Programming with C/C++

Aleksa Jelic (2024812)

Tilburg University

Instructors: Dimitar Shterionov and Paris Blom

8/1/2020

Table of Contents

Introduction 3
Related work 4
Methodolgy 5
Results 7
Discussion 7
Conclusion 8
Final Thoughts 8
References 9

Introduction

Following project is about creating an AI (Artificial Intelligence) bot that will be capable of playing candy crush-like game, more precisely “match-3” type of game.

When it comes about AI, AI is still controversial topic between many individuals and groups, the views are very different. Based on Legg, S., & Hutter, M. (2007) there are 70 different definitions of Intelligence itself. “Intelligence is the computational part of the ability to achieve goal in the world” (J. McCarthy, 2004 **as cited in** Legg, S., & Hutter, M. 2007). Therefore, creating a bot demands right computational part and its algorithmic implementation.

There are many different domains of AI, in this report automated reasoning in games will be applied. The goal of the bot is to choose the best possible moves and play the game to reach maximum possible score. This does not suggest reaching Alpha Zero levels of AI, but the crucial part is that bot choose matches based on algorithm not randomness.

AI bot could be characterized as self-learning agents that are programmed with NLP and ML which is to some extent correct regarding Joshi, N. (2020, February 24). On the other hand, Baumgarten, R., Colton, S., & Morris, M. (2009) created bot that played DEFCON (nuclear war simulation strategy game) in which AI techniques were used. Such as decision trees learning vase-based reasoning etc. In this project I would identify to DEFCON game where with right programs and algorithms simpler version of AI-bots can be created.

Regarding game in this project, “math-3” is type of game that you as a player or in my case bot needs to find 3 same figures vertically or horizontally next to each other by switching order of figures. When that happens figures or in this case numbers brakes and score is added to player/bot performance. When the broken tiles are broken other comes from the top of the board. The game is done after no more combinations are on the board.

The solution will be focused on 2 algorithms, the first greedy algorithm and depth first search algorithm. How and why those 2 algorithms are implemented will be explained in methods and result sections.

Firstly, I will introduce related topics on this matter following by methodology and results based on scores my bot has accomplished. I will finish of with discussion and conclusion.

Related work

I would relate my work to Baumgarten, R., Colton, S., & Morris, M. (2009) and their DEFCON game. They have used the board to predict best movements for nuclear ships to attack enemy. They have used decision trees which gave me idea to complete my project.

In agent bot games like we differ 2 approaches: server and client-side approach Dignum, F., Westra, J., van Doesburg, W. A., & Harbers, M. (2009). In server-side approach the dynamic cycle of the specialists is typically totally coordinated into the game, bringing about specialists that need to settle on choices inside one-time step of the game circle. Examples of this approach are Quake III, Bos Wars. On the other hand, client-side approach agents are discrete applications utilizing the organization data that is generally shipped off a customer game, examples Game bot connecting to Unreal Tournament 2003, Flexbot connecting agents to Half-life.

In paper Shin, Y., Kim, J., Jin, K., & Kim, Y. B. (2020) states that Monte Carlo tree search and finite-state machine algorithm are good approach for math-3 type of games, but their downside is that when the rules change, they should be modified. Therefore, they have use reinforced learning where their bot could learn form previous tries and adapt to situation and for an instance new rule. I have chosen other 2 algorithms as less complicated then mentioned above.

One more example of math-3 game can be found in Mugrai, L., Silva, F., Holmgård, C., & Togelius, J. (2019, August), where they built agent that imitates a drawn out human player who looks to deliberately upgrade the greatest number of focuses that can be accomplished through a progression of activities after a specific number of moves. As in previous paragraph MCTS algorithm was used.

Methodolgy

Greedy Algorithm

"An algorithm that always takes the best immediate, or local, solution while finding an answer. Greedy algorithms find the overall, or globally, optimal solution for some optimization problems, but may find less-than-optimal solutions for some instances of other problems (Paul E. Black, 2005)."

Since the greedy algorithm is supposed to be optimized locally, I set up a simple test, where I visited each point in the map and test for the four moves for that position, if they were possible. I tested first for vertical paths, to see how long they would be and retained the longest.

I then checked every position in the map again and determine how long the horizontal paths would be. I retained the longest path from both searches combined and returned that value.

The scores for both algorithm will be discussed in results section.

Depth First Search Algorithm

The depth first search is a graph search algorithm that allows you to look at every branch of a path. So, I thought this would be a way to determine the full length of a path. This should also, I believe, provide the optimal value for the map in many or most cases. It looks at every cell of the map and swaps the four elements (above, below, right and left) and does a depth first search for each. So, that is 4 * 100 searches for the full map. I also make sure before doing a depth first search that there is a vertical or horizontal path of at least length three, since the depth first serach doesn't check for this.

There could be cases where this fails to find the optimal value, because it counts sub-branches of length one. I might be able to add more optimizations and eliminate the flaw. To do the search I had to create a node structure that had two variables, one for the map value and one for whether the table was visited, Example (1):

```
struct node {
  int value;  // The value in the table
  bool visited; // This is for depth first search to marke a node as visited, so that it is only visited once.
};
```

Overall, the depth first search follows a path and all its branches. What I do is put a counter in to count the path length and it returns the total length. It won't tell if there are three in a row, with each x,y element, so before feeding an element into the depth first search algorithm, to make certain there are three in a row, I check with a few functions created for the greedy algorithm. I believe this is an optimal search strategy for this game.

Results

Result are straight forward. Regarding Greedy Algorithm are for map10.txt I achieved a score of 236. For map5.txt I achieved a score of 476. It is obvious that algorithm works better on smaller size maps.

On the other hand, expectedly DFS did better. For map10.txt I achieved a score of 382, and for map5.txt I achieved a score of 481. Both are improvements over the greedy algorithm.

Discussion

The bot performed well at some degree. There are no any errors and it gives pretty good score, unfortunately I was not capable enough in creating more advanced algorithm, but it did its job.

Regarding Greedy algorithm I wrote does not consider anything but straight lines. It doesn't deal with T shapes and L shapes, etc. This means it will miss a lot of better choices. To follow those paths would have required a smarter algorithm. To improve this, I could have tested for off shoots of the pattern of two or more from the line I followed. Therefore, this is the main reason for scoring lower than the other algorithm.

Better DFS, allows me to follow all the paths that a pattern makes in an efficient way. The drawback of the algorithm is that it doesn't count straight lines of three or more. So, to improve this algorithm I'd have to add that feature to it.

Much better improvement for overall algorithms would be looking 2 or more moves ahead, with that said bot would be able to have bigger picture to find better moves for upcoming moves.

Conclusion

The Depth First Search did slightly better for map5.txt and significantly better for map10.txt. This makes sense, since Depth First Search is more exhaustive and complete in checking paths. It finds the whole path. It really only needs to be tweaked for determine whether an offshoot is two units or more long. If that is added, this should be a globally optimum solution. Even as it is set up now, it is more likely to find a globally optimum than the Greedy algorithm. By contrast, the Greedy algorithm just looks at the straight lines up/down, left/right. It is going to miss a lot of information for this reason. Despite this, it didn't do badly in scoring.

Final Thoughts

I would like to acknowledge how interesting and challenging this project was, and how much I learnt from single project. I was not able to do completed algorithms like MCTS or FSM that would base on research bring my score up, but anyway I did my best.

For my own reference I will try to implement this project with some sort of user interface in the future.

References

Legg, S., & Hutter, M. (2007). A collection of definitions of intelligence. Frontiers in Artificial Intelligence and applications, 157, 17.\

Joshi, N. (2020, February 24). Choosing Between Rule-Based Bots And AI Bots. Retrieved January 08, 2021, from https://www.forbes.com/sites/cognitiveworld/2020/02/23/choosing-between-rule-based-bots-and-ai-bots/?sh=140767b0353d

Baumgarten, R., Colton, S., & Morris, M. (2009). Combining AI methods for learning bots in a real-time strategy game. International Journal of Computer Games Technology, 2009.

Dignum, F., Westra, J., van Doesburg, W. A., & Harbers, M. (2009). Games and agents: Designing intelligent gameplay. International Journal of Computer Games Technology, 2009.

Shin, Y., Kim, J., Jin, K., & Kim, Y. B. (2020). Playtesting in Match 3 Game Using Strategic Plays via Reinforcement Learning. IEEE Access, 8, 51593-51600.

Mugrai, L., Silva, F., Holmgård, C., & Togelius, J. (2019, August). Automated playtesting of matching tile games. In 2019 IEEE Conference on Games (CoG) (pp. 1-7). IEEE.

Paul E. Black, "greedy algorithm", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 2 February 2005. (accessed TODAY) Available from: https://www.nist.gov/dads/HTML/greedyalgo.html