# Project Summary

League of Legends: Teamfight Tactics (TFT) is a strategy game that pits the player against seven opponents in a race to build a powerful team to fight on their behalf. The goal is to build the best team composition using nine champions, each of which belongs to a class. Any additional champion increases the team score, however when certain criteria are met, the score is increased even more. When players collect multiple of the same champion, their team score increases by the most. In addition, certain champions are worth more than others, so collecting higher priority champions will also increase your team score by more than lower priority champions.

This project uses a simplified version of the TFT model. There will be a max of 6 of the same champions for the player and the complex system where champions level up with be simplified to a 6/4/2 of a kind system.

You will win against your opponent(s) based on this checklist:

* 6 of a kind
* 4 of a kind
* 2 of a kind
* Single

If there is a tie, you will win if you have the highest 6/4/2/1 of a kind.

Champions are denoted with letters. The champion of the highest value is A, J is the champion with the lowest value.

Champion values: A > B > C > D > E > F > G > H > J

The constraints of the model is that the player will have between 1-5 copies of any of the A-J champions.

There will be a board with that has 4 champions and can contain any amounts of any of the A-J champions. If there are two champion As on the board, then either of the As are suitable to take.

The goal is to get the champion that provides the best value. If we have 5 champions of type A, and there is an available A on the board, then we will get it. If we have 4 champions of type A, 1 champion of type B, then we will get the champion of type B because “five-of-a-kind” gives the same points as “one-of-a-kind”, and “two-of-a-kind” gives better points than “one-of-a-kind”. Finally, if we have 2 champions of type A and 2 champions of type B, we will get the champion of type A because the value of A is defined to be greater than the value of B.

This simplified version of Teamfight Tactics is very similar to the concept of poker hands where the “4 of a kind” beats the "3 of a kind”, and the “3 of a kind” beats the “two of a kind, which beats the single.

# Propositions

N\_n represents the nth copy of champion N that the player owns.

A\_1 is true when the player has one copy of the A champion

A\_2 is true when the player has two copies of the A champion, A\_1 will also be true

A\_n is true when the player has n copies of the A champion, A\_1, A\_2, …, A\_n-1 will also be true

B\_1 is true when the player has one copy of the B champion

B\_2 is true when the player has two copies of the B champion, B\_1 will also be true

B\_n is true when the player has n copies of the B champion, B\_1, B\_2, …, B\_n-1 will also be true

…

N\_1 is true when the player has one copy of the N champion

N\_2 is true when the player has two copies of the N champion, N\_1 will also be true

N\_n is true when the player has n copies of the N champion, N\_1, N\_2, …, N\_n-1 will also be true

N\_b represents that champion N is available for purchase from the board.

A\_b is true when the A champion is available to be bought from the board

B\_b is true when the B champion is available to be bought from the board

…

J\_b is true when the J champion is available to be bought from the board

For the purposes of this model, the letters will only be A , B , C , D , E , F , G , H , J and the number of each letter can only go up to 6. The reason why A\_6 is the highest value is because A\_5 is the highest initial value, and you can still add an A from the board, so A\_6 is the highest value you can make. The constraints are done in A\_6 because we want to do constraints based on the highest value we can make.

# Constraints

Owning n champions of a specific type means that you also own the the n-1, n-2, …, 1st champion of that specific type.

If A\_2 is true, A\_1 must also be true.

A\_2 -> (A\_2 /\ A\_1)

If A\_6 is true, A\_1, A\_2, …, A\_5 will also be true.

A\_6 -> (A\_6 /\ A\_5 /\ … /\ A\_1)

…

If J\_2 is true, J\_1 must also be true.

J\_2 -> (J\_2 /\ J\_1)

If J\_6 is true, J\_1, J\_2, …, J\_5 will also be true.

A\_6 -> (J\_6 /\ J\_5 /\ … /\ J\_1)

Constraint to determine the class that cannot be chosen (cannot be chosen from the board)

This constraint is added to increase the complexity of the model. A champion of a specific class will not be able to be chosen if the player also owns champions from selected other classes.

(A /\ B) -> ~C

(A /\ C) -> ~B

(B /\ C) -> ~A

(D /\ E) -> ~F

(D /\ F) -> ~E

(E /\ F) ->~D

(G /\ H) -> ~J

(G /\ J) -> ~H

(H /\ J) -> ~G

Constraint to determine higher valued champion (higher alphabetical order is higher valued)

(H /\ J) -> H

(G /\ H /\ J) -> G

(A /\ B /\ C /\ D /\ E /\ F /\ G /\ H /\ J) -> A

Example with a declared proposition: (B\_4 /\ C\_4) -> B\_4

Constraint to determine 6 of a kind:

(A\_5 /\ B\_5 /\ C\_4) -> (A\_5 /\ B\_5)

Constraint to determine 4 of a kind:

(A\_3 /\ B\_3 /\ C\_2) -> (A\_3 /\ B\_3)

Constraint to determine 2 of a kind:

(A\_1 /\ B\_1 /\ C\_4) -> (A\_1 /\ B\_1)

\*\*Notice how adding a C will only get a C\_5, which will add only “one-of-a-kind” points because you need C\_6 to get the “six-of-a-kind” points. Since “two-of-a-kind” points are higher than “one-of-a-kind” points, the constraint will imply (A\_1 /\ B\_1).

For A\_n, n <= 6. A player cannot get 7, 8, 9, … of a kind of a champion.

A\_7, A\_8, A\_9,… are all false and do not exist.

# Model Exploration: Results and Explanation

The model will return the best possible choice to complete team composition.

The board is initially generated with 8 champions, with a maximum number of 5 of the same kind. There will be 4 champions that are randomly generated for the players to choose from in order to complete the team compositions.

Strategies for choosing the champion:

* 6 and 4 and 2 of the same kind are prioritized
* Otherwise, the higher the letter the better
* If there is a tie of two kinds
* The higher the letter the better

Examples

|  |  |  |  |
| --- | --- | --- | --- |
| Initial compositions | Champions on board | Choice | Logic |
| ABCDDGGJ | BDEJ | B | B: 2 of the same kind is prioritized  D: 3 of the same kind is not prioritized  E: 1 of the kind…  J: 2 of the same kind but “lower” than B |
| AACCDDDD | ABCD | A | 3 or 5 of the same kind are not prioritized, therefore the greater letter is chosen |
| ACDFFJJJ | ACEJ | J | 4 of the same kind is prioritized than 2 of the same kind |
| ABDEFGGJ | ACDJ | A | A: the greatest 2 of the kind on the board |

# Past Models

This model gives the player up to 9 champions.

The model aims to find the best composition to win the game against the enemy team. 11 classes of champions (excluding Caretaker, Cruel, and God-King) are considered for this past model.

Possible classes:



In this model, a champion can have 2 classes.

Some other constraints:

The team can’t have all nine champions from the same class

A champion cannot have more than two classes

The benefits received for each class for having multiple champions in that class are limited to intervals. For example, at two and four assassins, the assassins receive an extra benefit, however at three assassins, the benefits are not increased from two assassins. These intervals are different for every class.

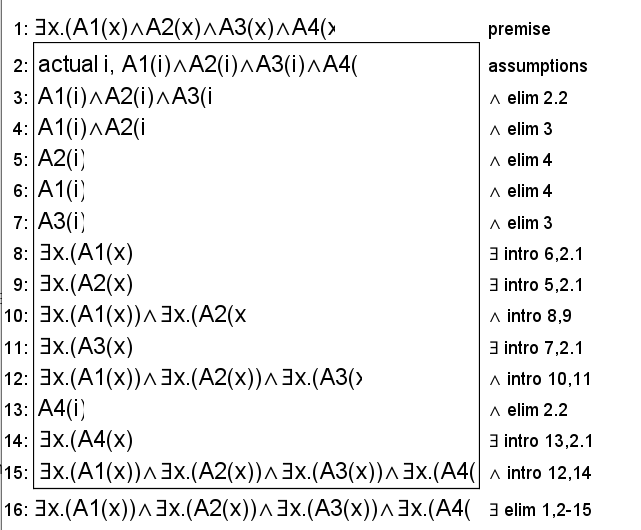
As it can observed, this model is a bit too complicated with having benefits from multiple classes. Our final model ended up with only having champions of a single type.

# Jape Proofs

1.)

If there are 4 champions of the same value to pick from, then each of the 4 will be similar choices, and the optimal champion to choose would be any 4 of them.

∃x.(A1(x)∧A2(x)∧A3(x)∧A4(x))⊢∃x.(A1(x))∧∃x.(A2(x))∧∃x.(A3(x))∧∃x.(A4(x))

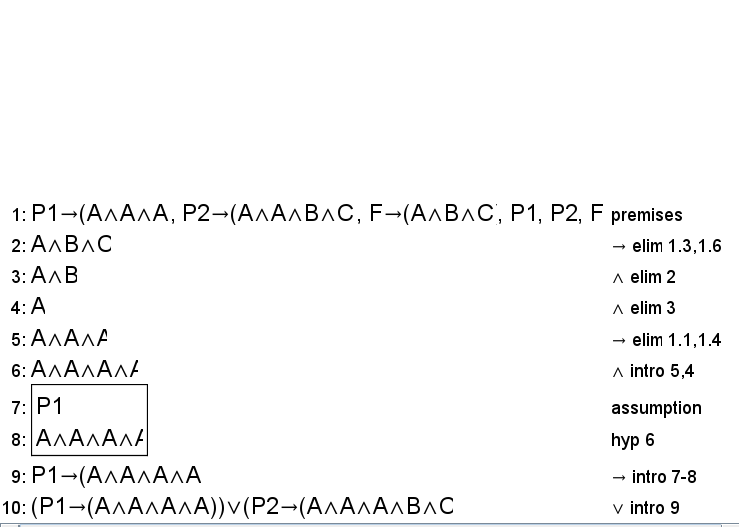


2.)

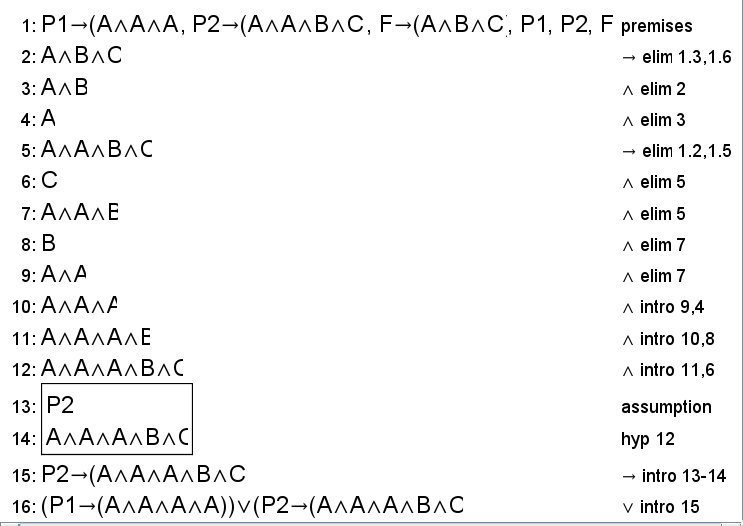
For a player 1 P1 with the hand (A1,A2,A3), player 2 P2 with the hand (A1,A2,B1,C1) and the field F containing the champions (A,B,C), it can be shown that both P1 and P2 are eligible to pick up another champion of type A, but only 1 player is able to take the champion A.

P1→(A∧A∧A), P2→(A∧A∧B∧C), F→(A∧B∧C), P1,P2,F⊢(P1→(A∧A∧A∧A))∨(P2→(A∧A∧A∧B∧C))

Showing eligibility for P1:

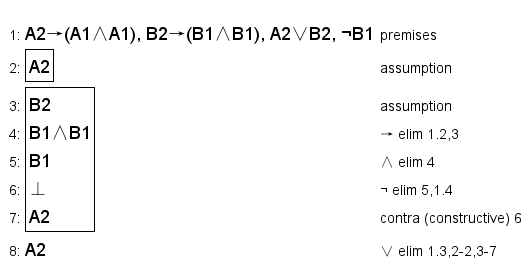


Showing eligibility for P2:



3.)

When the player has two champions A and B, and another two champions to choose from A and B, they will go with the bigger of the two which is A



# First-Order Extension

**For the propositions:**

Instead of representing each value of whether A\_1, …, A\_6 is in the possession of the player, we can have an identifier player-possession = x, where x is an integer, where 1 <= x <= 6.

The method to present the champions on the board for predicate logic is similar to the way it is represented for propositional logic because the number of a certain type of champions on the board is not useful, only the information about the existence of certain types of champions are useful.

Player’s hand

A(n): object n is the number of champion A the player owns

B(n): object n is the number of champion B the player owns

…

J(n): object n is the number of champion J the player owns

Board

Ab(n): object n is the number of champion A on the board

Bb(n): object n is the number of champion B on the board

…

Jb(n): object n is the number of champion J on the board

**For the constraints:**

Symbols: ¬∧∨→∀∃

Predicate logic allows the shortening of the constraint that when A(n) exists A(n-1),…,A(1) also exist.

Existence of n-1,…,1 given that A(n) exists:

∃n.(A(n)) → ∀m.(A(m)) such that 1 <= m <= n

For the added constraints of class restrictions, the formulas do not change when using predicate logic because the variables and the existential and universal do nothing extra that propositional logic does not already achieve.

(A(n) /\ B(n)) -> ~C(n)

(A(n) /\ C(n)) -> ~B(n)

(B(n) /\ C(n)) -> ~A(n)

(D(n) /\ E(n)) -> ~F(n)

(D(n) /\ F(n)) -> ~E(n)

(E(n) /\ F(n)) ->~D(n)

(G(n) /\ H(n)) -> ~J(n)

(G(n) /\ J(n)) -> ~H(n)

(H(n) /\ J(n)) -> ~G(n)

The constraint to determine the higher value for a champion is similar in predicate logic as it is in propositional logic. Since variables are used to determine the amount of a champion type that exist, the variables and the existential and universal are not helpful, and thus would indicate that the constraint would be similar in propositional and predicate logic.

(H(n) /\ J(n)) -> H(n)

(G(n) /\ H(n) /\ J(n)) -> G(n)

(A(n) /\ B(n) /\ C(n) /\ D(n) /\ E(n) /\ F(n) /\ G(n) /\ H(n) /\ J(n)) -> A(n)

(B(4) /\ C(4)) -> B(4)

The constraint to determine the highest valued champion, 6/4/2 of a kind, will be represented the same way as it has been represented in propositional logic because there is no benefit to adding variables values if the comparison is between A\_5 and A(5).

Constraint to determine 6 of a kind:

(A(5) /\ B(5) /\ C(4)) -> (A(5) /\ B(5))

Constraint to determine 4 of a kind:

(A(3) /\ B(3) /\ C(2)) -> (A(3) /\ B(3))

Constraint to determine 2 of a kind:

(A(1) /\ B(1) /\ C(4)) -> (A(1) /\ B(1))