

# Adaptive AI - A Written Tutorial

## Blake Skaja

### Table of Contents

[Table of Contents](#)

[Introduction](#)

[Acknowledgements](#)

[Resources](#)

[Adaptive AI - What is it?](#)

[The Game](#)

[Advantages of an Adaptive AI](#)

[Disadvantages of an Adaptive AI](#)

[Core Concepts](#)

[Requirements](#)

[Computational Requirements](#)

[Functional Requirements](#)

[Dynamic Scripting](#)

[Rulebases](#)

[Static Round Evaluation](#)

[Weight Adjusting](#)

[Turning Point](#)

[Simulations](#)

[Static vs Static](#)

[Unbiased Rulebases](#)

[Biased Rulebases](#)

[Mutating AI](#)

[Conclusion](#)

# Introduction

This document is written with the purpose of teaching how to implement an Adaptive AI in a video game. We will discuss what an adaptive AI is, why you would want to use one in your game, the core concepts of this type of AI and finally demonstrate the AI's power in a fighting simulation game.

# Acknowledgements

I would like to give credit to resources in which I was able to gather information on this type of AI system. As this topic is incredibly theoretical and still in the research infancy stages, the following resources provided a large amount of information for my tutorial.

Pieter Spronck

<http://www.socsci.ru.nl/idak/publications/papers/DynamicScripting.pdf>

<http://ilk.uvt.nl/~pspronck/pubs/PonsenCGAIDE.pdf>

Antiono and Patrick

<http://cs229.stanford.edu/proj2008/RicciardiThill-AdaptiveAIForFightingGames.pdf>

# Resources

The code for this project can be found at the public github repo listed below. Feel free to use the code as a guideline for implementing your own Adaptive AI system.

<https://github.com/Skyman12/AdaptiveAI>

# Adaptive AI - What is it?

An Adaptive AI is an artificial intelligence system that is able to perform online learning to make better decisions in the future. This type of AI is sometimes referred to as a dynamic AI or an evolutionary learning system. Regardless of the title, the core concepts are the same. An adaptive AI works by evaluating its success of specific actions and then adjusting the computer's play style accordingly. For example, in a fighting simulation game, attacking specific targets with specific abilities may lead to better results. By evaluating the success of actions and then dynamically adjusting the likelihood of using that same ability, we are able to see a performance increase in our AI's performance.

An adaptive AI can be implemented in any game in which the following criteria is satisfied:

- The game AI has to be able to be implemented via scripts.
- The success of actions have to be both recorded and evaluated for efficiency.

As seen, this criteria is fairly vague and can be applied to a wide variety of games. The most practical applications exist in turn based games, in which there is a natural time to evaluate the success of the last rounds.

## The Game

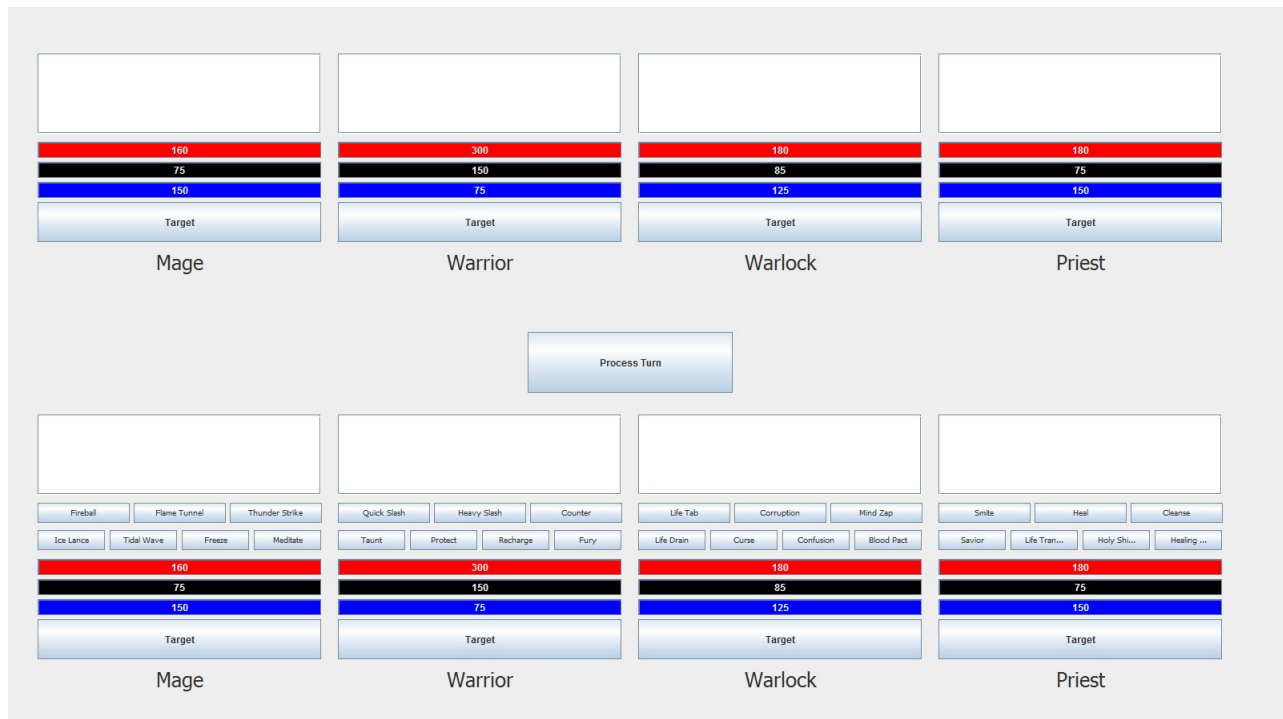
In order to implement an adaptive AI for this tutorial, a game had to first be created as a way to test the AI. While this tutorial is going to focus on how to create an adaptive AI for any game, it is worthwhile to know how the game that the simulations were run on operates.

The game created is a simple yet sophisticated team fighting simulation. It was designed to be easy to play and quick to pick up. However, there is enough different strategies in the game to make a smart AI worthwhile.

Each team is given four players to control. There are six different classes from which a player can build their team. The classes are Warrior, Rouge, Mage, Warlock, Priest and Bard. Like most MMO type games, the classes all operate a little differently. The Warrior has more health and shield, while the Priest can heal and buff teammates.

Each player has a set of characteristics, including health, shield and energy. Each player also has a set of three basic attacks and a set of four abilities. Both basic attacks and abilities have a speed associated with them. The basic attacks and abilities with lower speeds are processed first in the round. Abilities also have energy cost associated with them, while basic attacks are free. The basic attacks and abilities have functionality ranging from damage, healing, buffing, stunning, taunting, protecting and confusion.

Below is an image of the UI for the game.



The object of the game is to kill all of your opponent's players before they kill you. Throughout the remainder of this tutorial, references will be made to this game as a way of showing how these principles can be implemented. These adaptive AI concepts, however, can be applied to any game.

## Advantages of an Adaptive AI

The implementation of an adaptive AI system in your game provides a plethora of advantages. The most obvious advantage is the fact that the AI, if implemented this way, is going to provide the human player with a much more difficult and challenging experience. The adaptive AI is able to learn and adjust the computer's playstyle to learn from what the human player is doing. This helps eliminate the issues that arises when a human player learns a combination of moves that is always successful. Because the adaptive AI is able to perform online learning, a solution to the human's move combination can be discovered and implemented, making the game much more of a challenge.

Additionally, an adaptive AI helps a developer because it is able to learn and mutate throughout the course of the playing experience. It can be hard to design and implement an AI that is the most powerful without going through rigorous testing. Even then, there is a chance that the human player discovers and uses tactics that even the best programmed AI's have not been

scripted to deal with. By using a dynamic scripting system, the game is able to learn what works best and then build the AI using that knowledge.

## Disadvantages of an Adaptive AI

As with all AI systems, there are going to be drawbacks of using a specific methodology. With the adaptive AI implementations, there are a few limitations that are apparent. The most daunting of the disadvantages is the fact that an adaptive AI can have a long learning time to reach the turning point. This means that for a long period of a human player's gaming experience, the adaptive AI may not be fully competitive.

There is also a fear that the adaptive AI can learn inferior behaviors, which would lead to a less fun gameplay style. Because our AI system is constantly learning and improving its AI, there is also reason to believe that the AI could become too good and lead to a frustrating experience for the user. With all games, the AI needs to be fair, fun and beatable. An adaptive AI, has potential to become predictable and unbeatable.

There is also a fairly narrow field for application of this AI system. As stated before, this type of AI can only be implemented in games that rely on scripted actions and actions that can be immediately evaluated mathematically for their success or failure. This leads to a fairly specific range of games that this system would truly be effective for. Lastly, as this is a relatively new style of artificial intelligence, there is limited research and information available on the topic.

## Core Concepts

This section will discuss the theoretically components that make up an adaptive AI system.

### Requirements

In the research paper "Adaptive AI with Dynamic Scripting" by Pieter Spronck, a series of requirements, both computational and functional, are defined for the implementation of an adaptive AI.

#### Computational Requirements

The four computational requirements for an adaptive AI are as follows:

**Speed:** The AI must be able to quickly generate static round evaluations, as the learning process occurs during the game.

**Effectiveness:** The AI needs to consistently produce reasonably successful behavior.

**Robustness:** The AI needs to be able to deal with the element of randomness that is inherent in all video games.

**Efficiency:** The AI needs to quickly reach the turning point, or the point in which the AI is successful in its actions more often than it is unsuccessful.

## Functional Requirements

The four functional requirements for an adaptive AI are as follows:

**Clarity:** The AI needs to produce results that can be easily interpreted by the developers.

**Variety:** The AI needs to produce a variety of different actions, as an AI that produces the same action scripts is not as entertaining as one that has a multitude of actions.

**Consistency:** The AI needs to consistently reach its turning point in a low number of encounters.

**Scalability:** The AI needs to scale the difficulty level of its results to the human player's skill level.

The requirements provide a template for the bare minimum requirements needed for a successful adaptive AI.

## Dynamic Scripting

Spronck defines dynamic scripting as “an online competitive machine-learning technique for game AI, that can be characterised as stochastic optimisation.” Online is referring to the fact that our AI is going to be learning and adjusting during game play. Competitive alludes to the fact that our AI is going to be able to produce results that will challenge the human player.

Dynamic scripting is made up of several different components. We need each of the characters in our game to have a rulebase associated with them. After each round, we need a way to statically evaluate the results of the actions used each turn and adjust those rules in the rulebase according to the contribution. Below is a visual representation of how dynamic scripting works, taken from Spronck's research paper.

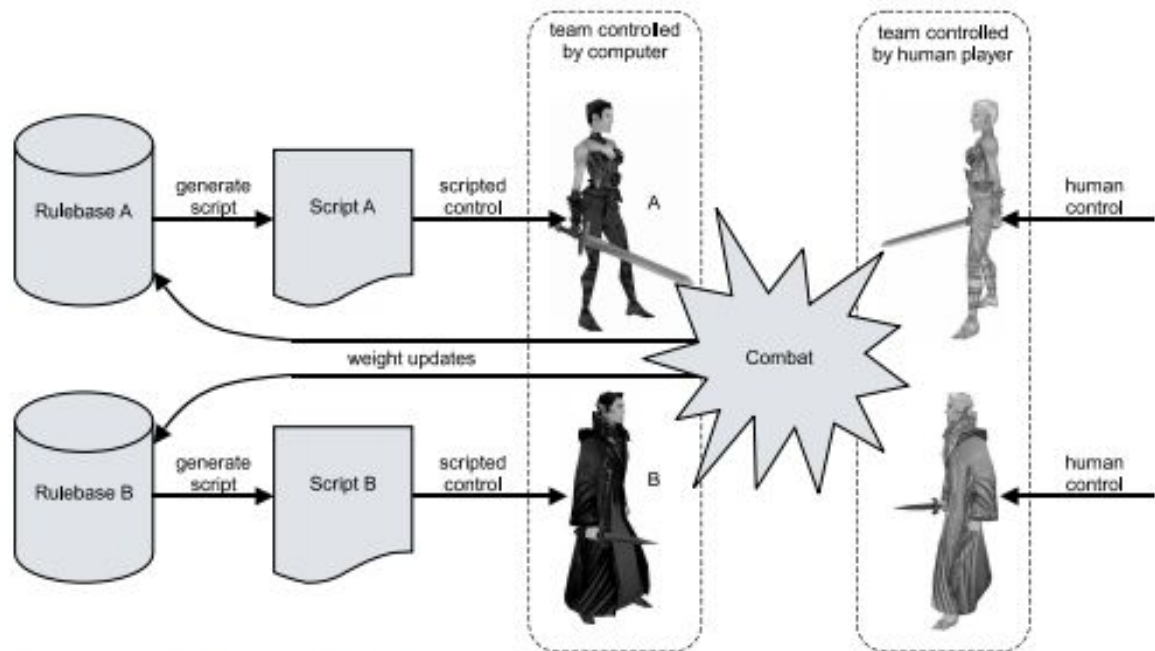


Figure 1. Dynamic scripting.

## Rulebases

Rulebases are a large portion of the adaptive AI methodology. Rulebases are defined per character and give meaning to what actions a certain character can perform. Each rule in the rulebase has a script component and a weight component. The script details the action that the rule will perform. For example, in the game that was discussed previously in this document, each character has two separate rulebases - one for their basic attacks and one for their abilities. Each rule in the rulebase has a weight value associated with it, which represents the likelihood of selecting that rule for execution during the next rounds.

### Example Rulebase for the Mage Class

Rulebase Name: Basic Attacks

Rule 1:	Fireball	25.0
Rule 2:	Flame Tunnel	15.0
Rule 3:	Thunder Strike	60.0

Rulebase Name: Abilities

Rule 1:	Ice Lane	18.0
Rule 2:	Tidal Wave	12.0
Rule 3:	Freeze	40.5
Rule 4:	Meditate	19.5

An important part of the rulebase system is the idea that all of the rules' weights in the rulebase always add up to a constant number. In this case, that number is 100.0. This means that when one rule is given an increase to its weight, the rest of the rules in the rulebase must be decreased to keep the total constant.

Additionally, each rule in the rulebase also has its own rulebase for determining the targets of the attack. Our game allows us to define attack criteria for how each attack's target should be chosen. For example, the Freeze ability for the Mage has this following attack criteria:

Rulebase Name: Freeze Targets

Rule 1:	Target Mage	34.5
Rule 2:	Target Priest	25.5
Rule 3:	Target Bard	15.0
Rule 4:	Random Target	25.0

What this is saying is that we want to target the opposing team's Mage, Priest or Bard the majority of the time. Sometimes we want to choose a random target to Freeze. Because this is a dynamically changing rulebase, if using Freeze on the opposing Mage provides the best results, the weight of that rule is going to be increased, which will lead to that script being used more in the future.

Another important component of the rules is the idea of implementing hard and soft caps.

A hard cap is a "can I" use this move statement. This evaluates if the rule in the rulebase can be used this turn. For example, in the game implemented for this tutorial, each ability has a energy cost associated with it. Our hard cap checks to see if the energy cost associated with that ability exceeds the current energy of our player. If so, we know that we cannot use that ability this round.

```
public boolean getHardCap() {
    // Check for energy cap - do they have enough energy
    if (theAttacker.currentEnergy < cost) {
        return false;
    }

    return true;
}

public boolean getSoftCap() {
    return true;
}
```



A soft cap is a “should I” use this move statement. This cap uses a logical approach to see if this rule should be considered for usage next turn. For example, in our Priest's healing spell, the soft cap checks to see if any of the Priest's teammates have lost more health than the heal spell will restore. If so, we can add this rule to the possible rules for execution. If no members of the Priest's team are injured, we do not want the priest to use that ability this turn.

```
@Override
public boolean getSoftCap() {
    ArrayList<Class> players = getAliveAllies(theAttacker);
    players.add(theAttacker);
    for (Class p : players) {
        if (p.baseHealth - p.currentHealth > 15) {
            return true;
        }
    }

    return false;
}
```

## Static Round Evaluation

Static round evaluation is the process of mathematically measuring the performance of the rules used in last round's execution. This is a crucial part of the adaptive AI process, as it is in this stage that the rules that lead to success are given an increase to their weights and the rules that lead to failure are given a decrease.

Static round evaluation takes the difference from the previous round to the current round to measure how the rules performed. Below is an example table that illustrates this process:

Round	Team 1 Score	Team 2 Score	Differential
1	2000	2000	-
2	1600	1700	Team 2 +100
3	1500	1500	Team 1 +100
4	800	1200	Team 2 + 400
5	500	700	Team 1 +200
6	0	200	Game Over - Team 2 Wins

For example, the initial scores of the teams are the same. A team's score is calculated by summing the scores for each player on the team. A player receives points for how much health, shield and energy they have left. A player loses points if they are stunned or confused. The higher the score for a player, the better that player is doing. A score of 0 means that player is dead.

After the second round, we compare and see which team performed better. Team 1 has a total score of 1600, while Team 2 has a score of 1700. This means that Team 2 performed better by 100 points, as the difference from round 1 to round 2 implies that Team 2 did more damage to Team 1 than Team 1 did to Team 2. This means that all the rules that were used during Team 2's round 2 execution are given a weight increase, as those rules and the targets of those rules led to a positive round for Team 2.

After the third round, we see that even though the total score is tied, Team 1 performed better from round 2 to round 3, as they lost only 100 points compared to Team 2's 200 point loss. This means that all the rules that were used during Team 2's round 2 execution are given a weight decrease, as those rules and the targets of those rules led to a negative round for Team 2.

## Weight Adjusting

The weight adjusting occurs right after the static round evaluation takes place. This is the process of determining how much a rule in a rulebase is going to increase/decrease and making sure that the rest of the rules in that same rulebase are compensated for to keep our weight total constant. In the code for our game, the process of doing this can be seen below:

```
public void processRound() {
    makeAIMoves();
    orderAttacks();

    for (Attacks attack : roundAttacks) {
        attack.executeAttack();
        attack.theTargets.clear();
        checkForGameOver();
        if (gameOver) {
            return;
        }
    }

    assignRoundEvaulation();
    updateDyanmicWeights();
}
```

All of the attacks for the rounds are executed, and the round evaluation is assigned. The dynamic weights are then updated to reflect the success or failure of the last used rules. We also assign the rules a slight adjustment that is unrelated to the overall success of the round. For example, if we used Freeze last turn and that was a successful action, but the round was still lost, we do not want to penalize this ability for as much as an ability that had no positive

effect on the round. This allows us to keep abilities that are performing well but happen to get paired with unsuccessful performing rules from taking too hard of a weight decrease.

## Turning Point

The turning point for an adaptive AI is the point in which the adaptive AI is able to routinely outperform the human or a static AI. In our game, this is measured when the adaptive AI is able to first win 10 games in a row. Low turning points for an adaptive AI are important, as the less games it takes the AI to learn and adapt means less time a human has to play the game before getting a good challenge.

## Simulations

As a way of measuring the power of the adaptive AI, simulations were run against a variety of different static AI configurations. The results were recorded and some interesting conclusions were drawn. In all of these simulations, the same team configuration was used. (Mage, Warlock, Bard and Warrior). This guarantees that the AI's and the AI's alone were the difference between the teams performance. Each simulation was run 10,000 times, with those results being recorded. We ran the 10,000 game simulations 100 times and took an average of the results to account for outlying game simulations in which the turning point could have been found too early due to random chance (in two equally balanced teams, in which each team has a 50% win chance, there is a chance that one team could win the first 10 games, simply due to random odds).

## Static vs Static

The first simulation that was run was a Static AI vs another Static AI. This test was run to show that two teams of the same configuration and the same basic AI should perform equally well against each other. The results are as follows:

```
-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 4974
Average number of Team 2 Wins over 100 simulations: 4976
Average number of Ties over 100 simulations: 48
```

What we found is what we expected. Over 100, 10,000 game simulations, we found that on average, Team 1 won 4,974 / 10,000 times and Team 2 won 4,976 / 10,000 times. The difference of two games over 10,000 game simulations implies that our game is balanced and if two identically AI teams compete, they are equally likely to win.

## Unbiased Rulebases

The next simulations introduced our adaptive AI into the game. We seeded our adaptive AI with an unbiased rulebase. An unbiased rulebase is one that gives each ability an equal chance of being initially used. For example, an unbiased rulebase for our mage would look like this.

### Example Unbiased Rulebase for the Mage Class

Rulebase Name: Basic Attacks

Rule 1:	Fireball	33.3
Rule 2:	Flame Tunnel	33.3
Rule 3:	Thunder Strike	33.3

Rulebase Name: Abilities

Rule 1:	Ice Lane	25.0
Rule 2:	Tidal Wave	25.0
Rule 3:	Freeze	25.0
Rule 4:	Meditate	25.0

Additionally, each rule would have an equal chance of choosing their targets.

Rulebase Name: Freeze Target's

Rule 1:	Target Mage	25.0
Rule 2:	Target Priest	25.0
Rule 3:	Target Bard	25.0
Rule 4:	Random Target	25.0

This allows the rules that are the most effective to see their weights rise, while the ineffective rules will have their weights decrease.

The adaptive AI was simulated against four different static AI templates.

**Balanced:** Equal chance for choosing a rule and target.

**Aggressive:** Rules that deal damage are given higher weights.

**Defensive:** Rules that heal, buff and protect teammates are given higher weights.

**CC:** Rules that stun, confuse or taunt enemies are given higher weights.

In the simulations, Team 1 represents the static AI, while Team 2 is the adaptive AI. The results of these simulations are as follows:

## Against Balanced Static AI

```

-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 2604
Average number of Team 2 Wins over 100 simulations: 7394
Average number of Ties over 100 simulations: 0
-----Turning Point Found-----
Average turning point over 100 simulations: 937
-----Before Turning Point -----
Average number of Team 1 Wins over 100 simulations: 683
Average number of Team 2 Wins over 100 simulations: 253
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 1920
Average number of Team 2 Wins over 100 simulations: 7141
Average number of Ties over 100 simulations: 0

```

## Against Aggressive Static AI

```

-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 2144
Average number of Team 2 Wins over 100 simulations: 7855
Average number of Ties over 100 simulations: 0
-----Turning Point Found-----
Average turning point over 100 simulations: 1143
-----Before Turning Point -----
Average number of Team 1 Wins over 100 simulations: 771
Average number of Team 2 Wins over 100 simulations: 371
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 1372
Average number of Team 2 Wins over 100 simulations: 7483
Average number of Ties over 100 simulations: 0

```

## Against Defensive Static AI

```

-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 4899
Average number of Team 2 Wins over 100 simulations: 5095
Average number of Ties over 100 simulations: 4
-----Turning Point Found-----
Average turning point over 100 simulations: 939
-----Before Turning Point -----
Average number of Team 1 Wins over 100 simulations: 781
Average number of Team 2 Wins over 100 simulations: 357
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 4118
Average number of Team 2 Wins over 100 simulations: 4737
Average number of Ties over 100 simulations: 4

```

## Against CC Static AI

```

-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 3984
Average number of Team 2 Wins over 100 simulations: 6014
Average number of Ties over 100 simulations: 0
-----Turning Point Found-----
Average turning point over 100 simulations: 617
-----Before Turning Point -----
Average number of Team 1 Wins over 100 simulations: 562
Average number of Team 2 Wins over 100 simulations: 254
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 3421
Average number of Team 2 Wins over 100 simulations: 5760
Average number of Ties over 100 simulations: 0

```

## Adaptive AI vs Static AI: Unbiased Rulebase Results

Team Type	Turning Point	Overall Win Percentage	Before Turning Point	After Turning Point
Balanced	937	73%	27%	78%
Aggressive	1143	78%	32%	84%
Defensive	939	51%	31%	53%
CC	617	60%	31%	62%

The results of those simulations are as shown above. Our adaptive AI had a higher overall win percentage than the static AI, regardless of the static AI's playstyle. Additionally, once the turning point was found, the win percentage for the adaptive AI could be as high as 84%. All in all, the adaptive AI, using an unbiased rulebase, was able to outperform all the static AI teams.

### Biased Rulebases

Building off of the results from the unbiased rulebase simulations, it was decided to run the same simulations, this time using a biased rulebase. A biased rulebase is one that is given an initial template and then allowed to adjust itself from that starting point. Since the defensive template was the static AI that performed the best in our previous simulations, it was decided to seed our adaptive AI with that starting template. The same simulations were run, and the results are as seen below:



## Against Balanced Static AI

```

-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 3674
Average number of Team 2 Wins over 100 simulations: 6323
Average number of Ties over 100 simulations: 2
-----Turning Point Found-----
Average turning point over 100 simulations: 586
-----Before Turning Point -----
Average number of Team 1 Wins over 100 simulations: 377
Average number of Team 2 Wins over 100 simulations: 208
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 3296
Average number of Team 2 Wins over 100 simulations: 6114
Average number of Ties over 100 simulations: 2

```

## Against Aggressive Static AI

```

-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 2891
Average number of Team 2 Wins over 100 simulations: 7104
Average number of Ties over 100 simulations: 4
-----Turning Point Found-----
Average turning point over 100 simulations: 353
-----Before Turning Point -----
Average number of Team 1 Wins over 100 simulations: 225
Average number of Team 2 Wins over 100 simulations: 127
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 2666
Average number of Team 2 Wins over 100 simulations: 6976
Average number of Ties over 100 simulations: 4

```

## Against Defensive Static AI

```

-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 4594
Average number of Team 2 Wins over 100 simulations: 5401
Average number of Ties over 100 simulations: 4
-----Turning Point Found-----
Average turning point over 100 simulations: 337
-----Before Turning Point -----
Average number of Team 1 Wins over 100 simulations: 245
Average number of Team 2 Wins over 100 simulations: 91
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 4348
Average number of Team 2 Wins over 100 simulations: 5310
Average number of Ties over 100 simulations: 4

```

## Against CC Static AI

```

Average number of Team 1 Wins over 100 simulations: 3477
Average number of Team 2 Wins over 100 simulations: 6521
Average number of Ties over 100 simulations: 1
-----Turning Point Found-----
Average turning point over 100 simulations: 357
-----Before Turning Point -----|
Average number of Team 1 Wins over 100 simulations: 244
Average number of Team 2 Wins over 100 simulations: 112
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 3232
Average number of Team 2 Wins over 100 simulations: 6408
Average number of Ties over 100 simulations: 1

```

## Adaptive AI vs Static AI: Biased Rulebase Results

Team Type	Turning Point	Overall Win Percentage	Before Turning Point	After Turning Point
Balanced	586	63%	35%	65%
Aggressive	353	71%	36%	72%
Defensive	337	54%	27%	67%
CC	357	65%	31%	66%

## Comparison of Biased and Unbiased Rulebases

The biased rulebase data is shown in **bold**.

Team Type	Turning Point	Overall Win Percentage	Before Turning Point	After Turning Point
Balanced	<b>586</b> -- 937	<b>63%</b> -- 73%	<b>35%</b> -- 27%	<b>65%</b> -- 78%
Aggressive	<b>353</b> -- 1143	<b>71%</b> -- 78%	<b>36%</b> -- 32%	<b>72%</b> -- 84%
Defensive	<b>337</b> -- 939	<b>54%</b> -- 51%	<b>27%</b> -- 31%	<b>67%</b> -- 53%
CC	<b>357</b> -- 617	<b>65%</b> -- 60%	<b>31%</b> -- 31%	<b>66%</b> -- 61%

What we can see from the following data is that using a biased rulebase drastically reduces the number of games that it takes for the adaptive AI to reach its turning point. This is important, as the implementation of an adaptive AI is reliant on being able to find that point as fast as possible. It is also seen that the biased rulebase led to some increases in win percentage, but also some decreases when compared to the unbiased rulebase adaptive AI. The explanation we have for this peculiar trend is that using a biased rulebase forces some less inferior rules to have their weights rise. For example, say that the Mage is the most successful when using Freeze on the enemy's Priest. However, the Mage is also successful when using Tidal Wave on the entire team, just not as successful as when it uses Freeze. Because our biased rulebase forces the initial weights of some rules to be increased, it is possible that we are going to find a set of winning rules that although produce winning results, are not the ideal way to win. This leads to winning faster, which is evident by the lower turning point, but not winning at nearly as high of a rate after that turning point is found.

## Mutating AI

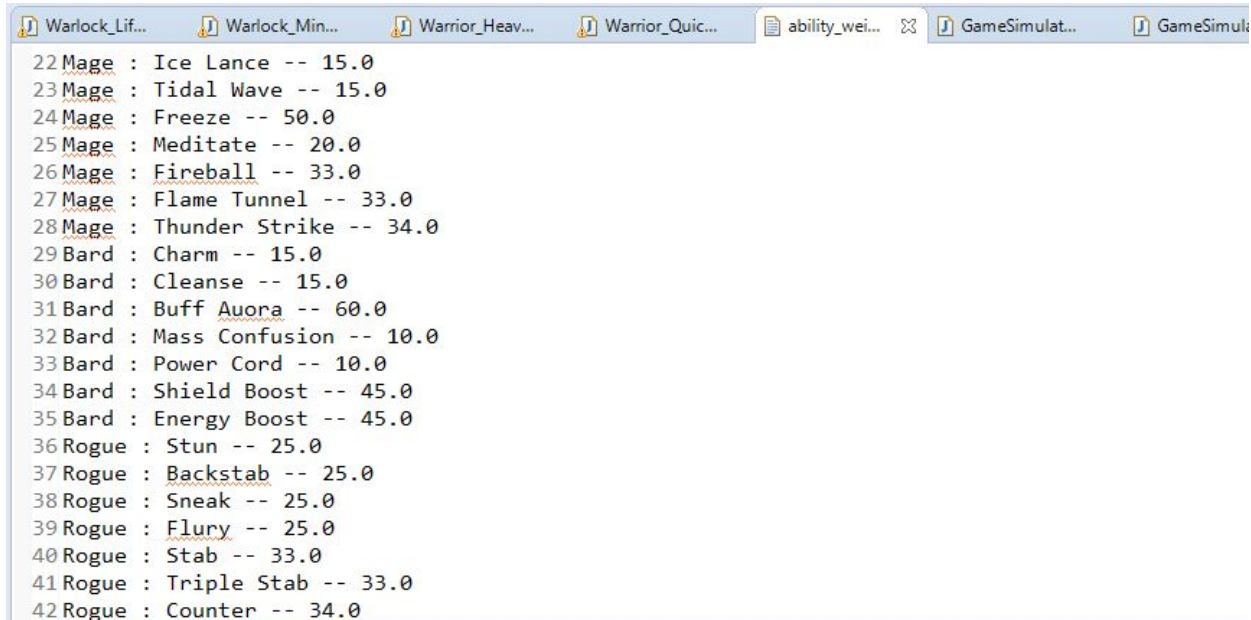
The last simulation run was the biased rulebase adaptive AI vs a Mutating AI, which is set to resemble how a human would play. Most gaming players will use a strategy as long as that strategy produces winning results. Once that strategy loses, a player will often switch and try a new tactic. This is the same as our mutating static AI. The mutating static AI will use a weight template until it loses, in which it will then randomly choose one of the four weight templates and use that one until it loses. The results of the mutating AI simulation are found below:

```

-----Simulation Complete -----
Average number of Team 1 Wins over 100 simulations: 3250
Average number of Team 2 Wins over 100 simulations: 6746
Average number of Ties over 100 simulations: 2
-----Turning Point Found-----
Average turning point over 100 simulations: 466
-----Before Turning Point -----
Average number of Team 1 Wins over 100 simulations: 342
Average number of Team 2 Wins over 100 simulations: 123
Average number of Ties over 100 simulations: 0
-----After Turning Point-----
Average number of Team 1 Wins over 100 simulations: 2908
Average number of Team 2 Wins over 100 simulations: 6622
Average number of Ties over 100 simulations: 2

```

What we see it that we still get very good results, even against a mutating static AI. Our turning point was slightly higher than the previous simulations, which is to be expected. However, this simulation proves that the adaptive AI is viable against a human's playstyle. The game implementation that was created provides a way for saving this data for human use. After every game, the data is written to a .txt file, in which the weights of the abilities can be reloaded the next time the human plays the game.



```

Warlock_Lif...  Warlock_Min...  Warrior_Heav...  Warrior_Quic...  ability_wei...  GameSimulat...  GameSimulat...
22 Mage : Ice Lance -- 15.0
23 Mage : Tidal Wave -- 15.0
24 Mage : Freeze -- 50.0
25 Mage : Meditate -- 20.0
26 Mage : Fireball -- 33.0
27 Mage : Flame Tunnel -- 33.0
28 Mage : Thunder Strike -- 34.0
29 Bard : Charm -- 15.0
30 Bard : Cleanse -- 15.0
31 Bard : Buff Auora -- 60.0
32 Bard : Mass Confusion -- 10.0
33 Bard : Power Cord -- 10.0
34 Bard : Shield Boost -- 45.0
35 Bard : Energy Boost -- 45.0
36 Rogue : Stun -- 25.0
37 Rogue : Backstab -- 25.0
38 Rogue : Sneak -- 25.0
39 Rogue : Flury -- 25.0
40 Rogue : Stab -- 33.0
41 Rogue : Triple Stab -- 33.0
42 Rogue : Counter -- 34.0

```

## Conclusion

This tutorial has hopefully allowed you to understand the basics of what adaptive AI is and how you would go about implementing a system yourself. All in all, adaptive AI is a neat concept at best. It provided promising results during the simulations and is relatively simple to implement. However, the long learning time (408 games with a biased rulebase), makes this implementation not practical for an actual game. Not only was the turning point much too high, but the abilities that were the most effective would often rise to have a weight of near 99%, making the game boring and no fun to play. This system would be great for testing and finding rules in the rulebase that are overpowered and unbalanced. Unless the game is truly balanced, the adaptive AI is going to naturally percolate some of the rules in the rulebase to a weight of 99, which makes sense as the AI wants to produce successful rounds.

I hope that you have learned and enjoyed this tutorial.

Questions, comments or concerns, please contact:

Blake Skaja - [bskaja@iastate.edu](mailto:bskaja@iastate.edu)