# BetaSplendor One

An AI For Gem Dominance in the Tudor Period

Cary FitzHugh

ETSU



## Project

This project will be to build and train an AI system which can play the board game Splendor using techniques explored in the AlphGoZero[[1]](#endnote-2) paper.

## 

## About the Game

Splendor is a board game played by 2 - 4 players. It is an engine-building and resource management game where players attempt to collect the most prestige points.

Prestige points are obtained through buying cards and by gaining a noble’s visit.

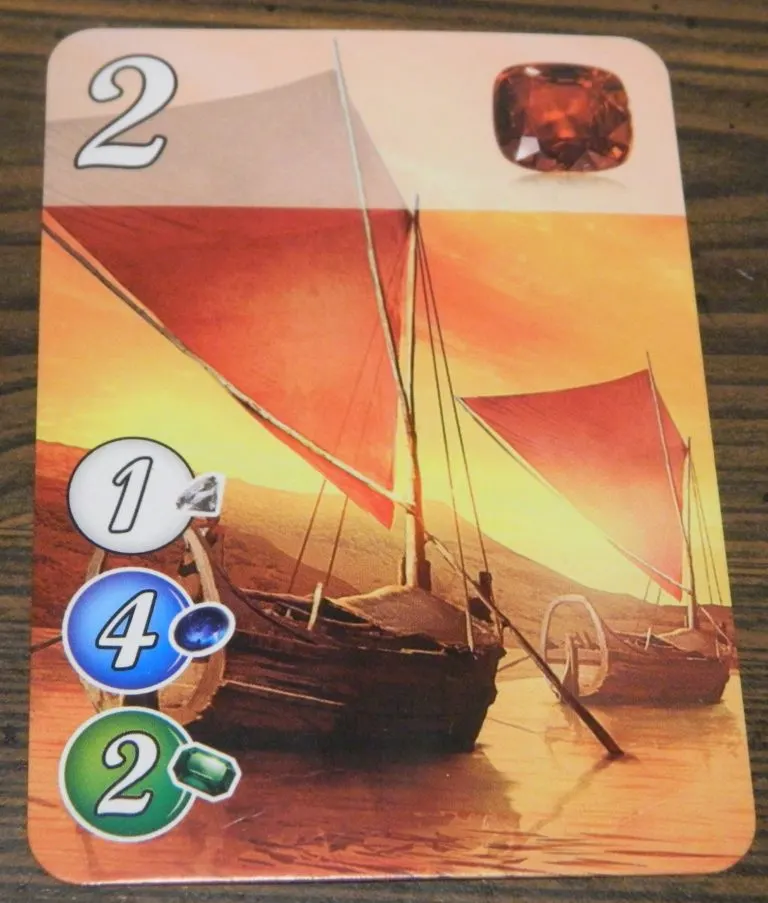
### Gems

Core to the game are 5 types of gems: Emeralds, Diamonds, Sapphires, Rubies, and Onyx(en?).

In addition, there are Gold tokens which can be used as a replacement for any gem (a wild-card).

### Development Cards

Development cards are the main game mechanic to grow your purchasing engine and obtain prestige points.

In this example, the development card provides 2 prestige points (top left).

It provides a gem bonus of one Ruby (top right).

Development cards provide a bonus of one gem of a certain type. Purchased cards provide a bonus for future purchases. e.g. If a card has a one emerald bonus, in subsequent turns you will have a “free” emerald to spend buying additional cards.

This card costs 1 Diamond, 4 Sapphire, and 2 Emerald to purchase.

Development cards are purchased using gems available in the player’s bank, the gold available, and the sum of all bonuses on all purchased cards.

The cards are split into 3 decks or tiers. Lower tier cards are cheaper, and provide few (if any) prestige. Higher tier cards are more expensive, but provide correspondingly higher prestige points.

### Nobles

Nobles are extra prestige point bonuses bestowed on a player when they buy certain patterns of development cards. Nobles only give prestige point bonuses. They are automatically provided to a player once their turn is over and they qualify for a nobles’ “visit”.

This example noble “visits” a player when that player has purchased 3 development cards with an emerald bonus, 3 with a sapphire bonus, and 3 with a diamond bonus. It provides 3 prestige points.

### Turns

Each turn a player can do one of the following each turn:

* take gems (if available) from the bank
* buy a development card
* reserve a card

#### Taking Gems

You can take up to three gems in a turn (if they are available in the bank). The gems must be of different types, though you can take 2 of a single type in some situations. At the end of your turn you need to return gems if you have more than 10 in your possession.

#### Development Cards

Each deck of development cards has the top 4 cards flipped over. You can buy these cards directly from the tabletop if you have enough gems + bonus.

You may also purchase a development card which you reserved previously.

#### Reserving Cards

You may reserve up to 3 cards from the table to your hand. Once reserved, you can purchase them like you would purchase other development cards. When reserving a card you can reserve a face-up card or you can pick from the top of the deck. If you pick a face-down card, only you can see it until it is purchased. When reserving you also get a piece of gold, which is a wild-card gem.

### Tabletop View

On the table, there are 5 nobles visible (3 for a 2 player game, and 4 for a 3 player game).

The top 4 development cards in each tier are visible as well.

Every player’s gems and development cards ***must*** be visible to all players at all times. The only exception is if a player reserves a card from the top of a tier deck, in which case only that player can see it until it has been purchased.



### End Conditions

The game is over when any player gets over 15 prestige points. If a player gets to 15 points, the round will continue (giving all players a chance to have the same number of turns). The winner is the player with the most points, ties go to the player with the fewest cards.

## AI Approach

|  |  |
| --- | --- |
| Property | Splendor |
| Partially Observable | *Players can see everything except what is in the decks (the order of cards coming up) and any hidden-reserved cards that other players have.* |
| Multi-Agent | *Many players are playing against each other, actively seeking to win.* |
| Deterministic\* | *Each action selected produces a deterministic result (only reserving cards from the top of a deck will have any amount of chance involved – they are not fully observable)* |
| Episodic | *The game builds upon previous actions each player takes.* |
| Static | *Each player gets the time to make their decision, the game board does not change while deliberating.* |
| Discrete | *Actions and outcomes are all discrete* |
| Known | *We know all the rules of the game beforehand.* |

### Search Tree

The game’s search tree is moderately complex.

Exhaustive searching of the tree space is prohibitive. A Splendor game can have upwards of 50 turns (each with potentially 4 phases for each player) This gives around 200 branching nodes for a given game. At each branch there are a number of valid move options (see figure).

In the beginning all 15 gem choice (PICK\_\*) actions are available, as well as all 15 reservation (RESERVE\_\*) actions.

As the game progresses, PICK\_\* actions become limited by gem availability. RESERVE\_\* actions stay roughly constant until cards begin to run out. The 15 BUY\_\* actions are least likely.

Worst case there could be 40+ actions available to a player. This gives an exhaustive search space of O(*turnsactions).* As an example, a 20 turn game, of 4 players, with a somewhat worst-case of 40 actions available per phase, gives an exhaustive search tree of

*4080*

Due to the size of this exhaustive search tree, other alternatives are needed. AlphaGoZero utilizes MonteCarloTreeSearch as a way to evaluate game states with extremely large search trees. It also uses a Neural Network to pick the next moves – foregoing exhaustive search tree state navigation and the time/size complexity involved. In Splendor there is also no utility function for a state, so techniques for pruning nodes will not be effective..

AlphaGoZero also does not use any human-created training logic. This is good because in the case of BetaSpledorZero (BSZ) we do not have access to any Splendor players with any real measure of skill.

BSZ will use the same approach as AGZ and use Reinforcement Learning with Deep Neural Networks, known as DQN (Deep Q Networks).

### Available Actions For BetaSplendorZero

#### Pick Gem Options

PICK\_DSE

PICK\_DSR

PICK\_DSO

PICK\_DER

PICK\_DEO

PICK\_DRO

PICK\_SER

PICK\_SEO

PICK\_SRO

PICK\_ERO

PICK\_DD

PICK\_SS

PICK\_EE

PICK\_RR

PICK\_OO

Total(15)

#### Buy Card Options

BUY\_TIER\_0\_0

BUY\_TIER\_0\_1

BUY\_TIER\_0\_2

BUY\_TIER\_0\_3

BUY\_TIER\_1\_0

BUY\_TIER\_1\_1

BUY\_TIER\_1\_2

BUY\_TIER\_1\_3

BUY\_TIER\_2\_0

BUY\_TIER\_2\_1

BUY\_TIER\_2\_2

BUY\_TIER\_2\_3

BUY\_RESERVED\_0

BUY\_RESERVED\_1

BUY\_RESERVED\_2

Total(15)

#### Reserve Cards

RESERVE\_TIER\_0

RESERVE\_TIER\_0\_0

RESERVE\_TIER\_0\_1

RESERVE\_TIER\_0\_2

RESERVE\_TIER\_0\_3

RESERVE\_TIER\_1

RESERVE\_TIER\_1\_0

RESERVE\_TIER\_1\_1

RESERVE\_TIER\_1\_2

RESERVE\_TIER\_1\_3

RESERVE\_TIER\_2

RESERVE\_TIER\_2\_0

RESERVE\_TIER\_2\_1

RESERVE\_TIER\_2\_2

RESERVE\_TIER\_2\_3

Total(15)

## Method

BetaSplendorZero will mimic much of the structure for AlphaGo Zero. One difference is that Go only models 2 players whereas Splendor will have 4 player networks competing for dominance. Following from that difference, there will be an adjustment to the step where new networks are pitted against the current best network after an episode is complete. Since the network is trained to be part of a game of 4 players, it may not be appropriate to determine a winner based on a head-to-head matchup with the “best so far” network.

A model has been written[[2]](#endnote-3) where the code mimics functional style where enumerated actions are passed into a state-change function, which returns the new table state.

The overall pseudo-code for the training algorithm follows:

### Pseudo-Code

best\_network = **RandomNetwork**()

def **MonteCarloTreeSearch**():

# Uses tree search and propagates either the NN

# evaluation of the node or a -1/1 for loss/victory

def **Train**():

# Attempts to train a network with it's history of moves

# if won reinforce, if lost, diminish

def **PitNetworks**():

# Return a set of networks, replacing a player with the

# best\_network so far.

for episode\_number in range(**NUMBER\_OF\_EPISODES\_TO\_TRAIN**):

episode\_networks = best\_network[**NumberOfPlayers**]

for game\_number in range(**NUMBER\_OF\_GAMES\_PER\_EPISODE**):

game\_state = **NewGame**()

while **NotFinished**(game\_state):

for player in **Players**():

p\_next\_move = **MonteCarloTreeSearch**(

game\_state,

iterations=1600,

neural\_net=episode\_network[player],

temperature=**TempFunction**(episode\_number))

game\_state = **MakeMove**(game\_state,

**SelectNextMove**(p\_winning, p\_next\_move))

winner = **Winner**(game\_state)

# Train networks

for player in Players():

episode\_networks[player] = **Train**(

episode\_networks[player],

**GetPlayerMoves**(game\_state, player),

weight=(player == winner ? 1 : -1))

# Compete!

wins = int[Players() + 1]

for player in **Players**():

for pit\_game in range(PIT\_GAMES):

networks = **PitNetworks**(

episode\_networks

best\_network,

player)

winner = **PlayGame**(networks)

wins = **MarkWinner**(winner, player)

if player == winner:

# If one performed measurably better than the previous best

best = max(mean(wins))

if (best - mean(wins)) > (PIT\_GAMES \* Players() \* WIN\_PCT\_THRESHOLD):

best\_network = best

### Network

A neural network will be constructed with inputs for each part of the table state. I plan on providing the entire table state to the network’s inputs, each one as an integer.

#### Inputs

bank (6) - [G, D, S, E, R, O)

4 x noble (6) - [prestige, D, S, E, R, O]

12 x tier\_X\_X (11) [prestige, Bonus (D, S, E, R, O), Cost (D, S, E, R, O)]

player (45) (x2, or 3 or 4)

bank (6) - [G, D, S, E, R, O)

bonus (5) - [D, S, E, R, O]

points (1)

reserved\_card\_0 (11) [prestige, Bonus (D, S, E, R, O), Cost (D, S, E, R, O)]

reserved\_card\_1 (11) [prestige, Bonus (D, S, E, R, O), Cost (D, S, E, R, O)]

reserved\_card\_2 (11) [prestige, Bonus (D, S, E, R, O), Cost (D, S, E, R, O)]

The inputs are a total of up to 342 discrete 8-bit values in a 4 player game (2 player is 252 values).

#### Outputs

The output nodes are mapped to probability preferences for actions mentioned earlier, as well as:

* one value for the predicted % chance the current board state will win for a player
* 5 values which are preferences for nobles if they try to visit this turn
* 5 values showing preference for different gem types (in the case that too many are retrieved, these preferences will let the system return the least desired gems until there are < 10 left).

## Conclusion

The BetaSplendorZero system takes much of the AlphaGoZero approach, with modifications to handle more than two players.

The project will be my first introduction to designing and training a neural network using Reinforcement Learning. Can a reasonably competent AI be created for this game simply by playing itself over and over in a way similar to how AlphaGoZero learned the 2-player game of chess?

If the AI gets good enough I would enjoy learning some alternative strategies.

## Prior Work

In surveying previous approaches there were a few papers and code which approached this game and tried to teach an AI to play it.

### Lapidary-AI

https://github.com/inclement/lapidary-ai/

It uses a simple neural network, with a few more rule simplifications than my AI will have.

### Splendor\_AI

https://github.com/mcandocia/splendor\_ai

https://maxcandocia.com/article/2018/May/04/reinforcement-learning-for-splendor/

This uses a neural network which is more advanced than lapidary-ai’s implementation. It uses fewer inputs than my network will have - as well as adding things which I feel is superflous data (tier of a card, etc).

### Commercial AI

https://www.daysofwonder.com/online/en/splendor/compendium/#AI

The commercial AI for splendor breaks down into 3 classes of AI: Balanced, Opportunistic, and Specialized.

### Rinascimento

https://paperswithcode.com/paper/rinascimento-searching-the-behaviour-space-of

Uses behavioral space mapping and techniques using MAP-Elites - to enable state space evaluations. Does not use neural networks.

1. Silver, D., Schrittwieser, J., Simonyan, K. *et al.* Mastering the game of Go without human knowledge. *Nature* **550**, 354–359 (2017). https://doi.org/10.1038/nature24270 [↑](#endnote-ref-2)
2. GitHub fitzhugh-etsu/splendor [↑](#endnote-ref-3)