From Snake, Play All the Games

Ashley Jin (ashpjin), Joseph Tsai (joetsai)

12-13-13

1 Introduction

Given almost no knowledge about the game, can an AI efficiently learn to play a game effectively and in real time? In this paper, we explore the idea of using future estimation as a general game-playing strategy. We posit that our generalized AI, built and tested on Snake, can easily learn to play other games as well. The majority of the paper details decisions and results on Snake but tests the AI on another game in section 6.2.

Snake is a popular one player game where the player controls a snake with the objective of increasing score by eating apples until the snake hits its own body or a wall and dies. We explore using reinforcement learning to estimate possible futures from a given state so that the AI can learn the gameplay and to maximize a given objective.

2 Related Work and Inspiration

This project was inspired by Tom Murphy's "The First Level of Super Mario Bros. is Easy with Lexicographic Orderings and Time Travel . . . after that it gets a little tricky" paper. Portions of the paper such as using lexicographic orderings are not relevant to this project. Instead, we focus on using futures analysis in our game-playing AI.

Murphy's game-playing AI classified possible futures as either good or bad and used the classifications to decide its next move. With futures, the value of a candidate state is no longer just a function of how much better it is compared to the current state but also a function of how much better the futures resulting from that candidate are. The AI must make choices that avoid bad futures, generally death, while simultaneously seeking good futures, futures that increase score or survival time. Murphy chose to explore 300-500 frames into the future for Super Mario Bros.[1] and did not experiment with different futures lengths. Although we will vary future lengths, the main goal is for the AI to make moves in real time. As our methods become more complex, the length of time we are able to look into the future is reduced. We will analyze the effects of the future estimation length when discussing experiments.

Murphy also explores backtracking to avoid getting stuck in local maxima during the game. In his experiments, Murphy found that Mario tends to get stuck optimizing score in a way that sacrifices game progress. He attempted to fix this situation by saving some states as checkpoints and considering alternate move sequences from these checkpoints to the current state[1]. If the current moves sequence still results in the best futures, no backtracking occurs. However, if a better moves sequence is found, the game truncates to the checkpoint and performs the alternate sequence (hence "time travel"). This concept differs from exploring futures in that the move sequences are longer and only move sets based on the original sequence are considered. Our snake game will get stuck in a similar maxima if the score generally increases with play time i.e. the play time is a positive term in the objective function. We experimented with different objective functions and backtracking in order to avoid the local maxima issue. These are discussed in sections 5.5 and 6.3.

3 Problem Definition

During development, our AI's goal was to maximize a function of snake length, correlated with the game score and apples eaten, and playtime (also referred to as cycles) without any prior knowledge of gameplay.

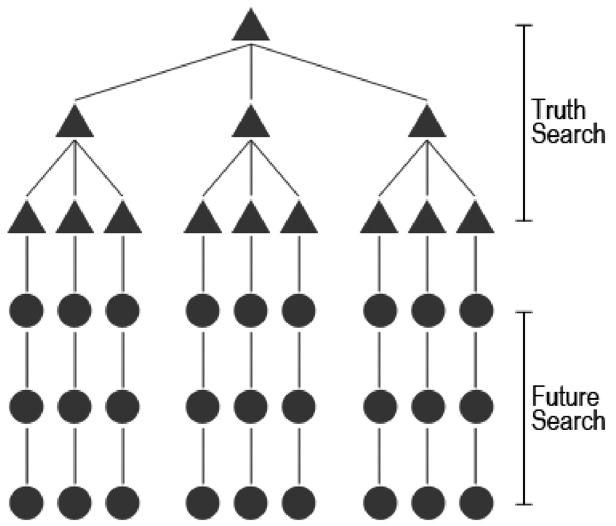


Figure 1: Diagram of search tree for snake AI

The objective defining function requires two weights, one for score and one for time, from a human agent. These weights define the relative importance of each factor. For example, giving score a weight of 1 and time a weight of 0 implies that we want the AI to maximize score only. The weight-based objective function is used to transform the state of the game into a score that the AI can use to measure how "good" a particular state is. We can evaluate the AI's success as improvement in the achieved objective score over the baseline.

In addition to the objective function, the AI has access to a list of next moves as well as the ability to clone the current game state and simulate taking actions. Given the next moves and futures analysis of the "goodness" of possible futures, the AI takes the ideal move for the current state. The snake has at most 4 possible moves in any given state: north, south, east, or west. However, one of the moves is always invalid since the snake cannot move backwards. Because the move set is small and well-defined, our AI considers all futures instead of randomly choosing a subset of possible futures like Tom Murphy's program does. We also store information about "randomness" such that the futures the AI considers are deterministic. For example, the AI can simulate when and where the next bomb will appear despite both being random events in a real game.

Learning to play the game itself requires the use of reinforcement learning techniques. The AI receives a positive reward by doing something that increases the state score (eating an apple or increasing the time cycle count) and a no reward for losing the game. The AI will learn that dying is bad by seeing that there is no way to increase the score after that event occurs. To search the space of possible futures, we combine the maxi portion of MiniMax with a limited-depth, truth depth, with an evaluation function based on: randomness and/or Markov Decision Process.

For this project, we have chosen to model "randomness" deterministically. Having the AI play the game without knowing exactly when and where apples or bombs will appear causes the futures space for any particular game state to grow infinitely. Efficiently searching the futures space and finding an ideal balance between solution quality and runtime would become more difficult but necessary.

4 Method

As mentioned, we will be basing this AI on the maxi portion of a MiniMax search. The "truth search" portion shown in Figure 1 represents this maxi search. It will exhaustively try all possibles moves until a depth of k. In the game of snake, the branching factor b is on average 3 for each of the directions the snake could move in. The computational complexity of the truth section is $O(b^k)$.

Since this aspect of the search is computationally expensive, k is a low value. This portion of the search is called the "truth" search because it exhaustively searches all possible moves and can provide an optimal result with regard to the search depth. The "future" search portion is essentially a special case implementation of the evaluation function for the leaves in a maxi search. It is called the "future" search because it is not a perfectly truthful measure of the utility of each leaf node, only a reasonable estimate. Our implementation of the evaluation function is to simulate j moves starting from each of the "truth search" leaf nodes. The way the each action is chosen will determine the quality of this evaluation function. Because our evaluation function is based on actual simulations, the utility returned is guarunteed to be equal to or lower than globally ideal value.

We explored randomized moves, Markov Decision Process with epsilon exploration, and Markov Decision Process combined with randomized moves. With the truth search and future search operating together, the computational complexity is $O(j \cdot b^k)$. So long as the maxi search depth k, is kept at a reasonable low value, this algorithm is capable of running in real time.

4.1 Random Futures

The baseline "future search" is to randomly select an action from each state until the snake dies or until j steps have been taken.

4.2 Markov Decision Process with ϵ exploration (MDP+ϵ)

Q-learning updates the MDP's Q-value for states on every (s, a, r, s') observed where the items in the tuple are current state, action, reward, and next state respectively. The Q-value for any state is updated using Equation 1.

$$Q_{opt}(s,a) \leftarrow Q_{opt}(s,a) - \eta[Q_{opt}(s,a) - (r + \gamma V_{opt}(s'))] \tag{1}$$

Related variable values are:

- Discount factor: $\gamma = .95$
- Step size: $\eta = .95$
- Exploration probability: $\epsilon = 0.01$
- Reward: $r =$ difference between objective function values of new state and current state
- $V_{opt}(s') = max_{a \in actions} Q_{opt}(s', a)$

We use Q-learning to update the MDP on every true step taken. In the future estimation, the AI will simulate many actions as provided by the MDP model. However, since these actions are not actually taken, the MDP is not allowed to learn from them. Instead, the MDP is only allowed to observe a state if the AI takes an action and arrives in that state in the actual game. We explore the impact of learning during the estimation, future cache, in section 6.3.

Our chosen representation of states is a tuple of (action, enumeration). Enumerations are listed in section 4.4 but the list is not given explicitly to the AI. A function transforms the current game state to an MDP state. This is another piece of information that the AI requires from the user. However, the AI does not have any knowledge about the translation or about what the MDP states represent.

Actions from state to state are the the four cardinal directions the snake is allowed to move in. Since the snake's head points in the direction it just moved, the last action is a good representation of the state since the allowed moves are dependent on directionality. New states are added only when they are seen. To prevent the AI from exploiting only the states and actions it already observed, the initial policy favors exploration with a dynamic epsilon probability. The initial policy is represented as a case function in Equation 2.

$$\pi(s) = \begin{cases} Rand(actions) & \text{if no info on any action} \\ \boxed{\begin{cases} Rand(actions) & \text{with prob } \epsilon \\ argmax_{a \in actions} Q_{opt}(s,a) & \text{else} \end{cases}} & \text{else} \end{cases} \tag{2}$$

When using MDP to estimate the future, the AI adheres to its current learned policy and attempts to take j steps. The estimated reward achieved indicates the goodness of the future according to the current policy.

4.3 Markov Decision Process and Random Futures Collaboration (MDP+R)

An expected weakness in the MDP+ϵ exploration lies with the epsilon. Although the probability is necessary for exploration, randomly exploring when the MDP explicitly knows the best action to take is unwise. Therefore, our final futures estimation method combines the Random Futures and MDP methods previously described. The Random Futures estimation replaces the epsilon exploration probability in the aforementioned MDP+ϵ process.

Basically, we take a maximum over the two estimations. If the estimated future generated by Random is better than the estimated future generated by the MDP+ϵ, take the action recommended by Random and vice versa. At the beginning, when the MDP has not learned much, the Random Futures has an equally good (or better) chance of finding a better future and its recommended action will be taken and learned from. Eventually, when the MDP has observed a large number of states and actions, it will develop a policy that will find futures better than random futures. This estimation method is essentially a "best of both worlds" approach to combining Random futures and MDP+ϵ exploration.

4.4 Implementation Details

We initially used an open source version of the snake game written in Python, Simple Snake[2]. However, in the course of cleaning up the game code, we essentially rewrote the game from scratch. We since extended the game to be easy to save state and added the appearance of bombs. We initially chose this version due to its simple appearance, ease of modification, and bugginess. Tom Murphy's AI was able to find and exploit bugs so we wanted to test if our AI could as well. However, little to none of the original code remains due to our extensions and restructuring. None of the bugs survived. Instead, we experimented with a legitimate feature described in 6.1.

In our version, bombs and apples appear uniformly at random in empty grid spaces at random times. There will always be one or more apples on the board with no upper limit. Bombs appear less frequently but never disappear; they are also capped to never occupy more than 1/5th of the available grid space. Apples and bombs should never occupy the same grid square. Items will never appear in the grid space the snake head moves to in the same iteration to avoid game losses that are fully out of the hands of the player. This decision prevents the AI from dying randomly as opposed to as a result of its own actions. The snake length grows by 1 every time an apple is consumed. If the snake hits its own body, a bomb, or a wall, the game ends. Our version does not speed up the game's pace as a function of play time.

When instantiating the AI agent object, the client is required to pass in a function callback that will be used to compute the game score. The AI's goal is to maximize this objective score as it attempts to play the game. The callback function takes in the game object and returns a floating point score value. The objective function currently being used is based on the number of cycles the snake has stayed alive and the number of snake segments. The default function, shown in the following code block is weighted toward snake segments (indicating more apples eaten) over staying alive. This function proportionally represents the importance of cycles and apples to a human player.

```
def default_objective(game):
        return 0.1*game.num_cycles + 1.0*game.num_apples
```

The client is also required to pass in a function that transforms game state to an MDP state. Creating this function requires extensive domain knowledge. Because we want the game-playing AI to be as general as possible, we relegate the state transformation decision entirely to the client. A sample tranformation function is shown below:

```
def game_state_to_mdp_state(game):
        return game.pre_action, game.pre_state # user defined
```

Our implementation defines the game grid as a 2D array where each element in the array has a type assigned. We refer to these types as grid square enumerations in the paper. Enumerations are: Apple, Bomb,

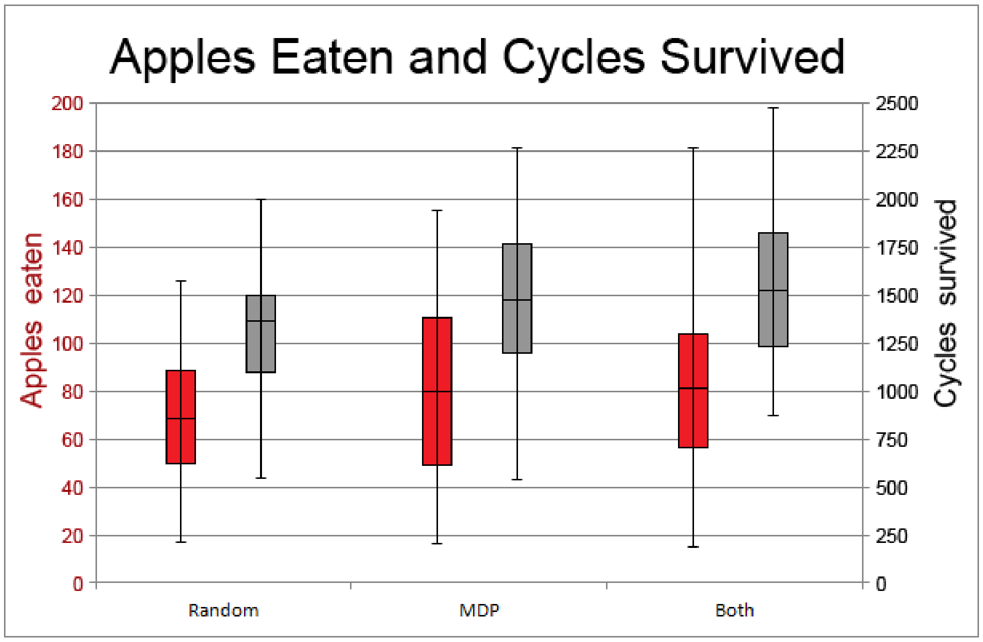


Figure 2: Results of apples eaten and cycles survived using the three estimation methods on the default objective function with regular Snake.

SnakeWest, SnakeEast, SnakeNorth, SnakeSouth, Empty. When the AI is considering what moves to make, it will query the game object for a list of available moves. In all the game simulations, the grid size was 40x40. The probability of apple appearance per cycle was 0.1 and the probability of bomb appearance per cycle was 0.05.

5 Methods Testing

5.1 Naive Baseline (no truth search)

Our baseline is an implementation that only looks one move into the future. If a move does not result in death, it gets added to a list of approved moves. The baseline chooses a move randomly from the list of approved moves. After running the baseline over 50 games, the average number of cycles (unit of playtime) was 570š156 and average number of apples (unit of score) was 7š4. The leading cause of snakey death was self strangulation since the baseline does not consider the long term consequences of each move. This baseline equivalent to using the maxi search algorithm with truth depth k of 1 and future depth j of 0.

Note: For sections 5.2-5.4, the objective function is: maximize $0.1*num\_cycles+1.0*num\_apples$ with truth depth of 3 and future depth of 100 so that the AI could play in realtime. We selected this objective because it captures the proportional importance of cycles and apples to a human player. Testing with different objectives is discussed in section 5.5.

5.2 Maxi with Random Futures

After 50 simulations, the number of cycles for Random Futures was 1363š278 and number of apples eaten was 67š25. Given that at most $\frac{1}{5}$ of the board can be occupied by bombs, the theoretical maximum number of apples that can be eaten is 1280. The average score of 67 apples indicates that the AI has much room to improve. We expected that randomly estimating the future would be fast but inaccurate. However, we found that randomly estimating the future can perform well though the results vary significantly.

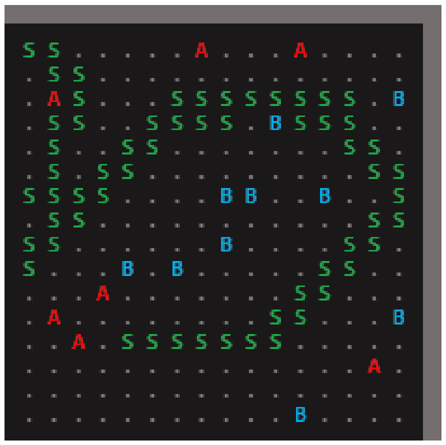


Figure 3: Diagonal snake pattern.
Exploration was turned off in order to generate an obviously diagonal pattern.

Analysis of the AI in operation produced some interesting points. First off, choosing moves with equal ratings uses random arbitration. This leads to erratic movements that lack any well defined direction. Ideally, the AI should learn that maintaining a consistent heading is usually beneficial to the overall score. Also, the snake would often spend a significant portion of the simulation cycles moving around without consuming any apples (even if they were directly adjacent). However, once the snake ate its first apple, it lost its fear of apples and started eating more frequently. Analysis of the AI state revealed that this was due to the fact that when the game starts, the default length of a snake is 3. Due to the physical layout of 3 squares on a grid, it is impossible for a snake of length less than 4 to hit itself. However, after consuming even one apple, the snake grows to length 4, where self strangulation becomes a possible future.

In the beginning, when the snake has a length of 3 and is next to an apple, the future search will show that the snake can survive longer by not eating the apple than by eating it. This is an artifact of the future-simulations being completely random with no intelligence. One possible idea counter to this problem is to multiply a discount factor on the result of the future search. This discount would weight the truth search more heavily than the future search in this scenario.

5.3 Maxi with MDP+ϵ

The MDP with epsilon futures using the same truth and future depth performed better overall than the random futures. Again, the performance showed high variance. Results are shown in Figure 2. This predictor was able to survive for 1446ś424 cycles and ate 81ś38 apples on average. The MDP based future demonstrated that the snake would learn to consistently move in the same heading. This is beneficial in that it allows the snake to move to new areas with more apples and also to keep itself spread out (thus avoiding self strangulation). In addition, the MDP will insist on following its learned policy and will be unable to adapt during the future estimation. For example, if the MDP's policy insists that going North is a good idea, it will continue choosing to generally head North in its future estimation even if it gets very close to wall in which case, turning usually results in a higher score. The random activation of the epsilon exploration allows the snake to twist and turn every now and then to explore different areas or to realize its heading for a dead-end.

Interestingly, the AI learned to travel in a new way by using the MDP. Because the default objective valued time cycles, the AI would sometimes choose to travel in a diagonal path, pictured in Figure 3, to its desired destination instead of going straight. This is because the Manhattan distance, and therefore accrued time cycle score, is greater when traveling diagonally on a grid rather than in straight lines.

5.4 Maxi with MDP+R

Taking the maximum future estimation's associated action resulted in consistently higher scores than either Random Futures or MDP+ϵ (See Figure 2). Although the variance is still high, the scores are much more consistent. This behavior was expected since this method of estimation is designed to counter MDP+ϵ's shortcomings. This future predictor was able to survive for 1546ś424 cycles and ate 81ś38 apples on average.

We observed that the AI relied heavily on Random Futures at the beginning when MDP had learned very little and also close to the time of death. The AI relies on Random close to death because it has usually trapped itself and the MDP's policy is insufficient for finding an escape. Random has a much better chance of trying an action that leads to escape because its options are unconstrained by a policy. At many non-startup or wind-down times, the AI relies on the MDP to choose actions since MDP was able to bring the snake to new patches of apples by travelling in a consistent heading.

5.5 Alternate Objectives

Running MDP+R experiments on different objective functions produced encouraging results. The observed behavioral patterns along with brief analysis are presented below.

5.5.1 I Hate Apples

The objective function here is maximize $1.0 * num\_cycles$.

With this objective, eating apples to increase score is completely irrelevant. During games with this objective, the AI would find a nice corner and cycle infinitely to avoid eating any apples. Technically, this AI beats the game, assuming you let it play forever.

5.5.2 I love Apples!

The objective function here is maximize $1.0 * num\_apples$.

With this objective, the AI always ate the apple if the apple was within the truth search depth. When the apple was further away, the snake exhibited a high probability of self-strangulation because eating the apple was deemed more important than longevity.

5.5.3 Moving is Hard Work

The objective function here is maximize $-0.1 * num\_cycles + 1.0 * num\_apples$.

With this objective, the AI would also quickly try to eat apples. It also preferred moving in straight lines as opposed to in diagonals. The AI would not explore a majority of the grid, preferring to stay close to the area it started in. Additionally, the AI would not survive for as long as it had been for two reasons: 1. again, it would sometimes rush into a situation where self-strangulation was a definite future simply because there was an apple in that direction that it wanted to eat and 2. if it could not find any apples in the truth or future search, it would decide that suicide would be the best way to maximize the objective. These behaviors are expected and seem to imply that the AI is general enough to adapt to different game requirements.

5.5.4 Only as Good as Your Apples

The objective function here is maximize $0.1 * min(num\_cycles, num\_apples * 5) + 1.0 * num\_apples$.

This objective was an attempt to force the snake to eat apples without necessarily ignoring cycles. Here the snake only points for score up to 5 times the number of apples it has eaten. However, these was little observable difference from the original objective. This is because the AI is not smart enough to learn how the objective function operates. It would realize that sometimes moving would gain a reward and not at other times. The AI is unable to capture the minute changes that cause the objective to switch. We believe this type of hinge objective function will consistently stump our AI (at least with the current state representation).

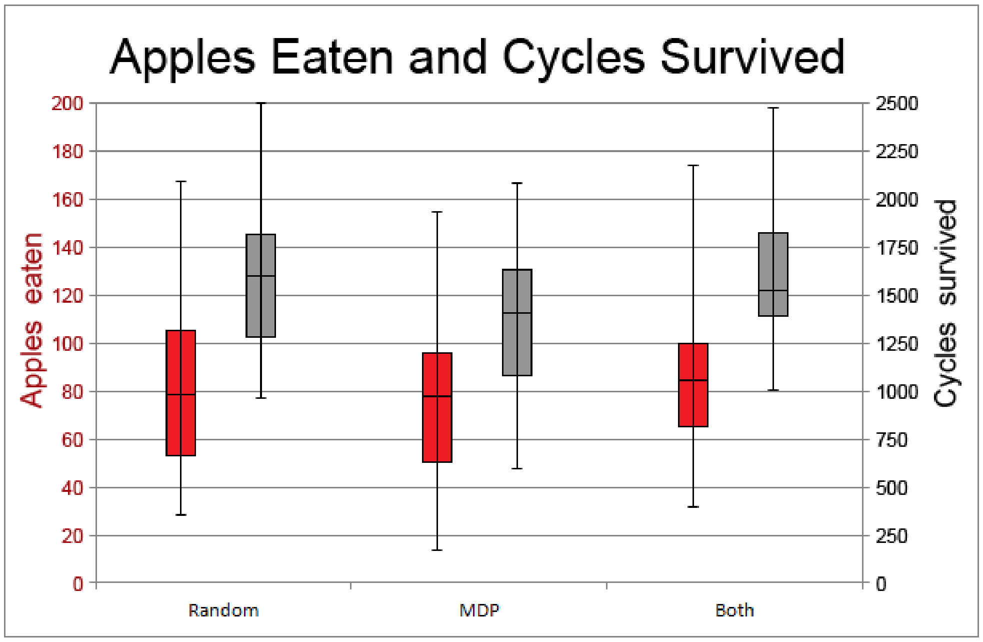


Figure 4: Results of apples eaten and cycles survived using the three estimation methods on the default objective function with Snake and cyanide pills.

6 Experiments

For these experiments, the truth and estimation depth may vary. We chose to constrain the depths with respect to whether the AI would be able to play in real time. We targeted 100-200ms to make an action choice on a single core of a Intel core i7 with 3.4GHz processors. Actual reported execution times may be slower because we are overutilizing all CPUs.

6.1 New Feature: Cyanide Pill

We added a Cyanide pill as an extra feature to our original Snake game. The Cyanide pill reduces the snake's length by 1 when eaten which in turn, lowers the game score but generally makes it easier for the snake to navigate the grid. Although, we did not give the AI any intuition about this new feature except to add it as an Enumeration that the AI will see if it enters that grid space, it learned to abuse the Cyanide pill to stay alive!

If the objective function included time cycles as a positive factor, the AI would sometimes consume the pills, reduce its length to 1, and navigate the remaining game without fear of strangulation. This behavior relates to the previous observation where the AI would observe that futures where snake length was greater than 3 were usually worse, resulted in death more frequently, than futures where length remained 3. We used $k = 3$ and $j = 100$ like before.

Using the MDP+R predictor, the AI was able to stay alive for 1616ś310 cycles and eat 87ś31 apples on average as seen in Figure 4. This is higher than the original game without cyanide pills. The reason for a higher score is because the cyanide pill allows the snake to escape perilous situations in some cases. For example, if the snake was surrounded by bombs and cyanide pills, it would eat the pill realizing that the deduction of a snake segment was a lesser evil than outright death.

6.2 New Game: Simplified Space Invaders

To test the reusability of our game-playing AI, we ran it on a simplified Space Invaders game, pictured in Figure 5. The simplified game allows the player to take actions left, right, or shoot. The invaders shoot with a probability of 0.1 and move left or right slowly but never move down. The objective function is defined as

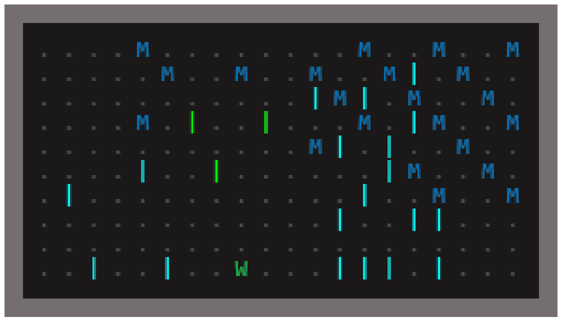


Figure 5: Example screenshot of the Space Invaders game in play.

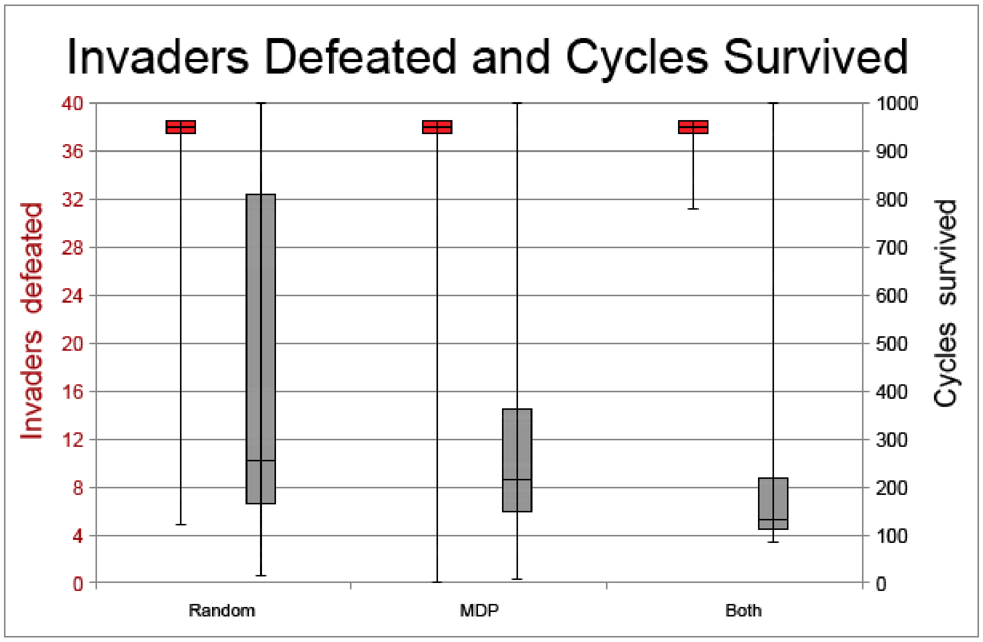


Figure 6: Results of number of invaders defeated and cycles played using the three estimation methods on Space Invaders.

maximize $-0.1 * num\_cycles + 1.0 * invaders\_destroyed$. The MDP states are defined as (previous action, enumeration of square directly above).

Our implementation of Space Invaders was inefficient and thus, we had to reduce the truth depth to 2 and the future depth to 10 in order for the AI to be able to play in realtime. Despite, the severely limited future foresight, the AI was still able to win at the game a vast majority of the time. The results shown in Figure 6 show that at least 75% of the time all three predictors would beat the game by destroying all 38 invaders. In a sample size of 50, both Random and MDP had cases where it died before destroying even a dozen enemy combatants. The MDP+R predictor at worst only destroyed 31 invaders.

The main metric of interest here is the number of cycles that the AI would play the game for. In order to avoid the AI getting stuck in an infinite loop (for example where the invader is on the left side and the AI refuses to leave the right side), the game was written to automatically cause player death if the number of cycles exceeded 1000 turns. In all three predictor, we can see that this worst case of 1000 cycles was unfortunately exercised at least 25% of the time. However, as we sweep across the predictors (Random, MDP, MDP+R), we can see a clear progression of the amount of cycles the AI needs to clear the game on median. The MDP+R predictor was able to clear the game in 139 cycles on median, MDP took 222 cycles on median, and Random took 251 cycles on median.

6.3 Backtracking and Future Cache

We experimented very simply with backtracking and future search. All the simulations that include backtracking and/or future search were run with a truth depth $k = 2$ and estimated future depth of $j = 40$. Again, these values were chosen so that the AI could play in real time.

Backtracking was implemented only for Random Futures. Implementing backtracking for MDP would have involved a permutation similar to Tom Murphy's backtracking. We did not have time to do this so we only discuss Backtracking for Random Futures. If the Random Future entered a state that had no available actions (equivalent to dying), the program would backtrack one move, remove the offending action from the actions list, and try again. If there were no more actions left, then the objective value was then returned. Note that this backtracking is not recursive. The program is only allowed to backtrack one time after one estimated death. If a second predicted death is encountered, the program must return the objective function value for the state. The ability to backtrack required that the game state be copied before evaluating an action. This was needed, so that the state could be "reverted" if the action resulted in death. The need to always copy game state severely slows down the AI execution rate; this was the reason the truth and future depths were reduced.

Future cache is implemented for both Random Futures and MDP+ϵ. The future cache stores a single path of up to $k + j - 1$ moves that resulted in the highest objective function value from the previous iteration estimations given that we took the first action in the sequence. Before the next iteration's estimations, calculate the score for the cached path and include that when considering best actions. If any of the estimated futures have a higher objective, the AI will still pick the action associated with that future and update the cached future to the better one. However, if the cached future is better than any newly estimated future, the AI will take the next step along the cached path. The general idea behind the future cache is that the Random Futures was sometimes able to find amazingly good futures by sheer luck. However, since the path that resulted in this lucky future was not stored and is not persistent across moves, the next move would be unlikely to find this lucky future again. If the best future and the path to that future were to be cached, then this known good future would be persistent across moves and hopefully allow for a higher score.

In Figure 7, a screenshot is shown of the Snake game being played with MDP+R with backtracking and a future cache. This specific run was able to reach a record high of 273 apples eaten and 3622 cycles survived.

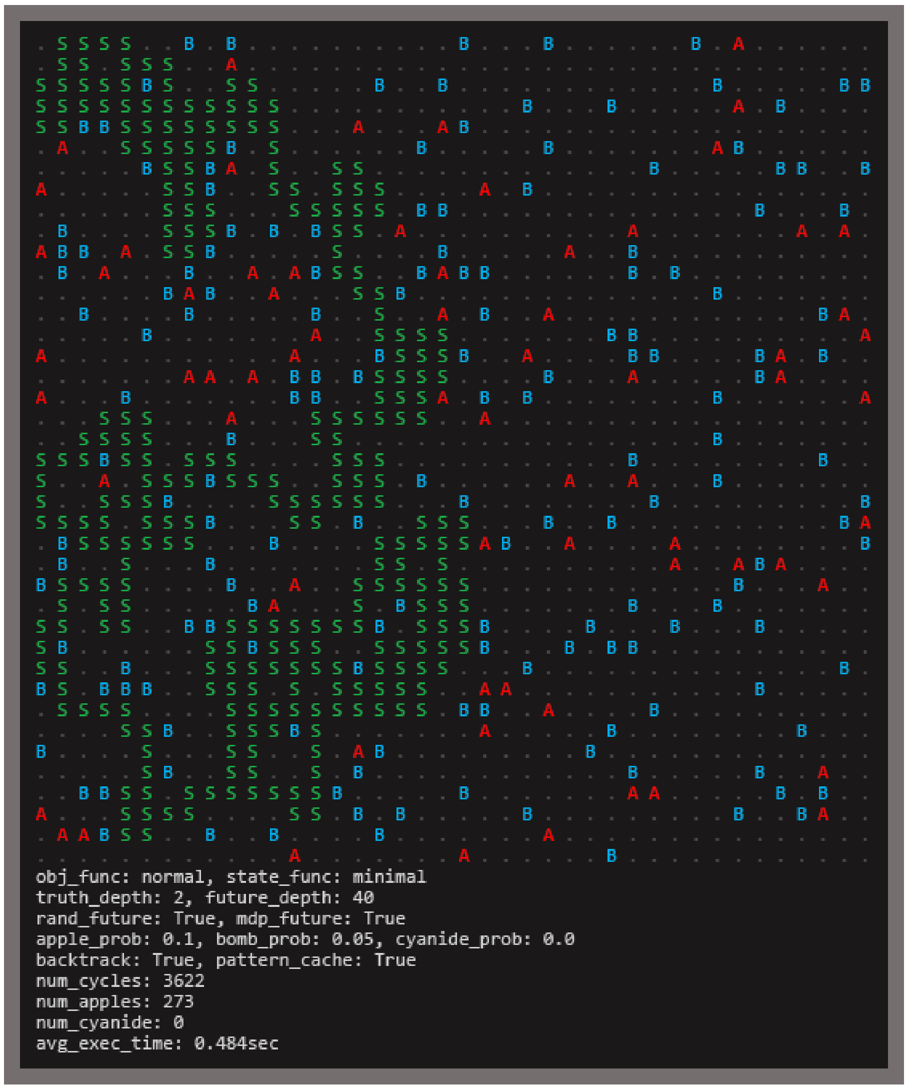
The Figures 8 and 9 show the results of backtracking and future cache applied to regular snake and snake with cyanide pills. The addition of backtracking to Random alone showed an improvement of 29 apples and 284 cycles in regular snake and an improvement of 10 apples and 68 cycles in snake with cyanide pills. Backtracking added to MDP+R similarly showed substantial gains. The most impressive gains were seen in when MDP+R was combined with both backtracking and future cache. Here, the AI was able to achieve 148ś45 apples eaten and 2096ś465 cycles survived for regular snake and 156ś47 apples eaten and 2224ś524 cycles survived for snake with cyanide pills. For regular snake, this represents an average of 2114% gain over the naive AI baseline.

7 Future Work

There are numerous small tweaks and tests that we thought of but did not have the time to test. We could weight the truth portion more than the estimation portion in choosing actions since the truth should have more impact than a guess. We could also vary the constants, i.e. gamma, epsilon, and eta, in the MDP to see the effects of changing each.

Although we dabbled with backtracking and future caching, there are still many more variations that were untested. It is unclear, at the moment, whether dropping the ability to play in real time and allowing the truth and future depths to be longer would make the AI's decisions better or if the current weaknesses are inherent to the methods and choices made.

We would also have liked to test the AI on more different games such as Pacman and Minesweeper. Although our AI works well on our simplified Space Invaders, it would be interesting to see if there is a family of games that our AI simply cannot adapt to play.



```
. S S S S . . B . B . . . . . . . . . . B . . . B . . . . . . B . A . . . . . .
. S S . S S S . . A . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
S S S S S B S . . S S . . . . . B . . B . . . . . . . . . . . . B . . . . . B B
S S S S S S S S S S S S . . . . . . . . . . . B . . . B . . . . . A . B . . . .
S S B B S S S S S S S S . . . A . . . A B . . . . . . . . . . . . . . . . . . .
. A . . S S S S S B . S . . . . . . B . . . . . B . . . . . . . A B . . . . . .
. . . . . B S S B A . S . . S S . . . . . . . . . . . . . B . . . . . B B . . B
A . . . . . S S B . . S S . S S S . . . . A . B . . . . . . . . . . . . . . . .
. . . . . . S S S . . . S S S S S . B B . . . . . . . . . . . . . . B . . . B .
. B . . . . S S S B . B . B S S . A . . . . . . . . . . A . . . . . . . A . A .
A B B . A . S S B . . . . . S . . . . B . . . . . A . . B . . . . . . . . . . .
. B . A . . . B . . A . A B S S . . B A B B . . . . . . B . B . . . . . . . . .
. . . . . . B A B . . A . . . S S B . . . . . . . . . . . . . . B . . . . . . .
. . B . . . . B . . . . . B . . S . . A . B . . A . . . . . . . . . . . . B A .
. . . . . B . . . . . . . A . . S S S S . . . . . . . B B . . . . . . . . . . A
A . . . . . . . . . . . A . . . B S S S B . . A . . . . B B . . . . B A . B . .
. . . . . . . A A . A . B B . B S S S S . . . . B . . . A . . . . . B A . . . .
A . . . B . . . . . . . B B . . S S S A . B . B . . . . . . . . B . . . . . . A
. . . S S S . . . A . . . S S S S S S . . A . . . . . . . . . . . . . . . . . .
. . S S S S . . . B . . . S S . . . . . . . . . . . . . . . . . B . . . . . . .
S S S B S S . S S S . . . . S S S . . . . . . . . . . . B . . . . . . . . B . .
S . . A . S S S B S S S . . S S S . B . . . . . . A . . A . . . B . . . . . . .
S . . S S S B . . . . S S S S S S . . . B . . . . . . . . B . . . . . . . . . B
S S S S . S S S B . . . S S . B . . S S S . . . B . . B . . . . . . . . . . B A
. B S S S S S S . . . B . . . . S S S S S A B . . A . . . . A . . . . . . . . B
. B . . S . . . B . . . . . . . S S . S . . . . . . . . . . A . . A B A . . . .
B S S S S . . . . B . . A . . S S S S S S . . . . . . . . . . . . B . . . A . .
. S . S S . . . . . B A . . . S . B S S S . . . . . . . B . . . B . . . . . . .
S S . S S . . B B S S S S S S S B . S S S B . . . . B . . . B . . . B . . . . .
S B . . . . . . S S B S S S . . S S S S S B . . . B . B B . . . . . . . . . . .
S S . . B . . . S S S S S S S B S S S S . . . B . . . . . . . . . . . . . . B .
B S . B B B . . S S S . S . S S S S S . . A A . . . . . . . . . . . B . . . . .
. S S S S . . . . S S S S S S S S S S . B B . . A . . . . B . . . . . . . . . .
. . . . S S B . . S S S B S . . . . . . . A . . . . . B . . . . . . . . B . . .
B . . . . S . . . S S . . S . A B . . . . . . . . . B . . . . . . . . . . . . .
. . . . . S B . . S S . . S . B . . . . . . . . . . . . B . . . . . B . . A . .
. . B B S S . S S S S S S S B . . . . . B . . . . . . . A A . . . . . B . B . .
A . . . S S S S . . . . S S . B . B . . . . . B . . . . . . . . . B . . B A . .
. A A B S S . . B . . B . . . . B . . . . . . . A . . . . . . . . . . . . . . .
. . . . . . . . . . . . A . . . . . . . A . . . . . . B . . . . . . . . . . . .
obj_func: normal, state_func: minimal
truth_depth: 2, future_depth: 40
rand_future: True, mdp_future: True
apple_prob: 0.1, bomb_prob: 0.05, cyanide_prob: 0.0
backtrack: True, pattern_cache: True
num_cycles: 3622
num_apples: 273
num_cyanide: 0
avg_exec_time: 0.484sec
```

Figure 7: Sample game using MDP+R with backtracking and future cache.

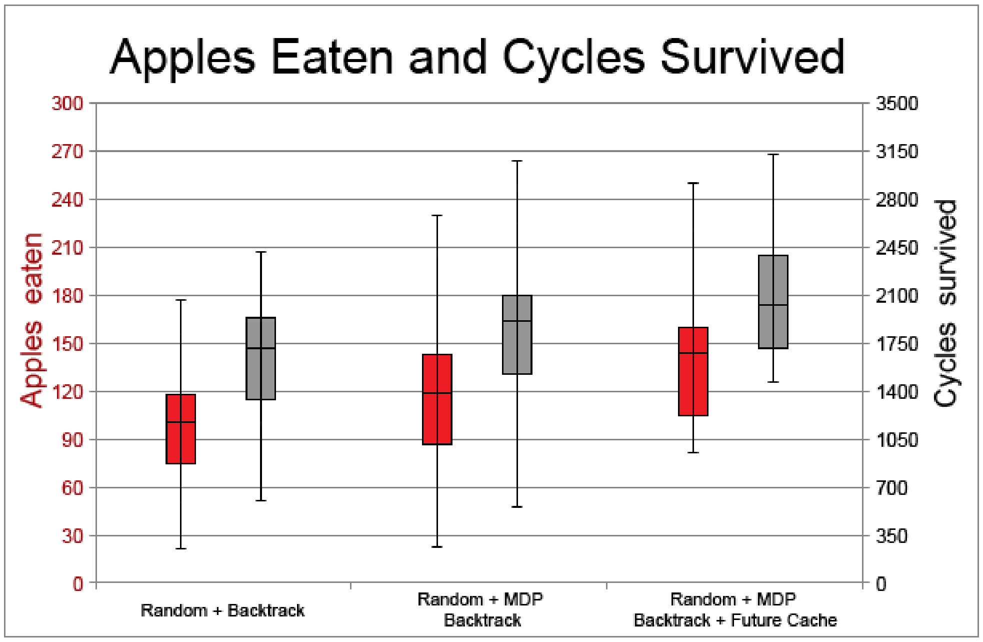


Figure 8: Results of apples eaten and cycles survived using the three experimental estimation methods on the default objective function with regular Snake.

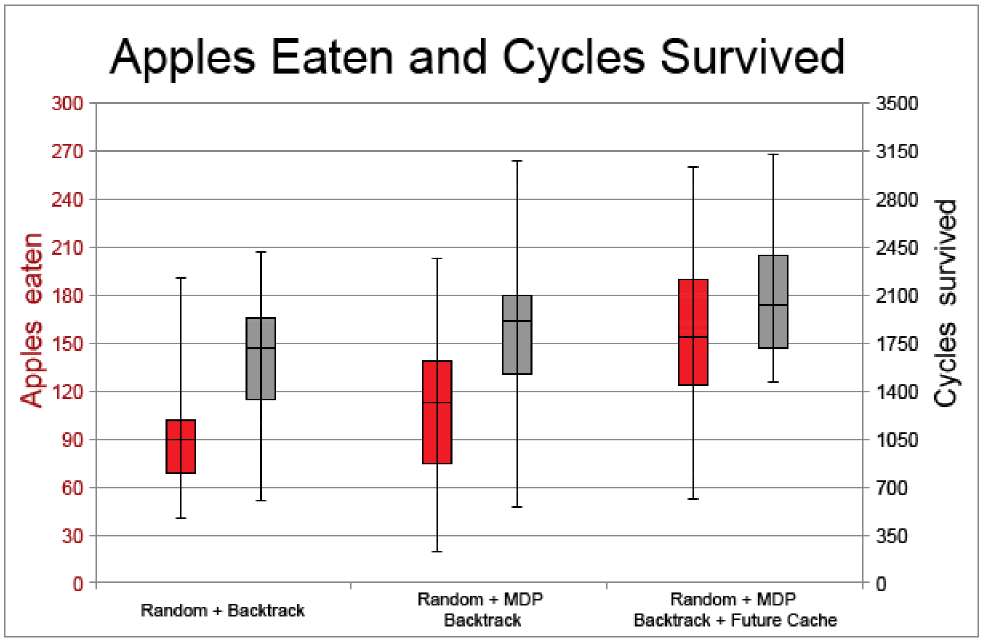


Figure 9: Results of apples eaten and cycles survived using the three experimental estimation methods on the default objective function with Snake and cyanide pills.

8 Conclusion and Challenges

Generalizing a game-playing AI is a difficult task mostly due to the gray area issues of "what counts as too much domain knowledge". By focusing solely on estimating the future and reducing domain knowledge to two user provided functions (objective and transforming game to MDP state), we were able to write a generalized AI with potential to adapt to many games.

However, choosing the objective and state transformation functions are difficult problems themselves. Different objectives can result in vastly different behavior and state transformation choice requires great domain knowledge on the part of the provider. In addition, game states may be infinite. Generalizing infinite state to finite states in a way that makes logical sense is not a trivial task.

Although the human client must give the AI two very specific pieces of information, the AI itself is adaptable. We consider this a success especially because there are still numerous ways to improve the AI using more backtracking, future caching, and minor tweaks. We hope that this AI can eventually learn to win snake!

References

[1] Murphy, Tom. 2013, The First Level of Super Mario Bros. is Easy with Lexicographic Orderings and Time Travel . . . after that it gets a little tricky. http://www.cs.cmu.edu/~tom7/mario/mario.pdf (10 Oct. 2013)

[2] Simple snake (2012), http://www.pygame.org/project-S.S.+Simple+Snake+-2447-.html (10 Oct. 2013)