TER Report
Takenoko

Game engine

For this study, we took the last version of the game we developed in the project/software engineering course and slightly tweaked the game engine to suit our needs. The games begin with a full board, either filled randomly (while still using the correct way to place tiles) or loaded from a plain text description. A few randomly selected tiles grow a bamboo shoot of a given size which never grows (not even after being eaten) so as to have enough bamboo to complete all the panda objectives in the deck. Each turn, the player picks as many objectives as he need to have a full hand. The game ends when the player has a chosen number of points.

Guaranteed AI

The first AI we wanted to implement was one that views the game board as a graph and finds the most efficient path to gather the bamboos it needs to complete its panda objectives.

First, we compute the bamboos we need to eat, according to an algorithm that can be refined to give better and better results. The one we have now sorts the objectives and find how many bamboo of each color we need to complete them all. We'll then always try to complete the most rewarding one.

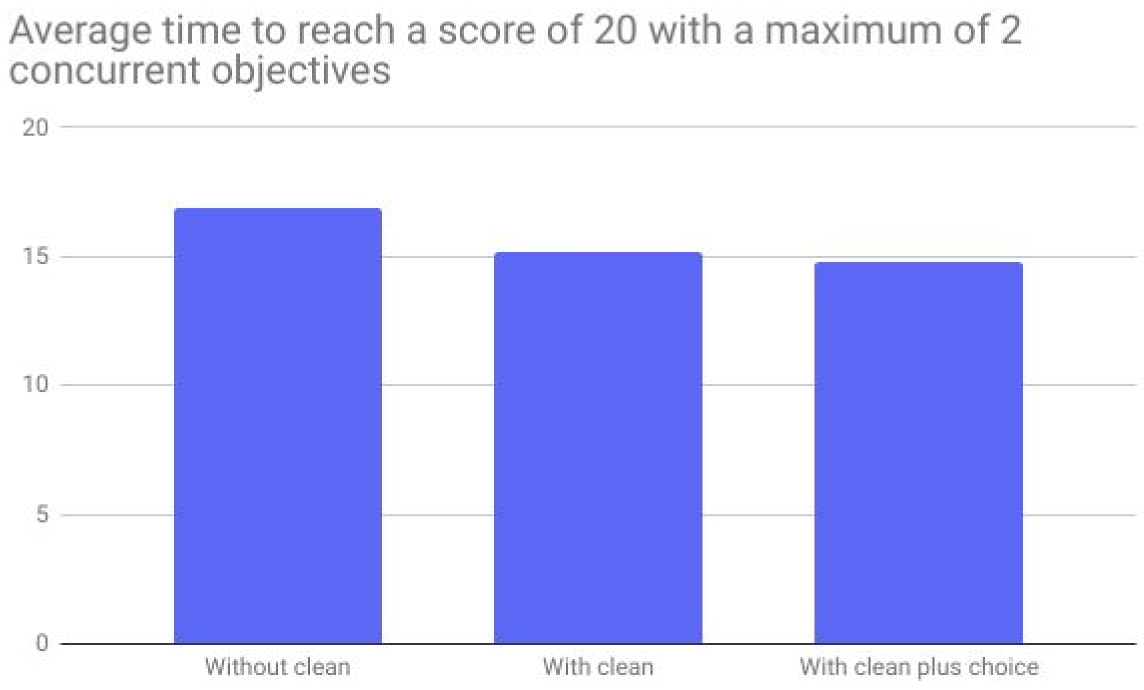
The second step is do determine where to move to acquire the right bamboos. The AI we have eat them sequentially, in the order we computed earlier, the goal being completing as quickly as possible the best objectives we have. We just look for the best path to reach a tile with a bamboo of the right colour. To do so, the AI computes a graph wherein all the vertices are tiles that have grown some bamboo, plus the tile on which the panda is, with an edge linking every vertex to all the others. For each edge *(a, b)*, a weight corresponding to the length of the shortest path on the board that goes from *a* to *b* is computed using an implementation of the breadth-first search algorithm. As there can be many paths of the same length from one tile to another on a given board, we wanted to make sure that any intermediate step needed for the panda to reach a given tile would be a tile with bamboo if possible.

To ensure that behaviour, we implemented an algorithm to remove all the edges not useful to shorten a path between its two vertices. This algorithm goes through each edge, and compute the shortest path between its vertices if it didn't exist anymore. If the path is longer, the edge is re-introduced in the graph, else it is kept out.

The resulting graph is thus a single strongly connected component, and any edge that has a weight superior to 1 means that there is no way to go from *a* to *b* without passing on a tile that has no bamboo on it.

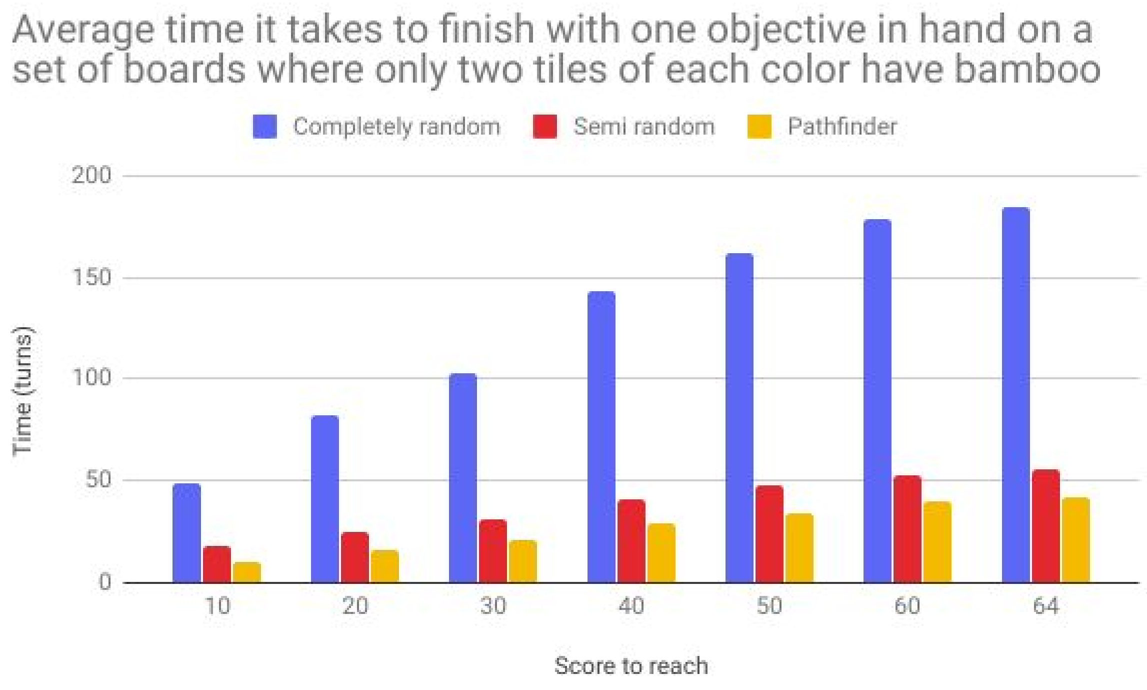
Then, we use Dijkstra's shortest path algorithm to compute the most efficient way to gather all the needed bamboos and remember that path. Now, each turn the bot will make the panda move to the next place on that path. If there are no more steps to follow, we compute a whole new path (sorting the objectives, computing the graph, etc…).

We noticed that even with our graph reduction, in some specific cases we had to choose between paths of the same length but not offering the same amount of bamboos. We introduced a way to determine which path to choose in this case, and this improvement brought again a slight reduction of the average time for a bot to reach the maximum score. This is shown in the graphic below.

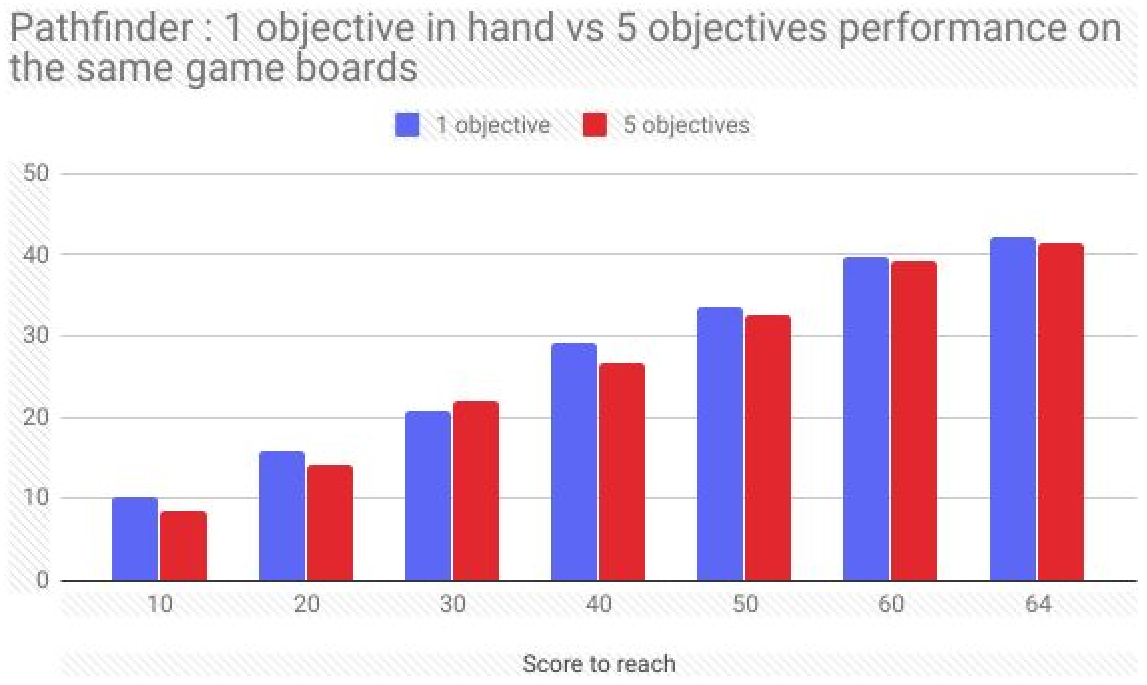


Results

We looked at three bots : our pathfinder bot, a completely random one which moves the panda anywhere it can, and a semi random one that moves the panda to any tile that has bamboo. The first two are always able to finish a game, but the third one will sometimes end up stuck moving between the two same tiles which is a waste of time. Here, we compare the time taken by each bot to reach scores ranging from 10 to 64 (64 being the maximum possible score). With always a single objective in hand, this is what we get :



As we can see, the pathfinder is consistently faster than the other two, with the semi-random bot following close behind and the completely random one way slower. This proves that our path-finding algorithm works better than just haphazardly going where the bamboo is.



The graph above show us that the pathfinder bot performs better when it has more objective in hand. This can be explained by the fact that it is able to complete more objectives at the same time.

Ambitious AI

The point of the second AI was to be able to make the game interesting for more than one player. Indeed, making the first bot play against itself would be nothing more than a game of tug of war where they both move the panda to where they need it to be.

The game now starts with a filled board as explained precedently except no bamboo is put anywhere. Thus, this bot not only uses the panda to eat the bamboo, but also the gardener to grow it. The goal is to implement a minimax algorithm. The player would try to move the gardener to where it can grow bamboos of as many colors as possible, while using the minimax to make that choice be the least favorable to the other player.

Conclusion

The first bot is, as expected, a lot more efficient than the random one. However, it is only slightly - although consistently - better on average than the semi-random one. Unfortunately, we did not have enough time to implement our second AI, but we expect it to be a definite improvement compared to the first bot, and an even greater one in regards to the random bots.