Pac-man. It also forces the human players to prioritize which food they should go after based on the position of enemy ghosts and the power pills. Gaps in the middle and allow multiple outlets for both ghosts and Pac-man into each other's maze. We have made changes to the framework so that every time the game runs, the maze generator algorithm is called.

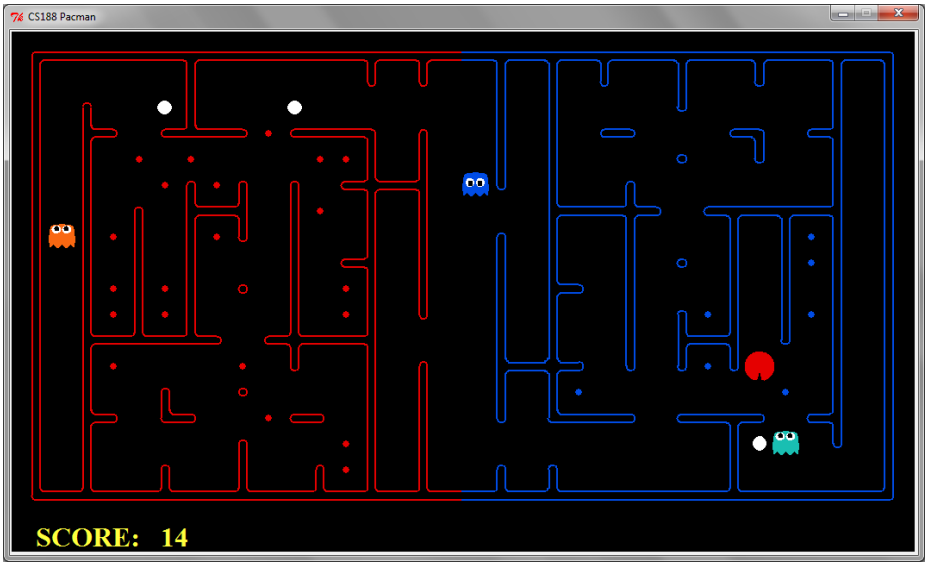
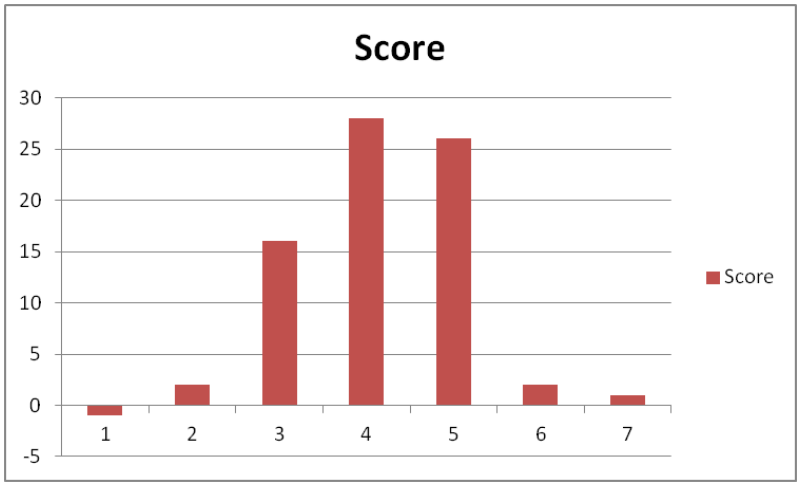


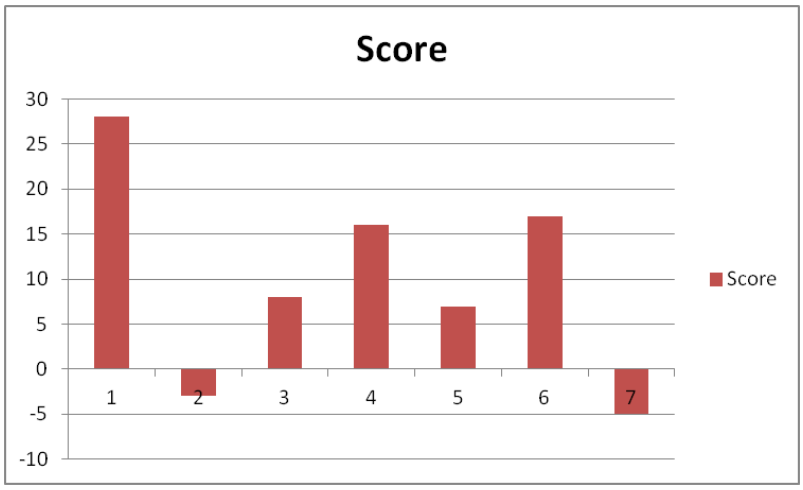
Fig 3: Mazes are designed to have deeper rooms/caves with multiple gaps in the middle.
Source: Actual game-play.

3. Play-Test Reports

The play-tests between AI agents showed that the Utility and the Adrenaline agents were more than a match for the Baseline agents provided by the framework. The Baseline's combination of an Offensive Reflex agent and a Defensive Reflex agent provide quite a combination for even human players to play against. But the combination of Utility and Adrenaline were more than a match against them. The following graphs show the performance of these agents over seven trials.

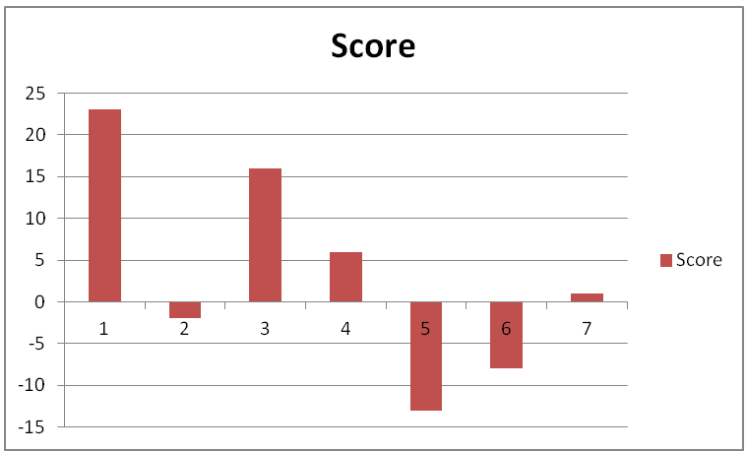


Adrenaline & Adrenaline(RED) Vs Reflexive Defensive & Reflexive Offensive(BLUE)
(Positive Score indicates RED team wins)



Adrenaline & Utility(RED) Vs Reflexive Defensive & Reflexive Offensive(BLUE)
(Positive Score indicates RED team wins)

Against each other, a pair of Adrenaline agents didn't really have a big advantage over a pair of Utility agents. Both performed equally well against each other, over seven different maps, where sometimes the layout of the map slightly favoured the Utility agents role.



Adrenaline & Adrenaline(RED) Vs Utility & Utility(BLUE)
(Positive Score indicates RED team wins)

Human play-tests proved that both the agents were quite difficult to perform against. A lot of our testers enjoyed the game since it's a very addictive version of Pac-man and the agents, although difficult to play against provided the right balance so that with practise it was possible to overcome the difficulty level. The generation of new levels also provided a different challenge to the players every time they played again.

During our play-tests we kept our frame-rate constant (at 25 fps). We've included a few lines of code which enables players to change the frame-rate in capture.py.

4. Discussion- Open Problems

4.1 Belief Distribution using Probabilistic Inference

The framework included a way to get noisy distances for teammates and opponents that were more than five squares away. By implementing probabilistic inference for opponents to determine their probable locations when they are far away, it would be possible to make more sophisticated and intelligent agents. Given more time, we would have tried to implement such a system. Furthermore, an adversarial search (like min-max) would be more easily implementable when the approximate location of the opponent is available.

4.2 Decision Making System Based on Time/Food Availability

It makes more sense for an agent to switch from offense to defence or vice-versa if it's able to determine that the opponents team is going to win, since the amount of food it has to eat far exceeds that of its opponents. Similarly, decisions based on time availability would have been a useful strategy and something we would have experimented with if we had more time.

4.3 Conclusion

Using a utility based system we were able to successfully implement two types of agents that performed well against human players. A combination of two Utility or Adrenaline or one of each proved to be quite a challenge to a lot of human players. But despite being slightly more offensive, two Adrenaline agents don't always beat two Utility agents. The variety of levels provided by the maze generator also created new challenge for the players every time they played the game. Our overall objective was to design agents which would be competitive to play against. A utility based AI system enabled us to design agents which would choose from several different options and implement the best decision.

5. References

CS-188, Contest: Capture the Flag, Introduction to Artificial Intelligence http://inst.eecs.berkeley.edu/~cs188/sp11/projects/contest/contest.html accessed December 2013

John DeNero and Dan Klein, Teaching Introductory Artificial Intelligence with Pac-Man http://www.denero.org/content/pubs/eaai10_denero_pacman.pdf accessed December 2013

Julian Togelius, Georgios N. Yannakakis, Kenneth O. Stanley, and Cameron Browne, Search-Based Procedural Content Generation: A Taxonomy and Survey http://www.ccs.neu.edu/course/cs5150f13/readings/togelius_sbpcg.pdf accessed December 2013

Kevin Dill Eugene Ray Pursel Pat Garrity, Gino Fragomeni, Design Patterns for the Configuration of Utility-Based AI

http://www.ccs.neu.edu/course/cs5150f13/readings/dill_designpatterns.pdf accessed December 2013