Joustus and the Expectiminimax Algorithm

Nathan Stoner, Zachary Hughes, Mark Brestensky

COMP 445

# **Introduction**

In 2014, the hit video-game “Shovel Knight” was released to high acclaim. In December of 2019, the game received a new campaign called “King of Cards,” and it introduced players to the game “Joustus”. Joustus is a unique card game played by two opposing players, . and the objective of the game is to collect gems that are scattered on a board of varying sizes [1]. Players can place cards on the board or push any card already on the board into a certain direction [1]. Players cannot place their cards on the gems to collect it; they must push one of their own cards onto the gem to collect it [1]. Each card can also push in a certain number of directions (those being up, down, left, and right) [1]. Even an opponent’s cards can be pushed around the board, and even to their final resting place in the board’s graveyard [1]. The game goes until all open board spaces are filled, and whoever has the most gems wins the match [1]. Players cannot pass their turn so if a move can be made, it must be made, even if it is disadvantageous [1]. Joustus is a game built for a single player, and as such has an intelligent AI system to give players a challenge. To see two of these AI systems compete head to head would be an extremely interesting match. . However, Joustus is not deterministic, which is a requirement of minimax. Though, Both players have the same amount of information and the game itself is zero-sum, but the non-deterministic factor would complicate making a minimax AI system to play the game.. As such, new methods and changes in the game would be required in order to simulate an AI game of Joustus. The method our team decided would be best for our Joustus AI is the implementation of Expectiminimax.

# **Expectiminimax**

In Joustus, the players have the same amount of information present to them. A player knows what cards are in their hand and what cards are in their opponent’s hand. Players also know what cards are in their deck, but they do not know the draw-order, so they have no information on what could be drawn next. This lack of perfect information means that the standard minimax algorithm cannot be applied for a game of Joustus. The AI system we create needs a way to account for random chance in their card draws and their opponent’s card draws. This is why we chose to utilize expectiminimax, a decision-making algorithm that takes chance into account for playing games such as this.

## *a. The Algorithm*

The expectiminimax algorithm is simply the minimax algorithm for games that have a degree of chance in them [2]. It contains minimum and maximum nodes, just like a minimax decision tree [2]. However, expectiminimax has chance nodes in its decision tree as well [2]. These chance nodes calculate the probability of all children that branch from it [2]. For example, in a game of Backgammon, the chance nodes would calculate the probability of each possible roll of the dice. From there, the algorithm will calculate the expected value, which is the average over all possible outcomes of the chance node. Because of the expected value, the positions don’t have definite minimax values [2]. This turns the minimax values into expectiminimax values [2]. The terminal nodes and minimax nodes work the same as in minimax, but for chance nodes, the expected value is calculated and weighed by the probability of each chance action [2]. As a result, the algorithm will tend to look more down the branches of chance nodes that have a higher chance of happening in the game [2]. This allows for a decision tree that takes into account chance.

## *b. Drawbacks*

One of the chief drawbacks of the expectiminimax algorithm compared to the minimax algorithm is the fact that the expectiminimax algorithm has a much longer run-time [3]. Minimax has a run-time of about O(b^m), where *b* is the branching factor and *m* is the maximum depth of the tree [3]. Expectiminimax, due to the fact that it is considering all possible chances in the game and their probabilities, has a runtime of about O(b^m n^m), where *n* is the distinct number of possibilities that can be obtained by chance [3]. This makes expectiminimax have a longer computational time than general minimax [3]. A fix to this is to simply limit the depth of how far expectiminimax is to search at any given point, creating what is called Cut-Expectiminimax, where the search-tree of expectiminimax is cut at some depth *d* [2]. However, even with that depth limit, the number of computations needed to be done could be much higher compared to minimax. As a result, *d* is usually very small. For example, Backgammon only searches three deep at most when this algorithm is applied to it [3]. Alpha-beta pruning also can’t be utilized efficiently since it relies on the idea of moves it knows for certain cannot occur [3]. However, in a game that has chance, alpha-beta has no idea if any moves are legal, only knowing once a piece of information from chance is revealed that makes certain moves legal, and thus cannot be applied [3]. Yet, for a game of chance, this is the only algorithm that can be utilized to effectively handle such games.

## *c. Hanafuda and Expectiminimax*

Another card game that has utilized expectiminimax was the game Hanafuda [4]. In Hanafuda, there is a deck of 48 cards divided into twelve families with five different types, each type giving a certain score [4]. The cards are divided into four groups, two groups of eight for the players, one group of eight for the table, and one group of 24 in a “stock” [4]. The player and table cards are known and visible, the stock cards are not [4]. Players try to match cards in their hand with those on the table if both cards are of the same family, or they add a card to the table. They then draw a card from the stock [4]. Expectiminimax is perfect for a game such as hanafuda, a card game with an element of chance [4]. In an experiment with utilizing expectiminimax on AI playing hanafuda, analysts found that expectiminimax cut to a depth of two won the majority of games compared to a randomizer algorithm and a greedy algorithm (and that the score of games it lost had a lower difference than the scores of games it won) [4]. Expectiminimax cut to a depth of three performed slightly better against the greedy algorithm [4]. As such, the analysts concluded that expectiminimax was the better algorithm in most cases for utilization in card games like hanafuda; card games with chance [4]. This is what drove us to implement expectimimax in our Joustus algorithm, since Joustus and hanafuda are both card games with a set goal where the only chance elements are found in card draws.

# **Works Cited**

[1] “Yacht Club Games,” Joustus Instruction Manual | Yacht Club Games. [Online]. Available: https://yachtclubgames.com/joustus-instruction-manual/. [Accessed: 10-Apr-2020].

[2] E. Melko and B. Nagy, “Optimal strategy in games with chance nodes,” Acta Cybernetica, no. 18, pp. 171–192, 2007.

[3] S. J. Russell and P. Norvig, Artificial intelligence: a modern approach. Upper Saddle River: Prentice-Hall, 2010.

[4] P. A. Ortega and D. A. Braun, “Free Energy and the Generalized Optimality Equations for Sequential Decision Making,” JMLR: Workshop and Conference Proceedings, vol. 10, May 2012.

[5] A. Cruz, L. Preuss, U. Souza, S. Serique, A. Ogasawara, E. Bezerra, and E. Ogasawara, “Amê: An Environment to Learn and Analyze Adversarial Search Algorithms Using Stochastic Card Games.”