# Project Proposal

### CSCI 5260

Cary FitzHugh

## Project

This project will be to build and train an AI system which can play the board game splendor.

## 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.

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

### Turns

Each turn a player can take gems (if available) from the bank, buy a development card, or 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

Development cards have a gem cost to purchase, may or may not provide prestige points, and also provide a single gem ‘bonus’ once purchased. 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.

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.

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.

#### 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.

### Nobles

At the end of each turn, nobles come into play. You qualify for a noble when it’s development card requirements are met (e.g. a noble might require you to have purchased 3 ruby and 3 emerald development cards). Only one noble can visit you each turn.

## Approach

The game is mostly-observable (except for reserving cards from the deck).

It is multi-agent (2-4 players) who compete against each other, but it is not necessarily zero-sum.

It is deterministic.

It is sequential, since each move changes the board for subsequent moves.

The game is static, so an AI can take it’s time to determine it’s next move.

It is discrete, and the game is fully known.

The game has a moderately complex search tree. At the beginning of play, there are 15 different combinations of gems to take, as well as another 14 options for reserving various cards. As the game progresses, there could be an additional set of 15 choices involving which cards on the table (or reserved) you would like to purchase.

Because of this search tree size, graph search strategies will be avoided. AlphaGo is an AI which was used to play Go - another game with high branching factors. AlphaGo’s approach was to use Reinforcement Learning to learn how to play Go, and then play against itself over and over to get better and better. I will approach this project in a similar manner.

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 a float value. Each player has about 30 state values (bonus counts for each type of gem, # of points, cost and points for each reserved card, etc). It appears that the number of inputs to the network will most likely be close to 200 inputs.

The output nodes are mapped to each of the roughly 50 possible actions that a player could take each turn.

A trained DQN will be provided the input state of the board, and it will calculate it’s outputs. Only valid moves will be kept, and then the actions will be normalized probabilistically, and a choice made for which move to perform.

I plan on having 2 more DQN networks which will handle two unique cases during the game: what gems to return if you have more than 10 and which noble to accept if there is more than one willing to visit.

All of the DQN networks will be trained by playing competetive matches against itself over and over.

## Data Set Overview

Since the plan is to use Reinforcement Learning with Deep Neural Networks (DQN) - there is no existing data set. The system will follow in the footsteps of AlphaGo and play against itself continually using Reinforcement Learning to improve.

## 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.

## Expected Learning

The project will be my first introduction to designing and training a neural network using Reinfocrment Learning. Can a reasonably competant AI be created for this game simply by playing itself over and over. If the AI gets good enough I would enjoy learning some alternative strategies.