AA 228 Final Project - The Candy Crusher

Antoine Alary
Civil and Environmental Engineering
aalary@stanford.edu

Alexis Bonnafont
Energy Science Engineering
bonnaf@stanford.edu

***Abstract*—This paper investigates decision-making under uncertainty through the lens of puzzle-solving games, specifically Candy Crush, which presents stochastic dynamics, complex state spaces, and sequential decision-making challenges. We compare multiple strategies for optimizing gameplay, including baseline random and greedy approaches, Monte-Carlo Tree Search (MCTS) with various configurations, a Convolutional Neural Network (CNN) for offline training, and a hybrid MCTS-CNN model. A custom game interface was developed to emulate Candy Crush mechanics and facilitate testing. Algorithms were evaluated on score performance and computational efficiency. Results demonstrate the superiority of MCTS-based methods. Using enhanced implementations with fixed-depth exploration or CNN-guided evaluations, the running time for MCTS was cut in half. These advanced approaches significantly outperform baseline algorithms and exhibit competitive performance with human players. Therefore, this work highlights the effectiveness of combining search-based and learning-based methods for decision-making in stochastic environments.**

I. Introduction

Decision-making under uncertainty is a critical challenge in artificial intelligence (AI), particularly in domains involving stochastic environments, complex state spaces, and sequential decision-making. One such domain is puzzle-solving games like Candy Crush, which exemplify these challenges due to their combinatorial nature, dynamic game mechanics, and the need for strategic foresight. These games require agents to balance immediate rewards against long-term objectives, often in the face of incomplete information and evolving board configurations.

[1] descibes general AI applications in the context of games as of 2020, discussing NPC training, game-generation and player experience modeling. In particular, [2] and [3] show implementations of Reinforcement Learning techniques for FPS and Atari games. The most notable application of AI to games might be the creation of Alpha Zero by Google Deepmind. Authors describe in [4] how their general-purpose algorithm, mixing online Monte-Carlo Tree Search with offline Neural Network training can yield supra-human results on three of the most famous games in history: chess, shogi and Go. Furthermore, [5] delves into the subject of AI ethics in the context of games, exploring important considerations like artificially induced emotions and privacy, that must always be kept in mind when working on AI applications. Finally, the MCTS algorithms described in this paper are derived from the textbook [6].

In this project, we explore various algorithms to enable efficient decision-making in the context of the Candy Crush game. We focus on designing, implementing, and comparing different strategies for an agent to maximize its score within a constrained number of moves. The work aims to analyze how these algorithms perform under uncertainty and assess their ability to optimize decisions by leveraging simulations, heuristics, and learning techniques.

To facilitate experimentation and ensure algorithmic reproducibility, we implemented a game interface that emulates the core mechanics of Candy Crush. Additionally, we set up standardized game scenarios to compare the algorithms based on their performance and computational efficiency.

Through this project, we aim to provide insights into the strengths and limitations of different decision-making approaches, including online, offline and mixed approaches. This will contribute to the broader understanding of AI strategies in sequential decision-making under uncertainty. The findings may extend to other games or to other domains where adaptive and informed decision-making is critical.

II. General methodology

A. Game interface

To optimize for playing strategies, an interface is needed for the agent to train on. Because direct access to the game was not possible, a copy of the game was developed. It allowed for easier interactivity and the possibility to add features. A Candy Crush board was represented as a 7x7 matrix of Candy objects, encoding the color and type of the candy. The main game dynamics were implemented. A move consists in swapping two candies. A move is legal if it leads to an alignment of 3 or more same colored candies, or if it swaps two special-typed candies. When three candies are aligned (or when special candies are swapped), they are popped and disappear. When more than three candies are aligned (or if they form a specific shape), special candies are created. When special candies are popped, they in turn pop all candies on their row/column (striped candies), all candies in their immediate surrounding (wrapped candies) or all candies of a given color (color bomb candies). More specific dynamics apply when swapping two special candies. The score is updated by increasing linearly with the number of candies popped, with significant bonuses for special candies. This leads to a double incentive to create special candies: they lead to more candy popping, and give a score boost when they are themselves popped. Once a move is performed and all candies that needed to pop disappeared, candies fall from row above, and new candies appear with a randomized color at the top of the board.

Although the implementation described above is not a perfect replica of the game dynamics, it is sufficiently detailed to allow for complexity. The reader is encouraged to interact with it on the following link.

B. *Game set-up for algorithm comparison*

All the agents that will be described in the following sections will go head-to-head in playing the game on randomly generated initial boards. A succession of 10 moves will be played and final scores as well as runtime will be compared. The number of 10 moves was selected to balance computational tractability and the ability to develop complex strategies involving creating and popping special candies.

III. Agents algorithm and training

A. *Baselines*

The first baseline we are going to compare the agents to is a random baseline. For this algorithm, a random move is selected among the legal moves and played, and the board is then updated.
The second baseline is a greedy algorithm. The agent simulates all the legal moves and selects the move that leads to the best immediate score improvement.
Finally, on a small subset of boards, we will compare the trained agents to human playing. For this, two people with reasonable background in playing the game selected moves according to their best judgment.

B. *Monte-Carlo Tree Search*

Monte-Carlo Tree Search (MCTS) is an online, simulation-based search technique that incrementally builds a search tree and uses random rollouts to evaluate the long-term consequences of actions. In all MCTS algorithms presented subsequently, a UCB1 metric was used to balance exploration and exploitation, with a hyperparameter C_{explo} driving this trade-off.

In our implementation, the primary challenge was handling the high degree of uncertainty in the future states due to the stochastic nature of candy drops and the appearance of new candies. This uncertainty can cause exponential growth in the state space and reduce the utility of deep rollouts. To mitigate this, we introduced a hyperparameter that limits the maximum number of child states generated from a given state-action pair. Once this limit is reached, we randomly select from the previously generated next states during subsequent expansions. This design choice constrains the branching factor, allowing the MCTS to focus on a more manageable set of candidate moves.

Furthermore, we explored a variant of MCTS with a fixed search depth. Since simulating the game too far ahead becomes less informative due to the increasingly random distribution of newly generated candies, limiting the tree depth can yield more stable predictions of outcome quality. By capping the search depth, we reduce computational overhead and ensure that the agent's computed values are based on states for which the random candy generation is less dominant. In practice, this fixed-depth MCTS provides a balanced trade-off between computation time and the quality of decision-making, allowing for deeper strategic reasoning without incurring excessive computational costs or noise from highly uncertain future states.

C. *Offline training*

While MCTS relies on online computation and rollouts, we also implemented an offline learning approach using a Convolutional Neural Network (CNN). The goal of this approach is to approximate a value function or Q-function, enabling the agent to quickly evaluate states without performing extensive online search.

To build this value approximation model, we generated a dataset of 100,000 board configurations with corresponding estimated Q-values of each actions, derived from running a limited-depth MCTS on the boards. Each board was represented as a multi-channel input where each channel corresponds to a particular candy type and encodes its spatial distribution.

Despite the complexity of translating raw board configurations into accurate long-term value estimates—particularly because the reward structure (cascading matches and special candy creation) is not straightforward—the CNN captured useful state features. For instance, as we can see on Figure 1 it tended to identify configurations with a high potential for special candies or large matches. Although the predictions were not perfectly calibrated, the CNN consistently assigned higher values to states where strategically beneficial moves (such as setting up chain reactions or special candy formations) were available. This offline-trained model thus provides a computationally inexpensive state evaluation tool that can inform other decision-making algorithms.

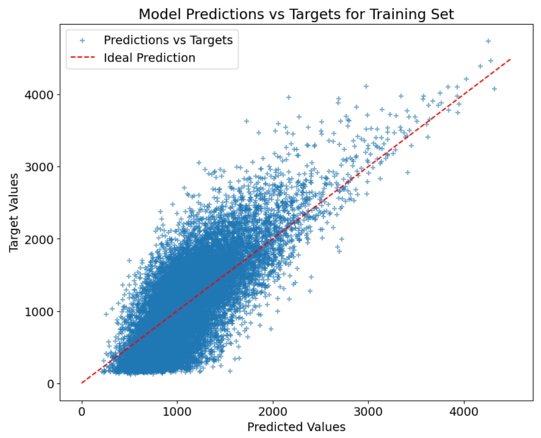

Fig. 1. Offline Model Prediction on Training Set

D. *Combined algorithm*

A key enhancement over pure search-based or purely learned approaches is to integrate the neural network's learned value function into the MCTS framework. Rather than relying solely on random rollouts or shallow heuristics, this combined method uses the CNN's predictions to guide both the selection and evaluation phases of MCTS.

By combining the neural network's ability to recognize high-potential configurations with MCTS's structured exploration, the hybrid approach efficiently narrows the search space and provides more reliable state evaluations. In turn, this reduces computational overhead and could lead to more informed decisions.

IV. Hyperparameters Tuning

Choosing hyperparameters is critical for balancing the trade-offs between computational cost, search depth, exploration, and evaluation quality in MCTS. In particular, we focused on two key parameters: the exploration constant C_{explo}, which governs the exploration-exploitation trade-off in node selection, and N_{random}, which controls the maximum number of random child states considered when expanding a node. Our objective was to identify values for C_{explo} and N_{random} that maximize performance given a fixed number of simulations (2,000).

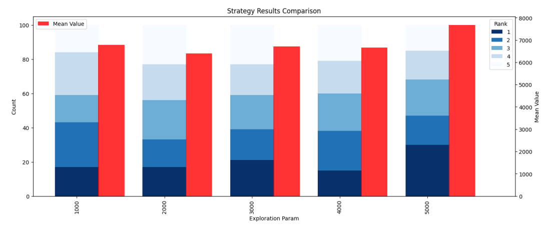

Fig. 2. Impact of varying C_{explo} on MCTS performance. Higher values increase exploration, but may dilute focus on promising moves.

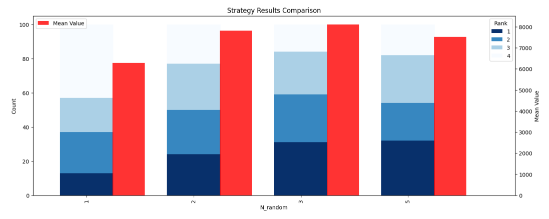

Fig. 3. Influence of different N_{random} values on MCTS. Adjusting this parameter controls branching complexity and rollout variability.

As shown in Figure 2, an exploration parameter C_{explo} of approximately 5,000 consistently achieved a favorable balance, preventing premature convergence on a single strategy and allowing sufficient exploration of alternative moves. Meanwhile, Figure 3 demonstrates that limiting the branching factor to $N_{random} = 3$ helped stabilize the quality of the search, preventing excessive branching that would dilute the computational budget among too many low-quality states.

V. Results

To evaluate our methods, we conducted a large-scale comparison using 1,000 randomly generated initial boards. Each algorithm was allowed to perform a sequence of up to 10 actions to maximize the total score. The process was repeated across all boards, with each algorithm selecting actions iteratively until the 10-action limit was reached. After completing this process on one board, the algorithm moved on to the next.

For each test, we recorded both the final score and the total computation time required to process all 10 actions on a given board. By aggregating these results, we can assess not only the average performance of each algorithm in terms of score, but also its computational efficiency. The resulting data provides a comprehensive comparison that captures both the effectiveness and the scalability of our decision-making approaches.

A. Score and ranking of each strategy

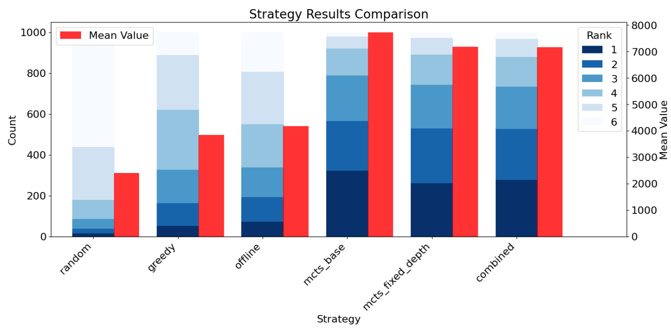

Fig. 4. Average final scores and ranks achieved by each algorithm over 2,000 test boards

The Figure 4 provides a graphical comparison of the average final scores achieved by each algorithm over the 2,000 test boards. This visualization highlights how different decision-making strategies trade off between immediate gains and longer-term planning. The height of each bar indicates the mean final score obtained after 10 actions.

B. Comparing strategy head to head

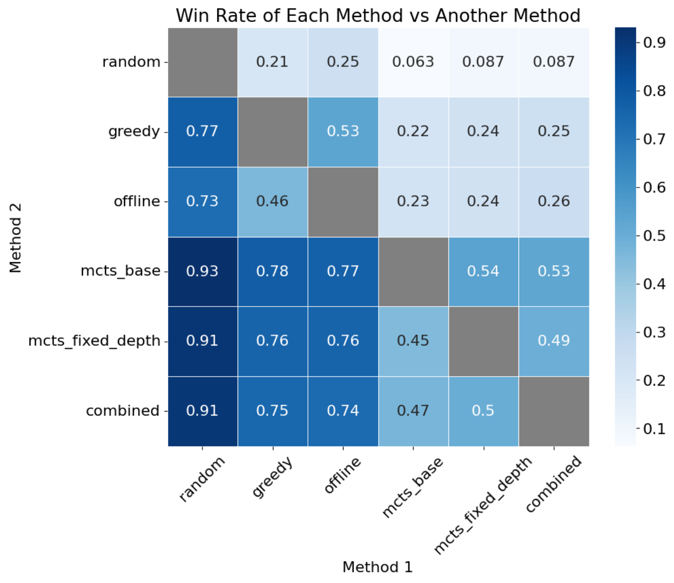

Fig. 5. Pairwise win rates between algorithms

The pairwise win rate matrix (Figure 5) clarifies how each method performs when directly compared to another. MCTS variants, including the fixed-depth and combined approaches, dominate the simpler baselines (random and greedy) and show strong advantages over the offline-only approach. The combined strategy often maintains a competitive or superior edge even against other MCTS variations, highlighting that combining learned evaluation with tree search yields more robust and consistently better decision-making.

While the greedy method slightly outperforms the offline model in head-to-head comparisons, it is important to note that the offline approach does achieve the top score on some boards and, on average, can produce higher peak results. This discrepancy can be explained by the nature of the board distributions. Most boards are relatively unpromising, and for these the CNN's output is noisy and does not consistently outperform the straightforward greedy heuristic. However, on the relatively rare boards that present high potential for large chains or special candy formations, the offline-trained CNN excels. In these cases, it identifies opportunities that the greedy approach misses, generating significantly higher scores. This dynamic highlights that while the offline method may not be consistently superior in general play, it has a distinct advantage on boards where strategic foresight is crucial.

It is also noteworthy that even the random baseline maintains a non-zero win rate against more sophisticated methods. This observation highlights the inherent variability and stochasticity in Candy Crush reward system. On occasion, random moves can inadvertently trigger favorable cascades or special candy alignments, resulting in unexpectedly high scores purely by chance. Such outcomes underscore the game's complexity and the importance of robust strategies that perform well not only on average, but also under the inherent randomness of the environment.

C. Computational Cost Comparison

The time comparison (Figure 6) highlights the significant computational overhead associated with more sophisticated methods. The simplest strategies—random and greedy—complete their computations orders of magnitude faster than the more advanced MCTS-based algorithms. While random and greedy methods only require straightforward evaluations of immediate moves, MCTS variants involve extensive tree expansions, rollouts, or neural network evaluations for each decision.

Among the MCTS methods, the baseline and combined versions are particularly costly, taking up to thousands of times longer than random or greedy strategies. Even the fixed-depth variant of MCTS, which imposes a shallower search horizon, remains substantially slower than reactive methods. However, a carefully tuned MCTS variant we developed achieved a performance level comparable to the standard MCTS approach while being roughly twice as efficient in terms of computation time. This improvement demonstrates that strategic adjustments to search depth, exploration parameters, or value

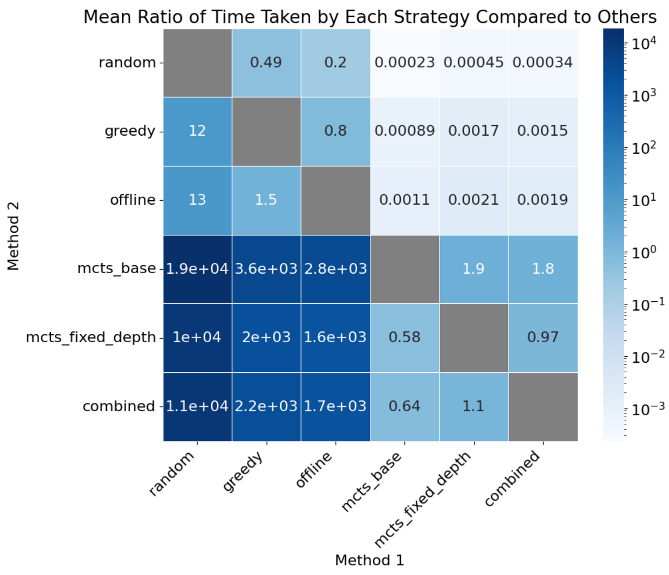

Fig. 6. Mean ratio of computation times between each pair of strategies. Each cell shows how many times longer one method takes compared to another

estimation techniques can yield high-quality decisions at a more manageable computational cost.

D. Comparing with human baseline

The last baseline we compared our strategies against was a human baseline. Players with reasonable background in playing the game recorded their ten moves for 40 randomly generated initial boards. The different strategies described in the previous parts of this report were tested on the same initial 40 boards. Results are presented in a different subsection of this paper because we are unsure if 40 boards are enough to arrive to convergence on estimating the level of each strategies, but these results are still useful to estimate if the different strategies could compete on a similar level than human.
As illustrated in Figure 7, on the 40 boards played by all agents, humans performed better than "simple" strategies (random, greedy, offline), but on a similar level than the three MCTS agents. In particular, the base MCTS performed better than humans, achieving the highest 1st-place rate and the highest mean score. More games should be tested to confirm these results, but this comparison illustrates the power of MCTS search for sequential decision-making in game situations to achieve human-like performances.

VI. Conclusion

This project compared different strategies to achieve the highest possible score in the Candy Crush game. This game exhibits uncertainty in its transition model: random candies appear at the top of the board to fill holes. A basic implementation of the game was constructed. Three baseline were used: a random agent, a greedy agent, and human playing on a subset of game boards. Two online strategies were implemented using Monte-Carlo Tree Search with pruning of some states to limit computational complexity. An offline Convolutionnal Neural

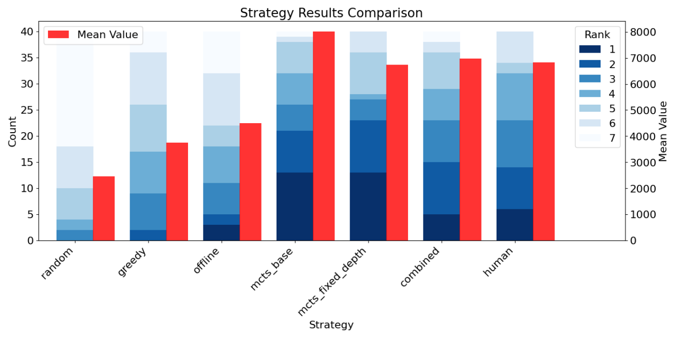

Fig. 7. Comparison with human baseline on 40 randomly generated boards.

Network was trained on the output of MCTS searches to evaluate the potential of boards to generate high scores. MCTS and this offline evaluation method were combined for a last agent.

Results show that all three MCTS agents largely outperform the simpler methods. While these MCTS strategies take longer to run, using a fixed-depth for the MCTS or using the output of the CNN can cut the computation time in half compared to the basic MCTS model. These three methods also perform at a similar level than humans on a small subset of tested boards. In conclusion, this paper asserts the efficiency of MCTS strategies to solve sequential decision-making games with uncertainty. We believe that even higher scores could be obtained by a better training of the CNN to enhance the combined MCTS. For instance, more simulations could be run on the MCTS to provide more precise results as inputs to the CNN. Other functions and structures could also be tested to improve the neural network. Finally, it would be interesting to compare performances of the different agents on specific boards to identify what behaviors can be identified, rather than comparing aggregated mean score data. For instance, we hypothesize that the CNN performs better on high-scoring situations, but fails to distinguish efficiently between low-scoring boards, because these don't exhibit very specific recognizable patterns.

VII. Contriubtions

Both authors made substantial contributions to this project. Antoine focused on implementing and refining the game mechanics, as well as improving the agent with the fixed-depth MCTS variant, while Alexis developed the initial MCTS framework and led the training and integration of the offline CNN-based evaluation. Both authors collaborated closely on the overall analysis of results and redaction of the paper.

All the code can be found on the repository (on the branch Nouvelle_DA): https://github.com/AlexisBnnft/CandyCrusher

References

[1] B. Xia, X. Ye and A. O. M. Abuassbav, "ORecent Research on AI in Games," Limassol, Cyprus, 2020, pp. 505-510.
[2] Lample, G., Chaplot, D. S., " Playing FPS Games with Deep Reinforcement Learning," Proceedings of the AAAI Conference on Artificial Intelligence, 31(1), 2017.
[3] Lukasz Kaiser et al., "Model-Based Reinforcement Learning for Atari," 2024
[4] David Silver et al, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," Science, 362, 1140-1144, 2018.
[5] D. Melhart, J. Togelius, B. Mikkelsen, C. Holmgård and G. N. Yannakakis, "The Ethics of AI in Games," IEEE Transactions on Affective Computing, vol. 15, no. 1, pp. 79-92, Jan.-March 2024.
[6] Mykel J. Kochendefer, Tim A. Wheeler, Kyle H. Wray, "Algorithms for decision-making," MIT press, 2022.