GCU Glasgow Caledonian University

University for the Common Good

Ensemble decision making in real-time games

Rodgers, Philip; Levine, John; Anderson, Damien

Published in:
2018 IEEE Conference on Computational Intelligence and Games (CIG)

DOI:
10.1109/CIG.2018.8490401

Publication date:
2018

Document Version
Peer reviewed version

Link to publication in ResearchOnline

Citation for published version (Harvard):
Rodgers, P, Levine, J & Anderson, D 2018, Ensemble decision making in real-time games. in *2018 IEEE Conference on Computational Intelligence and Games (CIG).* IEEE, 2018 IEEE Conference on Computational Intelligence and Games (CIG), Maastricht, Netherlands, 14/08/18. https://doi.org/10.1109/CIG.2018.8490401

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please view our takedown policy at https://edshare.gcu.ac.uk/id/eprint/5179 for details of how to contact us.

Download date: 17. Apr. 2021

Ensemble Decision Making in Real-Time Games

Philip Rodgers
Computer and Information Science
University of Strathclyde
Glasgow, United Kingdom
philip.rodgers@strath.ac.uk

John Levine
Computer and Information Science
University of Strathclyde
Glasgow, United Kingdom
john.levine@strath.ac.uk

Damien Anderson
Computer and Information Science
University of Strathclyde
Glasgow, United Kingdom
damien.anderson@strath.ac.uk

***Abstract*—This paper describes an Ensemble Agent for the classic arcade game Ms. Pac-Man. Our approach decomposes the problem into sub-goals. An expert agent is created for each sub-goal, with all experts reporting to a central arbiter. Our Ensemble Agent has achieved the AI world record for the arcade version of Ms. Pac-Man with a score of 162,280. For comparison, a MCTS-based monolithic agent was also created, based on the same accurate forward model that the Ensemble Agent uses, reaching a score of 115,180.**

***Index Terms*—ensemble, mcts, pac-man, real-time, decision**

I. Introduction

A. Ensemble Systems

Ensemble systems have been used for classification problems since the late 1970s [1]. They made use of feature partitioning to create multiple classifiers. These early classification systems used ensembles for a number of reasons.

- If there is too much data, divide and conquer techniques can be applied to split the data into more manageable subsets. Each subset can be run through a separate instance of the classifier.
- Too little data can be used more efficiently by creating multiple, overlapping training sets, with each set being used to train a separate classifier.
- A level of redundancy can be achieved by creating multiple classification systems, each trained on a unique or overlapping subset of the training data. The chances of misclassification can be lower for an ensemble classification system than for a monolithic system.
- Ensemble systems can easily handle heterogeneous data. A separate classifier can be built for each data source, and the ensemble system can combine the results for an overall classification.

A modern example of a powerful ensemble system is IBM's *Watson* [2]. Watson was originally created to play the TV quiz show *Jeopardy*, but has since been opened up for general use. It uses natural language recognition to analyse questions and generate queries. It sends the queries to multiple sources of answers, known as *many experts*, and combines the answers, calculating confidence levels for each of the answers. Watson has shown great potential in helping doctors with patient diagnoses [3].

The Ensemble decision system described in this paper uses concepts from ensemble classification systems, namely feature partitioning and expert systems, and applies them to real-time decision making. The Ensemble decision systems can be used to build complex agent behaviors out of simple components or *voices*. Each voice can be considered an agent in its own right, but with a simple, single goal or task. At the heart of the system is an *arbiter* which takes the outputs, or *opinions*, of the voices and generates the final decision, as shown in Figure 1.

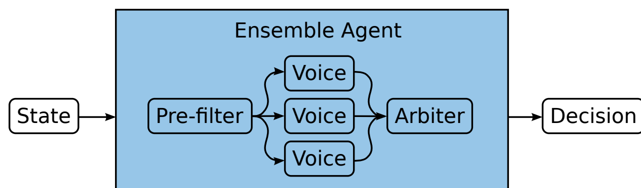

Fig. 1: Ensemble Decision System

The concept sounds similar to that of a *subsumption architecture*, but it differs significantly in that the decision making is not being deferred from one component to the next. Instead, all of the voices have an opinion all of the time, with each voice contributing to the net result. Ensemble decision systems far more akin to ensemble classification systems than subsumption architectures.

Ensemble decision systems are inherently flexible. They can have voices added or removed without affecting the other voices or the arbiter, only the final decision. Greater efficiency can also be achieved through the separation of behaviours that can be handled either reactively or deliberatively. Ideally all voices would be reactive, but some behaviours, such as ghost avoidance in Ms. Pac-Man, require some deliberation in order to achieve good results. These deliberative voices can be kept as simple as possible by disregarding anything unrelated to its given task. Pre-filtering is an option step that removes any moves known to be invalid or determined to be bad from previous iterations.

In its simplest form, the Ensemble Decision System was envisioned to have three primary component classes, short-, middle- and long-range goals. These can be seen as survival, tactics and strategy. This idea is by no means a requirement, and the Ms. Pac-Man ensemble agent described in this paper does not strictly adhere to this structure.

B. Pac-Man

The original Pac-Man is an arcade machine from 1980, created by game designer Toru Iwatani of Namco and released in the United States by Midway Manufacturing Corporation. Pac-Man was a massive success, being the second highest grossing arcade game after emphSpace Invaders.

In Pac-Man, the player controls the main character around a maze using a four-way joystick. The aim of the game is to complete each level by eating all the pills dotted around the maze whist avoiding the four antagonistic ghosts. Each level also has four *power-pills* that allow Pac-Man to become *energised* for a short period of time, during which the ghosts can be eaten. Eating a power-pill also has the effect of making the ghosts reverse direction. In later levels the time that the ghosts are edible drops to zero. Two bonus items, often referred to as *fruit*, also appear per level.

Points are scored by eating normal pills (10 points each), power pills (50 points each), ghosts (200, 400, 800 and 1600 points if eaten in succession) and bonuses (100, 300, 500, 700, 1000, 2000, 3000 or 5000, depending on the bonus). From level 13 onwards, the bonus is always worth 5000 points.

C. Ms. Pac-Man

Ms. Pac-Man is a sequel to Pac-Man. It has essentially the same game mechanics but with several enhancements over the original game. There are four mazes, as opposed to the single maze of Pac-Man. The bonuses move around the maze, rather than appearing stationary in the centre of the maze, and in the later levels the bonus item is chosen pseudo-randomly.

The main difference, and what makes Ms. Pac-Man more appealing to players and AI developers alike, is the non-deterministic behaviour of the ghosts. The original Pac-Man game is entirely deterministic, so players can learn patterns to complete each level. The game can be beaten over and over by the simple repetition of the correct pattern. Ms. Pac-Man introduced enough random behaviour in the ghosts to allow for strategies but not patterns.

Ms. Pac-Man was chosen for this project as it is well known and there is a lot of prior AI research for this game [4]. Most of the prior work has been done using either the screen-capture Ms. Pac-Man competition framework[1], or the Ms. Pac-Man vs. Ghost-Team framework[2]. This project uses our own emulator, written in Java and capable of playing the original Ms. Pac-Man code. The emulator is described in section IV.

The original game has more complexity than the Ms. Pac-Man vs. Ghost-Team framework, and while the screen-capture framework is true to the original game, it has its own usability issues. The use of a emulator written in Java gives the simplicity of the Ms. Pac-Man vs. Ghost-Team framework and the authenticity of the screen-capture framework.

Ms. Pac-Man is a good benchmark of an AI system as it combines simplicity and difficulty. There are at most four possible options to choose from for any given state, and the search space is confined to a single-screen maze. Despite this simplicity, Ms. Pac-Man remains a hard problem for AI agents and humans alike. This is primarily because of the enclosed nature of the game space and the four-to-one ghost ratio. Simply trying to keep a certain distance from the ghosts is likely to end up with Ms. Pac-Man being trapped. Understanding how the ghosts will react to in particular situation, and so avoiding being trapped, is the key to survival in Ms. Pac-Man.

D. High Scores

The highest published score for an AI playing the original Ms. Pac-Man is 44,630 [5], using the screen-capture framework. This score was the maximum of 100 games, with level six being the highest reached. This would be considered a good score for a human. The best human players can reach scores in excess of 900,000, clearing more than 130 levels[3]. The highest score recorded by the Ensemble agent is currently 162,280 at level 24. This result was achieved while recording a video and is not part of the experimental data. The video is available on YouTube[4].

Ms. Pac-Man was recently released on Steam. The leader board would suggest 30,000 to be a reasonable average score for a human. In discussion with Patrick Scott Patterson—a video game advocate, journalist and record holder—Mr. Patterson suggested that six-figure scores were rare, and that only a handful of players in the world are capable of playing the game at this level. The current official world record is 933,580, set by Abdner Ashman in 2006. Only five people have officially reached over 900,000 points.

In their paper *Hybrid reward architecture for reinforcement learning* [6], van Seijen *et al.* describe an agent for the Atari 2600 version of Ms. Pac-Man. The agent they describe is very similar to the Ensemble agent described in this paper, but they use many more voices and the voices are learned rather than hand-coded. Their agent is effectively unbeatable, but the Atari 2600 version of Ms. Pac-Man is a lot simpler than the arcade version.

II. Monte-Carlo Tree Search in Ms. Pac-Man

MCTS has been applied to Ms. Pac-Man before. In their paper *Monte-Carlo Tree Search In Ms. Pac-Man* [7], Ikehata and Ito describe a MCTS agent for the screen-capture framework Ms. Pac-Man. For the MCTS simulations, Ikehata and Ito used a simplified model of the game. Despite using this simplified model, the agent entered and won the 2011 IEEE Ms. Pac-Man competition.

In 2014, Pepels et al published their paper *Real-Time Monte Carlo Tree Search* [8]. The paper describes an MCTS agent for the Ms. Pac-Man vs. Ghost Team framework. In this framework, agents have only 40ms to choose a move, so the team use depth-limited roll-outs and branch re-use to achieve competitive play.

[1] http://csee.essex.ac.uk/staff/sml/pacman/PacManContest.html
[2] http://www.pacmanvghosts.co.uk/
[3] https://www.twingalaxies.com/game/ms-pac-man/arcade
[4] https://youtu.be/Y9YazqWaEAM

III. Ms. Pac-Man vs. Ghost-Team

Initial experiments for this project were done using the Ms. Pac-Man vs. Ghost-Team framework. This led to some useful insights and a very capable agent—usually reaching the global time-limit at around level 12, often without losing a life. The agent played especially well against highly predictable ghosts, such as the aggressive ghost team, where it could group the ghosts together, eat a power-pill and then eat all the ghosts in quick succession.

To find out how the agent would fare against a top-ranking ghost team, we contacted the author of the *Memetix* ghost team, Daryl Tose, who very graciously sent us his code. As it turned out, despite the *Memetix* ghost team being almost completely deterministic, and the Ensemble agent being very capable against predictable ghosts, the agent rarely got past the first level. The conclusion we made was that however strong the Ms. Pac-Man agent is, the ghosts can always win if they work together as a team.

IV. JAMES

JAMES was written from the ground up to be an object-oriented Ms. Pac-Man emulator. Large sections of the code came from the ArcadeFlex project[5][6], an automated Java port of the Multi Arcade Machine emulator (MAME)[7]. The code was constructed as a core emulator, with a full emulator built around it. The core emulator emulates the CPU, RAM and I/O. The full emulator adds windowing, graphics and keyboard support. This allows the core emulator to be used as a forward model not tied to the 60 frames per second of the full emulator.

Agents interact with the game via the Game API. Game state information is obtained by interpreting the contents of specific memory locations within the emulator. Actions are performed by setting the values of the memory-mapped I/O ports. The Game class abstracts away from these low-level operations.

A graph data structure was created for maze-based queries. All-pairs tile distances were pre-calculated using the Floyd-Warshall algorithm [9]. In addition to the all-pairs distances, every tile stores the distance to every other tile for each available move. These directional distances were also pre-calculated, this time using A* search with the Floyd-Warshall computed distance as the heuristic.

Using the emulated Ms. Pac-Man code as a forward model is extremely accurate, but very inefficient. An alternative forward model, the *simulator*, was created.

V. The simulator

The simulator is a Java-native partial model of the game. A lot of work went into making the simulator as accurate as possible, especially with regard to ghost behaviour. Although the simulator it is not 100% accurate, it is generally accurate enough if synchronised with the emulator before each use.

[5]https://www.facebook.com/arcadeflex/
[6]https://github.com/georgemoralis/arcadeflex029
[7]http://mamedev.org/

Fig. 2: Opposing opinions

The simulator is very fast compared to the emulator, more than making up for the loss of accuracy.

A simple test was conducted to gauge the relative speeds of the emulator and simulator forward models. In this test, Ms. Pac-Man travels from the bottom-right corner of maze one to the bottom-left corner. This is a straight path, with pills, that takes 198 frames. In real-time, at 60 frames per second, that is just over three seconds. Averaged over 1000 runs, the emulator took 23.1ms; 137 times faster than real-time. The simulator took just 0.2ms; 16,417 times faster than real-time and 119 times faster than the emulator forward model.

VI. Ensemble Agent for Ms. Pac-Man

For the Ms. Pac-Man agent, the tasks were defined as:

- **Eat pills.** This is primarily a long range goal to clear the level.
- **Eat fruit.** This is a medium range goal to collect extra points.
- **Eat ghosts.** This is another medium range goal to collect extra points.
- **Avoid ghosts.** This is a short or medium range goal, depending on whether Ms. Pac-Man is escaping from a close-range ghost or avoiding being trapped.

It became apparent during initial experiments that simply having each voice offering its preferred move at any given point lead to a lot of deadlocks. The voices would often have opposing opinions due to the completely disparate nature of their goals. In Figure 2, Ms. Pac-Man is approaching a junction with three options: UP, LEFT or DOWN. The pill eating voice will vote to go DOWN, the ghost eating voice will vote to go LEFT and the fruit eating voice will vote to go UP. No reward is worth dying for, so the ghost avoiding voice will veto DOWN, leaving a deadlock between UP and LEFT. The move could be picked at random, or the arbiter could be crafted with some domain knowledge to make a more informed decision.

The final Ensemble Agent uses a technique similar to fuzzy logic, where each voice *rates* each of the available moves according to its own metric.

Using the same scenario as in Figure 2, with DOWN vetoed by the ghost avoiding voice, the other three voices need to present their ratings for UP and LEFT. As the voices are all distance based, the ratings will be the inverse of the distance to the goal in each available direction. If the voices are weighted equally, the resulting move would be UP. An approximation of the calculation can be seen in Table I.

	Pill eater	Fruit eater	Ghost eater	Sum (approx.)
UP	1/7	1/3	1/24	1/2
LEFT	1/7	1/24	1/4	2/5

TABLE I: Calculating move values

This solution is far less likely to lead to a tie-break situation, and it is also more flexible in terms of weighting each voice's opinion. In the above example the agent decided to go UP because the fruit is closer than the edible ghost, but it was close. Generally, there is likely to be more chance of eating the fruit in the future than the ghost, so LEFT would probably have been a better choice. Weighting the ghost eater higher than the fruit eater would have changed the decision to LEFT. Rating the pill eater low, because pills are relatively low value and static, would likely make the agent head towards the fruit after eating the ghost. The arbiter never actually targets anything, or makes any sort of plan. It simply chooses the highest combined-rated move at any given point.

The final Ensemble Agent for Ms. Pac-Man in *JAMES* is composed of four voices, with an arbiter taking the opinions of each voice and combining them to make the final decision.

- **Ghost Dodger.** Avoiding ghosts is the most important aspect of the game, and is also the hardest to do, computationally. It is the average result of sparse sampled, depth-limited roll-outs. This voice is discussed in detail in the next section.
- **Pill Muncher.** This voice rates each move as the inverse of the tile distance to the nearest pill in that direction. Pills near ghosts are artificially made to look further away, meaning that the pill muncher will rate safe pills higher than those with ghosts near by.
- **Fruit Muncher.** This voice has no opinion unless there is a fruit bonus on the screen. If a fruit is on the screen, the voice will attempt to intercept it. It rates the available moves as the inverse of the tile distance to the fruit.
- **Ghost Muncher.** This voice only has an opinion if Ms. Pac-Man is energised. This voice uses a similar sparse sampling technique as the Ghost Dodger voice, but it could also simply rate each move based on the distance to the nearest edible ghost. The sparse sampling technique allows for elegant behaviour such as intercepting ghosts, rather than simply chasing them.

A. Ghost Dodging

The final Ghost Dodger algorithm uses the simulator for depth limited search, and rates each move based on sparse random sampling. This algorithm is closely related to the averaged depth-limited search technique we applied to the game 2048, as demonstrated at the IEEE CIG2014 conference in Dortmund [10].

The algorithm is given 10ms in which to make random depth-limited, Monte-Carlo style samples through the maze. Every time it reaches its depth limit of 8 without dying, the initial move's score gets incremented. At the end of the 10ms, or it has found enough safe paths to be sure a move is safe, the voice returns its rating for each move, based on how many times it reached the depth limit. The depth of 8 was chosen as a trade-off between depth and the number of samples possible in the time-frame. For this algorithm, a *move* is a straight line from the current position to the next corner or junction. This simplifies the algorithm as Ms. Pac-Man's direction does not need to be re-calculated mid-move for cornering.

Because the simulator is not 100% pixel-perfect, it may determine a path to be safe when in the real game it is not. This usually occurs if there is a ghost very close. This becomes a real problem when deciding whether or not to clear a path of pills. Ms. Pac-Man pauses for a single frame when she eats a pill but the ghosts do not, so a following ghost will be faster. This quite often leads to situations where the simulator determines a path to be safe to traverse, only to realise its mistake when it is too late.

B. Arbitration

As previously discussed, the ensemble voices offer ratings for each available move. It is the job of the arbiter to combine the ratings into a decision. Voices associated with risk, such as the Ghost Dodger, are treated as multiplicative. Voices associated with reward, such as the Fruit Muncher, are treated as a sum. The overall value of a move is the sum of its rewards multiplied by the risk factor of that move.

In general, the first k voices of the total n voices are multiplicative, with the remainder being a summation. The value of each move m for each voice V_i is multiplied by weight W_i. The product of all the risk-based voices is calculated to give us a relative measure of risk. In this case, 0 indicates certain death and 1 indicates no risk.

$$RISK_m = \prod_{i=1}^{i=k} V_{i,m} W_i \tag{1}$$

We also need to calculate the reward associated with each move. This is done by summing all the move values for each of the reward voices:

$$REWARD_m = \sum_{i=k+1}^{i=n} V_{i,m} W_i \tag{2}$$

To calculate the final vector of move ratings R, we multiply these two factors. This ensures that any reward, no matter how large, will be nullified if the risk is too great.

$$R_m = RISK_m \times REWARD_m \quad (3)$$

The arbiter simply chooses the move corresponding to the highest valued R_m. If more than one move have the highest rating, the arbiter will select at random from the highest rated moves.

In this project the weights were hand-crafted and tuned until the behaviour was deemed good enough. These weights are unlikely to be optimal, so weight optimisation is a potential area for improvement.

1) Evolving Weights: Instead of hand-crafting the weights, as a proof of concept, a simple one-plus-one evolutionary strategy was used to optimise the weights of the voices. A baseline score was recorded over 100 games, then the weights were adjusted by a small random amount. Another 100 games were played with the new weights. If the new weight were kept if the average score improved. This was done 1000 times, for a total of 100,000 games. An early version of the Ensemble AI was used for this experiment, and it was able to increase the ability of that player. In this experiment, a noticeable increase in average score of approximately 30% was achieved. Unfortunately this technique takes a very long time, and it was not used in any of the final agents. In their paper *The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation* [11], Lucas et al describe an algorithm for optimising N-Tuple values for noisy and expensive problems. This approach could achieve better results quicker than the simple one-plus-one approach.

2) Dynamic Weights: Dynamic weighting is also a possibility, combining the Ensemble arbiter with a finite-state machine. Using such a technique would allow the agent to adjust the voice weights depending on the situation. This could be of use in, for example, stealth games. One set of weights could be used during stealthy missions, or portions of missions, while another set of weights used in situations where the player's stealth has been compromised. Dynamic weighting could also be used in general video game playing (GVGP) [12], especially if combined with general-purpose components.

VII. Monte-Carlo Tree Search

The MCTS agent was developed to set a high bar, and to demonstrate what a purely deliberative agent was capable of. Despite using the simulator forward model, and hence no knowledge of fruit, the MCTS agent manages to play to a very high standard.

A. Deliberation and Double Checking

In order for a our MCTS agent to perform well, it needs time to deliberate. In real-time games this is tricky. If the MCTS agent was trying to play synchronously with the game, a decision would need to be made every 16ms. Our simple implementation of MCTS was incapable of even playing the game, let alone play it well.

In order to solve the problem, we allow the MCTS agent to utilise the time it takes to get to its next target to run the simulations of what will happen when it gets there.

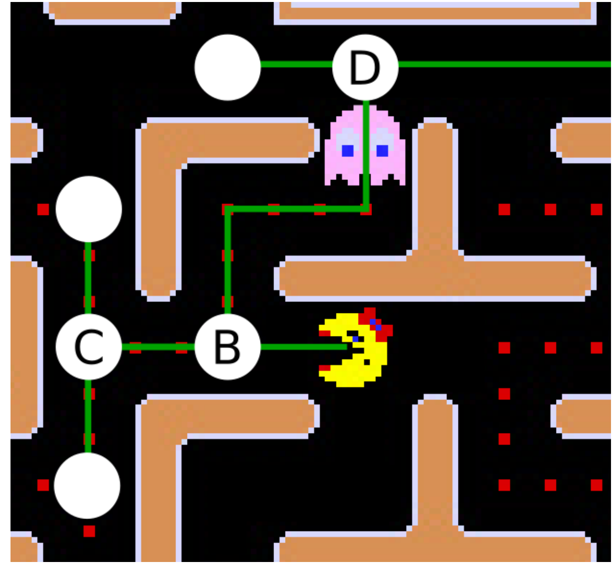

Fig. 3: MCTS paths

In figure 3, the agent is at point A and has committed to traveling to point B. Path AB has already been simulated and checked using the emulator forward model, so the agent can be almost certain that it is safe. During the time it takes to reach point B, the MCTS algorithm is running simulations from point B onwards.

The instant Ms. Pac-Man enters the tile associated with point B, the MCTS algorithm is stopped and the best move from point B is selected. In figure 3, BD always leads to death, so BC is selected. The agent will then double check BC using the emulator and commit to that move. The MCTS algorithm will start simulating moves from point C onward. If something unexpected happened and the emulator check shows a problem, the agent will cancel that move and go back; in this case, towards point A. The MCTS algorithm will be restarted from point A.

Using this travel time for deliberation allows the MCTS algorithm to visit each move hundreds, if not thousands of times.

B. Selection Phase

Preliminary experiments with exploration verses exploitation showed that with high exploration, and hence highly symmetric trees, the agent did not perform so well as with high exploitation. The symmetric and asymmetric agents both scored roughly the same on average, but the asymmetric agent's average level reached was 15, as opposed to the 10 of the symmetric agent[8].

From watching two versions of the MCTS agent playing, it could be seen that the symmetric agent starts off well, and does a great job of eating the edible ghosts, but when the level is almost complete—especially if there are two small, separate clusters of pills, the agent doesn't seem to know what to do.

[8]These results were obtained using an early version of the agent. The final MCTS agent plays much better, but the findings on symmetry still hold true.

There is very little reward available but lots of ghost death to be avoided. These situations lead to the agent getting stuck in a local optimal loop, going back and forth avoiding ghosts, but not getting the level complete. If the agent gets pushed by the ghosts close enough to one of the clusters of pills, it will take them. As soon as only one cluster of pills remains, the agent can see the win and will go for it.

In contrast to this, the asymmetric MCTS agent does not seem to do as good a job of eating the ghosts, and so scores fewer points, but it doesn't get stuck in local optima as often or for as long. The asymmetric nature of the trees means the agent is more likely to be able to see a winning path through two distant clusters of pills.

The average reward of a given move is used in the selection phase in place of the number of wins. As this reward is based on the score delta of the move, the exploitation value tends to be very high. To counter this, and create a better balance of exploration verses exploitation, am exploitation factor of 2000 is used.

C. Simulation Phase

Pure MCTS makes random, legal moves until an end-state is reached. While this approach works well for zero-sum games, for an arcade game like Ms. Pac-Man the only true end-state is GAME OVER. We could use intermediate end-states, such as when the agent is killed or the level is completed, but losing a life is also bad, and finishing a level is unlikely to happen making random moves.

Full roll-outs are difficult to assess, being as the vast majority of them will be death. Even if we use an evaluation function with a partial roll-out, the roll-out is likely to end in death. Another issue with evaluation functions is that they will heavily bias the roll-outs. If, for example, the evaluation function rewards distance from ghosts, the MCTS algorithm will avoid ghosts and end up being trapped just like a distance rule-based ghost avoidance algorithm.

The final, and best performing, MCTS agent has no roll-out phase at all. Once a new leaf is generated in the expansion phase, the current score delta is returned for back-propagation. The only cases where the score is not simply returned is when either the agent is killed or the level is complete. In the case of a death, the reward (score) is divided by 10. This has the effect of guiding the selection away from that part of the path, but without poisoning the entire branch, as was the case when the reward was set to zero on death. In the case of level completion, an extra 1000 points are added to the reward. That sounds low, but it is enough to get the agent going in the right direction without taking unnecessary risks to get the win.

D. No Emulator Checking

For the experiments where the agents cannot access the emulator, the MCTS agent did not do so well. Because it is always thinking one decision point ahead, it relies on the emulator to ensure the current path is safe. Removing this accurate checking mechanism proved to be a problem for the MCTS agent. The less than 100% pixel-perfect simulator can sometimes regard a path as safe when it is not; when a ghost is following Ms. Pac-Man very closely, for example. Just a single pixel discrepancy between the simulator and the actual game can result in completely different decisions from the ghosts, especially at close-quarters.

In order to have the MCTS agent able to perform reasonably well without the emulator, it needed to be able to react to its mistakes. This was achieved at the expense deliberation time by changing the algorithm to return a decision at each new tile, rather than at each corner or junction. The actual MCTS algorithm still used the corners and junctions as nodes when expanding the tree, but the tree gets rebuilt each time the agent enters a different tile. This allows the agent to effectively change its mind half way along a corridor, if a danger becomes apparent.

VIII. Experiments

The experiments themselves were very simple. Each agent plays 100 games at normal speed. The results of these experiments are then compared on both score and level reached. The level reached being used as a measure of the survivability of the agent. All experiments were run on a single desktop computer.

- **Machine**: 2011 Dell OptiPlex 790
- **CPU**: Intel Core i7-2600 running at stock 3.4GHz
- **RAM**: 16GB DDR3 running at 1333MHz
- **OS**: Linux Mint 17
- **JVM**: Oracle Java 7

Variations of the two main agents, MCTS and Ensemble, were used for comparison and fairness. Along with the 'best' versions of each agent, experiments were run with and without access to the emulator forward model. The MCTS agent makes far greater use of the emulator, so it is to be expected that this agent suffers more for having it removed. Hopefully the experiments show that the emulator is very useful, not a necessity.

Experiments were also run where the Ensemble does not include the fruit munching component. The fruit bonuses are not included in the simulator, so the MCTS agent has no knowledge of the fruit. This gives the Ensemble agent an unfair advantage in terms of point scoring. Turning off the fruit component removes this advantage.

IX. Results

Both agents use the emulator as a short-range, accurate forward model. The MCTS uses the emulator to double-check before it commits to a path, and the Ensemble uses the emulator to pre-filter the available moves to remove certain-death moves. The Ensemble also uses it as double-check for immediate death on the current move.

The MCTS agent reached level 13 in all 100 game played and reached level 21 in 68 of the 100 games. In terms of survivability, the MCTS agent is very strong. The main weakness of the MCTS agent is its relatively poor points

scoring. This is primarily due to the lack of knowledge of the fruit bonuses.

Because the MCTS agent does not have any knowledge of the fruit, a version of the Ensemble agent without the fruit muncher voice was also tested. In this configuration, the MCTS agent and the Ensemble agent score almost identical average scores.

The MCTS agent has greater survivability than the Ensemble, with an average level reached of 20.38 to the Ensemble's 18.67. That the Ensemble agent manages to score equally well in fewer levels than the MCTS agent, even without fruit, suggests that the Ensemble agent is slightly better at capturing ghosts than the MCTS agent.

Adding the fruit component to the Ensemble resulted in a 26% increase in maximum score, with a 33% increase in average score. This increase in score did come with a very small drop in survivability, with the average level reached dropping from 18.67 to 18.37, less than 2%.

A. *Results With emulator*

As can be seen in Table II, when compared to the MCTS agent the Ensemble agent scores significantly better. Most of the extra points seem to come from the MCTS agent's inability to effectively capture fruit. When the Ensemble agent has the fruit muncher voice disabled, there is no significant difference in scoring compared to the MCTS agent.

Agent	Minimum	Maximum	Mean	Std. Err.	p-value
Ensemble	44860	153280	118610	2477	$< .0001$
MCTS	57920	115180	89278	1286	
Ensemble no fruit	36310	121360	89095	1547	.9276

TABLE II: Comparison of scores with emulator

Table III shows the levels reached by each agent. These figures give a measure of the agents' survivability. Compared to the Ensemble agent, the MCTS agent is significantly better at surviving. The fruit munching voice does not significantly affect the survivability of the Ensemble agents.

Agent	Minimum	Maximum	Mean	Mode	p-value
Ensemble	6	25	18.37	21	
MCTS	13	24	20.38	22	$< .0001$
Ensemble no fruit	8	24	18.67	21	.5852

TABLE III: Comparison of levels reached with emulator

B. *Results Without emulator*

In terms of both scoring and survivability, all agents were significantly worse off for having the emulator forward model removed (p-value $< .0001$), but he MCTS agent was the most dramatically affected. The MCTS is more heavily dependent on the emulator for accurate checking of paths. The ensemble uses the emulator much less; only for pre-filtering moves, and as an extremely short range (eight frames) safety check.

The full Ensemble's performance drops by about 23% for both average score and average level reached. The Ensemble

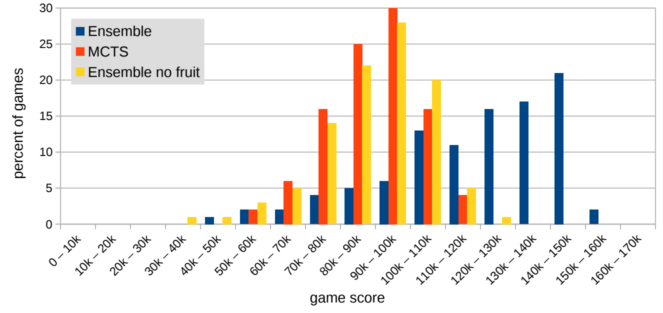

Fig. 4: Chart of score ranges reached with emulator

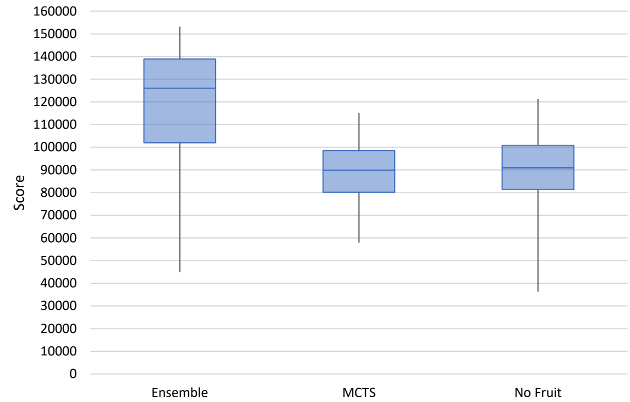

Fig. 5: Box chart of agent scores with emulator

without fruit loses about 19% in average score, and 17% in average level reached. The MCTS agent, however, loses a huge 48% in average score, and 52% in average level reached.

Table IV shows the scores for each agent over 100 games. Compared to the MCTS agent, both of the Ensemble variants are much higher scoring.

Agent	Minimum	Maximum	Mean	Std. Err.	p-value
Ensemble	16760	145510	91167	2848	$< .0001$
MCTS	3930	92660	46672	2028	
Ensemble no fruit	26200	115350	72586	2064	$< .0001$

TABLE IV: Comparison of scores with emulator

In Table V we can see that the MCTS agent is now significantly worse at surviving than the Ensemble agent, and that removing the fruit munching voice does not significantly change the survivability of the Ensemble.

Agent	Minimum	Maximum	Mean	Mode	p-value
Ensemble	3	22	14.19	16	
MCTS	1	20	9.83	7	$< .0001$
Ensemble no fruit	6	23	15.49	21	.0642

TABLE V: Comparison of levels reached without emulator

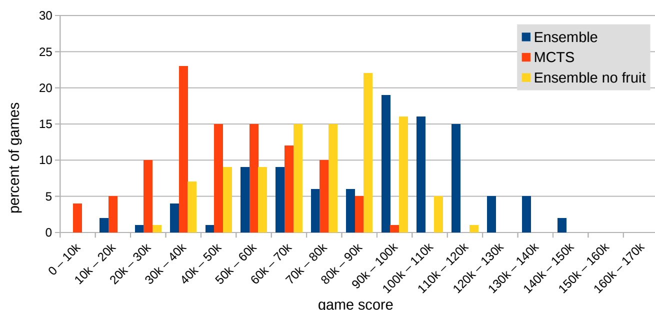

Fig. 6: Chart of score ranges without emulator

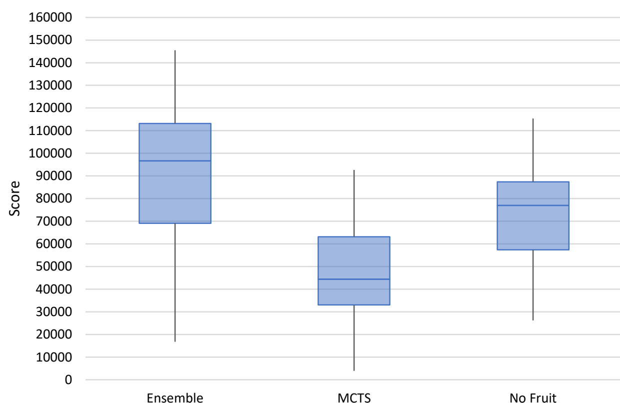

Fig. 7: Box chart of agent scores without emulator

X. Conclusion

This paper described the use of an Ensemble Decision System to create a Ms. Pac-Man agent. It also described a new framework that allows Java controllers to be written for the original Ms. Pac-Man arcade game. Using this framework, and a powerful simulator forward model, we were able to create very high scoring agents.

The Ensemble agent scored a world record AI score for Ms. Pac-Man, outperforming a MCTS-based monolithic agent based on the same system and forward model.

The Ensemble decision systems show great potential as efficient and flexible alternatives to monolithic agents, and also as lightweight augmentations to existing systems. Adding fruit awareness to the MCTS agent, for example. This could be done without interfering with the MCTS algorithm in any way, only potentially altering the chosen move.

XI. Future Work

The simple arbiter could be replaced by something more sophisticated and dynamic; possibly a trained neural network or a genetic algorithm to learn a strategic sense of the game. The experiments evolving the voice weights of the ensemble were mildly successful, but slow and tedious. It does leave open the possibility of using better optimisation algorithms.

The results for Ms. Pac-Man are very good, but it is only one game. We would like to see Ensemble Decision Systems applied to GVGP problems, to get an understanding of how flexible they can be, and if multi-purpose, or reusable, voices can be created.

XII. Code

The code used in this project can be downloaded from GitHub at https://github.com/philrod1/james

Acknowledgements

The authors would like to thank:
George Moralis and the ArcadeFlex developers.
Scott Lawrence for the Ms. Pac-Man disassembly.
Jamey Pittman for the Pac-Man Dossier

References

[1] B. Dasarathy and B. V. Sheela, "A composite classifier system design: Concepts and methodology," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 708–713, May 1979.

[2] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager *et al.*, "Building watson: An overview of the deepqa project," *AI magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[3] Y. Chen, J. E. Argentinis, and G. Weber, "Ibm watson: How cognitive computing can be applied to big data challenges in life sciences research," *Clinical Therapeutics*, vol. 38, no. 4, pp. 688 – 701, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0149291815013168

[4] P. Rohlfshagen, J. Liu, D. Perez-Liebana, and S. M. Lucas, "Pac-man conquers academia: Two decades of research using a classic arcade game," *IEEE Transactions on Games*, vol. PP, no. 99, pp. 1–1, 2017.

[5] G. Foderaro, A. Swingler, and S. Ferrari, "A model-based cell decomposition approach to on-line pursuit-evasion path planning and the video game ms. pac-man," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 281–287.

[6] H. van Seijen, M. Fatemi, R. Laroche, J. Romoff, T. Barnes, and J. Tsang, "Hybrid reward architecture for reinforcement learning," in *NIPS*, 2017.

[7] N. Ikehata and T. Ito, "Monte-carlo tree search in ms. pac-man," in *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, Aug 2011, pp. 39–46.

[8] T. Pepels, M. H. M. Winands, and M. Lanctot, "Real-time monte carlo tree search in ms pac-man," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 245–257, Sept 2014.

[9] S. Warshall, "A theorem on boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, Jan. 1962. [Online]. Available: http://doi.acm.org/10.1145/321105.321107

[10] P. Rodgers and J. Levine, "An investigation into 2048 AI strategies," in *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26-29, 2014*. IEEE, 2014, pp. 1–2. [Online]. Available: http://dx.doi.org/10.1109/CIG.2014.6932920

[11] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation," *ArXiv e-prints*, Feb. 2018.

[12] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General video game playing," *Dagstuhl Follow-Ups*, vol. 6, 2013.