Artificial Intelligence for Modern Boardgames

Frank Madrid
California State University, Bakersfield
fmadrid@cs.csubak.edu

Contents

I. Introduction

A. Project Description

In the last twenty years, there has been a rise in *abstract* or modern board games. Unlike most board games, these differ from the board games many of us played as kids; such as, *Monopoly*, *Scrabble*, or *Mancala*. Modern board games emphasize strategy, a consistent downplay of luck and conflict, and lean towards economic rather than militaristic themes. They incorporate abstract physical components, hidden information and indirect player interaction in the form of bidding, competition for scarce resources, or trade and negotiation. These qualities embody the difficulties associated with designing an artificial intelligence for modern board games. Many classic board games such as *Chess*, *Checkers*, and *Go* have been the subject of intense research and much work has been done in the past on designing artificial intellence to play these games. These intelligent agents are sufficiently advanced in that they frequently beat the best human players. Unlike their predecessors, modern board games have yet to be the subject of such intense research.

> *"The study of games within artificial intelligence stimulates the mind in ways akin to the real world, and provide ample opportunity to create interesting and challenging problems that are equally applicable to other, more 'serious' pursuits."*
>
> *Why Study AI in Games?*
>
> TOMMY THOMPSON - Researcher at University of Derby

1 Artificial Intelligence in Boardgames

Chess has been the subject of much research since the 1950's and is considered to be one of the most difficult games to design an Artificial Intelligence to play due to the game's complexity and 10^{120} board states. *Chess* is a two-player strategy board game played on a chessboard, a checkered gameboard with 64 squares arranged in an eight-by-eight grid. An example of a board state can be seen in figure 1. The objective of the game is to 'checkmate' the opponent's king by placing it under an inescapable threat of capture. To this end, a player's sixteen pieces are used to attack and capture opponent's pieces while supporting their own. Common implementation issues include: (i) board state representation, (ii) search techniques, and (iii) leaf evaluation.

1. **Board representation** - the use of a data structure to represent a single position within the game.
2. **Search techniques** - identifies possible moves and selects the most promising ones for further examination.
3. **Leaf evaluation** - the use of a heuristic function to identify the value of the current board position.

In 1997, *Deep Blue* became the first machine to win a chess game against the reigning world champion Garry Kasparov. *Deep Blue* uses a heuristic function to determine the optimum values for many parameters by analyzing thousands of master-level games and implements parallel alpha-beta search algorithms.

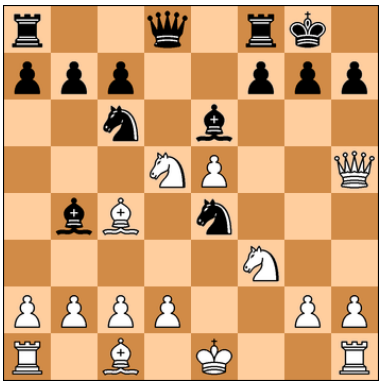

Figure 1: *Sample board state for the boardgame Chess.*

The boardgame *Go* has been the subject of much research in the field of artificial intelligence and is considered to be more difficult to solve than *Chess*. *Go* is a game involving two players who alternately place black and white playing pieces, called "stones", on the vacant intersections of a board with a 19 x 19 grid of lines. The objective of the game is to have surrounded a larger total area of the board with one's stones than the opponent by the end of the game. Figure 2 shows a sample board state of the game. Thus, an intelligent agent must repeatedly decide which location to place his stone to maximize an enclosed area by the end of the game while simultaneously minimizing the maximum enclosable area of his opponent. Various architectures have been proposed for handling this problem, the most popular of which include:

- Minimax tree searching
- the application of Monte Carlo methods
- applications in pattern matching
- creation of knowledge-based systems, and
- the use of machine learning.

The results of this research has been applied to similar fields in cognitive science, pattern recognition, machine learning, and combinatorial game theory. It is left to be shown if these architectures can be applied to board games with difficult-to-model characteristics and if their results have similar research applications.

2 Designing an Intelligent Agent for Seven Wonders

When the concept of Artificial Intelligence was first considered in the 1940's, artificial intelligence was purely academic in nature, and had little practical applicability besides playing games. At this point in time,

Figure 2: *Sample boardstate for the boardgame Go*

algorithms were not able to analyze real-world problems characterized by their ambiguity and ill-defined nature. By limiting the problem domain to a ruleset within a board game, researchers have been able to make immense progress in search, learning, and simulation techniques. As a result, artificial intelligence can now analyze real-world problems.

Dr. Sean McCulloch hosted a REU program at Ohio Wesleyan University where he developed programs to intelligently play the abstract boardgames *Football Strategy*, *Battle Line*, and *Empire Builder*. These intelligent agents were then shown to have problem solving applications in game theory and graph theory.

In this project, I will design an implementation of the game Seven Wonders for the Java Platform SE 8, explore architecture currently in use to implement board game AI and there applicability in modern board games, design a series of intelligent agents for Seven Wonders incorporating a subset of these architectures together with formalized expert knowledge, and lastly, perform a series of experiments to determine each architecture's propensity to perform as an intelligent agent for Seven Wonders.

After the experimentation, I hope to find results signifying a high propensity for the application of artificial intelligence architectures to perform as effctive intelligent agents and ultimately draw conclusions to the architecture's applicability in modern board games.

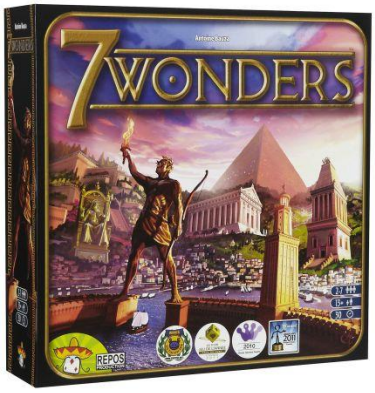

Figure 3: *Box cover for the board game Seven Wonders*

B. Project Goals

1 Game Implementation

Being the medium in where all experimentation will reside in, the design and implementation of the game *Seven Wonders* for JAVA Platform SE 8 and the design of an appropriate test bed to conduct my experiments will be my primary focus. This implementation will consist of the complete ruleset, which includes:

- **Abstraction:** An abstraction of the *player* and game element entities; such as, the *wonder boards*, *coins*, *cards*, and *conflict tokens*.
- **Cost Resolution:** Ascertaining whether the cost associated with constructing a *structure* or *stage* can be satisfied when considering renewable or non-renewable resources and when there are multiple ways to satisfy the cost.
- **Player Interaction:** The ability to engage in commercial exchanges with other players to satisfy the cost requirement for structures.

To test the performance and efficiency of the implementation, an agent which performs random actions during the game's execution will be designed to test the Java-based implementation.

2 Architecture Survey Analysis and Designing an Intelligent Agent

The next milestone of this project will consist of a survey analysis of architecture's currently implemented within classic and modern board games. I will be analyzing each architecture's propensity as an effective intelligent agent within modern board games which incorporate concepts such as, hidden information, indirect and direct player interaction, abstract components, time sensitivity, and stochastic elements.

A subset of these architectures which are most believed to be effective architectures will be implented as intelligent agents for Seven Wonders. In addition, a series of rule-based expert systems will also be implemented to measure the effectiveness of the architectures.

3 Experimentation and Results

The objective of this experiment will be to study the effectiveness and efficiency of each intelligent agent. This will be determined by analyzing the outcomes of a substantially large sequence of games between each of the intelligent agents. Though results against human players are most preferential, due to the time required to complete each game and the large number of required experimental trials, it is currently infeasible to obtain such results. Through this experimentation, I will evaluate the effectiveness of each intelligent agent and ascertain which agent had the best results.

II. Seven Wonders

A. Introduction

Seven Wonders was created by Antoine Bauza in 2010 and originally published by Repos Production in Belgium. *Seven Wonders* is a **card drafting game**[1] played using three decks of cards featuring depictions of ancient civilizations, military conflicts, and commercial activity. The game is highly regarded having won more than thirty gaming awards including the **Kennerspiel des Jahres**[2] award in 2011.

A game of *Seven Wonders* takes place over 3 Ages, each using one of the 3 card decks (first the Age I cards, then Age II, and finally Age III). These Ages are each played similarly, with each player having the opportunity to play 6 cards per Age to develop his or her city and build their Wonder. At the end of each Age, each player compares their military might with their two neighboring cities (belonging to the players to their right and left). At the end of the third Age, the players count their victory points; the player with the most points wins the game.

B. Game Elements

1 Wonderboard

At the start of the game, each player randomly receives a double-sided gameboard called the *wonderboard*. Each *wonderboard* depicts one of the seven wonders of the ancient world as described in Isaac Aasimov's *Wonders of the World*. Each board gives an initial resource to the player while each side, *Side A* and *Side B* offers a different progression of *stages* where generally the stages of side B are more complex. Each stage has a construction cost and gives a bonus when constructed. Figure 4 depicts side A of the Éphesos wonderboard.

Figure 4: *Side A of the Éphesos wonderboard.*

[1] A card drafting game is a game where players pick cards from a limited subset, such as a common pool, to gain some immediate advantage or to assemble hands of cards that are used to meet objectives within the game.

[2] The Kennerspiel Des Jahres award expands the prestigious Spiel Des Jahres to recognize the "expert" game of the year which is meant for a more experienced audience and gamers looking for more of a challenge.

2 Coins

The *coins* come into play in commercial transactions tying a city to its two neigboring cities. A player lacking the required resources to satisfy a construction cost may pay adjacent neighbors to use their resources, at two coins per resource, though this cost can be mitigated via the effects of some structures. This transaction does not limit the player's use or access to his own resources.

Figure 5: *The 'one' and 'three' value coins.*

3 Conflict Tokens

The *conflict tokens* are used to represent the military victories and defeats between neighboing cities. *defeat tokens* with a value of -1 are used at the end of each age. The *victory tokens* with values $+1, +3$, and $+5$ are used at the end of Ages I, II, and III respectively.

Figure 6: *The four conflict tokens awarded to the winners and loses of each military conflict.*

4 Cards

Each card represents a structure, and playing a card is referred to as *building a structure*. There are seven types of Age cards, representing different types of structures determined by the color of their background:

1. Brown cards, *Raw Materials*, provide up to two of the four raw material resources. These resources are used to satisfy costs of other structures or wonder stages. The brown card in figure 7 has a cost of 1 *gold* and provides 2 *brick* resources.

2. Gray cards, *Manufactured Goods*, provide one of the three manufactured goods resources. These resources are also used to satisfy costs of other structures or *wonder stages*. The gray card in figure 7 has no cost and provides 1 *papyrus* resource.

3. Blue cards, *Civic Structures*, grant a fixed number of victory points. The blue card in figure 7 has a cost of 1 *wood* and 2 *brick* and provides 5 *victory points*. If the player had previously constructed the *Baths*, the cost can be ignored.

4. Yellow cards, *Commercial Structures*, can grant coins, resources, victory points, and/or decrease the cost of buying resources from neighbors. The yellow card in figure 7 has a cost of 1 *stone* and 1 *glass* and provides 1 *gold* and 1 *victory point* per yellow structure built by the player. If the player had previously constructed the *Caravansery*, the cost can be ignored.

5. Green card, *Scientific Structures*, provides one of three scientific symbols. Combinations of these symbols provide a substantial amount of victory points. The green card in figure 7 has a cost of 1 *wood* and 1 *papyrus* and provides one *tablet* scientific symbol. By constructing this structure, the *Academy* and *Study* structure costs during the next Age are mitigated.

6. Red cards, *Military Structures*, provide one to three shield symbols dependent on the age of the respective structure. These symbols are added together to determine a player's military strength, which is used during conflict resolution at the end of each age. The red card in figure 7 has a cost of 1 *stone* and 3 *ore* and provides 3 *shields*. If the player had previously constructed the *Walls*, the cost can be ignored.

7. Purple cards, *Guilds*, provide either a scientific symbol of the player's choice or victory points based on a particular structure color, set of wonder stages, or number of conflict tokens owned by a player and/or his neighbors. The purple card in figure 7 has a cost of 3 *brick*, 1 *papyrus*, and 1 *loom* and provides 1 *victory point* per green structure built by the player's neighbors.

Figure 7: *Examples of brown, gray, blue, yellow, green, red and purple cards.*

When building a structure, a player must first pay the construction cost located in the upper left area. Alternatively, the structure can be built for free if the player has built te structure indicated on the card. If the area is empty, then the structure is free and requires no resources for its construction. The seven

resource types which constitute the cost of the structures are depicted in figure 8. Once constructed, the player immediately gains the effect of the card with the exception of structures which grant *victory point* based effects which are resolved at the end of the game.

Figure 8: *The seven types of resources: wood, stone, brick, ore, glass, loom, and papyrus.*

C. Game Overview

1 Age

A game begins with Age I, continues with Age II, and ends with Age III. At the beginning of each Age, each player recieves a hand of 7 cards, dealt randomly, from the corresponding deck of cards. As shown in figure 9, each Age uses its own deck of cards. Each Age consists of 6 game turns. During each turn the players simultaneously put into play a single card and performs the desired action. A game turn takes place as follows:

1. Choose a card
2. Choose an action
3. Give your hand of cards to the player sitting to your left or right and receive another hand of cards from the player sitting next to you.

On the last turn, the remaining 7th card is discarded and unused. During Age I and Age III, hands are pass clockwise (the player to your left) while hands are passed counter-clockwise during Age III. This change ensures each player is not receiving cards from the same player throughout the entire game.

Figure 9: *The back of the cards for each of the three ages.*

2 Actions

Figure 10 depicts the three possible actions a player may take. Each turn, a player may choose to do one of the following:

1. Construct a card
2. Discard a card to gain three gold from the supply
3. Use a card to build a stage of his wonderboard

When discarding a card, the card is placed in the general discard pile. When building a stage of a wonderboard, the player must pay the construction cost of the Wonder and then tuck a card from their hand underneath the Wonder.

Figure 10: *The three actions available each turn: (1) construct a structure, (2) construct a stage of his wonderboard, or (3) discard a card for 3 gold.*

3 Military Conflict

At the end of each age, military conflicts are resolved between neighbors. This is done by comparing the number of shield symbols owned by each player and then awarding victory points and a victory token to the winners of each conflict and penalizing the losers of each conflict with the loss of a point. Depending on the age, the winner receives one, three, or five victory points dependent on the age the victory was achieved in as depicted in figure **??**.

4 End of Game

Once the Age 3 is over, players compute their final score. The player with the most points is then declared the winner. *Victory points* are earned from the following categories:

- Conflict tokens: Number of *victory points* depicted on the conflict tokens
- Gold: One *victory point* for every three *gold*
- Wonder stages: *Victory Points* granted by Wonders

- Structures: *Victory Points* granted by structures
- Scientific symbols: Seven *victory points* for each set of scientific symbols and $x^2 + y^2 + z^2$ *victory points* where $x, y,$ and z is the number of each scientific symbol.

III. Project Implementation

A. Objects

A game of *Seven Wonders* contains the following entities: (i) *Player*, (ii) *Stage*, (iii) *Structure*, and (iv) *Wonderboard*, the following properties: (i) *Constants*, (ii) *Effect*, (iii) *Resurces*, and (iv) *Strategy*, and lastly, the following helpers: (i) *Conflict*, (ii) *ResourcesManager*, and (iii) *ScienceSymbols*. Figure 11 depicts the relationships between these objects.

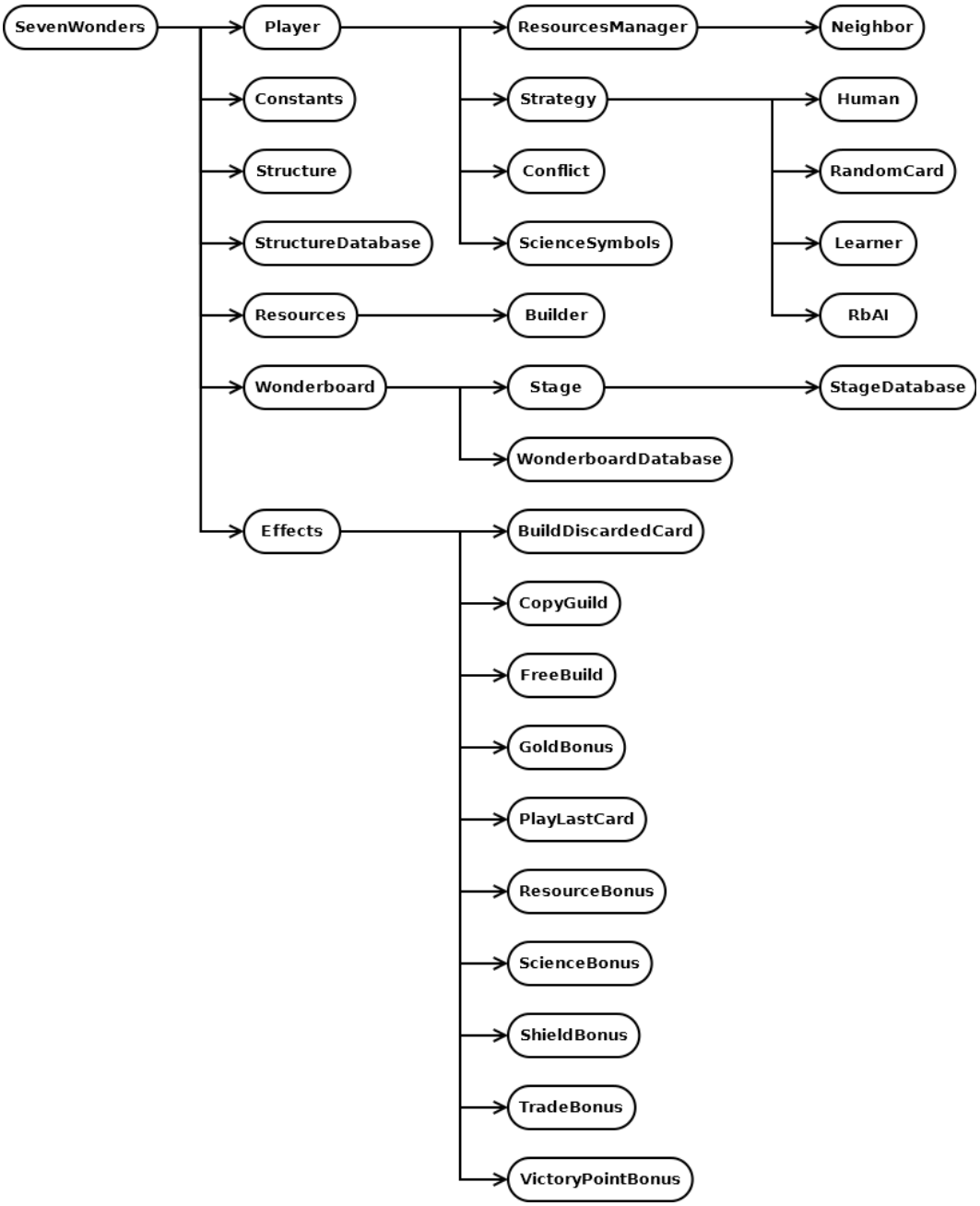

Figure 11: *Object representation for each game element within* Seven Wonders.

1 SevenWonders

The class *SevenWonders* is the object representation of the game. This object manages the players, library and discard, assigns hands to players, and action and effect resolution.

Name	Type	Description
id	int	Internal identification used to reference this instance of the game.
seed	long	The seed for all randomly generated numbers. Allows the user to reproduce the results of any experiment by referencing the seed used for the experiment.
players	ArrayList<Player>	List of players playing the game. A game must have at least 3 players and no more than 7 players.
age	Age	Represents the current round of the game. A game is played through 3 rounds, Age I, Age II, and Age III. The age dictates which cards will form the library for this round, as well as the direction hands are passed.
turn	int	The game's current turn in the round. For each round, there are 6 turns.
library	ArrayList<Structure>	List of structures which form the library. The library is repopulated at the beginning of each round and is then distributed in sets of 7 to each player.
discard	ArrayList<Structure>	List of all structures discarded by players. As an action, a player may discard a card to earn three coins.
structureDatabase	ArrayList<Structure>	Database forall structures used within the game.
wonderboardDatabase	ArrayList<Structure>	Datasebase for all wondebraords used within the game.

Figure 12: *Class* SevenWonders *member variables*

2 Player

The class *Player* is the object representation of a player of the game. The object manages the player's resources, hand, structures, military conflicts, science symbols, effects granted from structures and wonder stages, and the player's neighbors.

Name	Type	Description
name	String	Internal identification. Player's name.
wonderboard	Wonderboard	Wonderboard object assigned to the player by the game.
agent	Agent	Artificial intelligence archetecture
selectedAction	Action	Action proposed by the agent
selectedStructure	Structure	Structure proposed by the agent to perform the selected action on
hand	ArrayList<Structure>	Set of structures which constitutes the player's hand
builtStructures	ArrayList<Structure>	Structures constructed by the player
effects	ArrayList<Effects>	Effects granted from constructed structures and wonders
conflicts	Conflict	Manages shields, points, victory tokens, and defeat tokens from military conflicts
resources	ResourcesManager	Manages resources provided by ResourceBonus effects and construction costs
science	Science	Manages points and scientific symbols earned from Science-Bonus effects
leftNeighbor	Player	Player sitting directly to the left
rightNeighbor	Player	Player sitting directly to the right
freeBuild	boolean	Denotes if the player can currently ignore the construction cost of all cards
playLastCard	boolean	Denotes if the player can play the 7th card this turn as opposed to discarding it for no effect

Figure 13: *Class* Player *member variables*

3 ResourcesManager

The *ResourcesManager* class is helper to class *Player*. *ResourcesManger* manages all resources produced from structures and wonder stages constructed by the player and trading information for each neighbor. This class determines whether the construction cost of a structure or wonder stage was satisfiable by the player's resources or a combination of resources owned by the player and one or both of his neighbors. For example, suppose the player would like to satisfy the following construction cost:

ConstructionCost = [WOOD, STONE, ORE, BRICK]

with the following resources:

staticResources = [WOOD, BRICK]

dynamicResources = [STONE, ORE], [WOOD, STONE, ORE, BRICK]

then the player can generate the following resources combinations:

$$\begin{aligned} \text{resourcesCombinations} = &\text{ (1) [WOOD, BRICK, STONE, WOOD]} \\ &\text{(2) [WOOD, BRICK, STONE, STONE]} \\ &\text{(3) [WOOD, BRICK, STONE, ORE]} \\ &\text{(4) [WOOD, BRICK, STONE, BRICK]} \\ &\text{(5) [WOOD, BRICK, ORE, WOOD]} \\ &\text{(6) [WOOD, BRICK, ORE, STONE]} \\ &\text{(7) [WOOD, BRICK, ORE, ORE]} \\ &\text{(8) [WOOD, BRICK, ORE, BRICK]} \end{aligned}$$

of which resource combination (3) and (6) can satisfy the construction; therefore, the construction cost is satisfiable and the player may construct the associated structure.

Name	Type	Description
staticResources	Resources	Resouces configuration produceable by the player
dynamicResources	ArrayList<Resources>	Resources granted from ResourceBonus effects which give a single resource from a set of resources
resourcesCombinations	ArrayList<Resources>	All possible resources permutations generated by the resources dynamicResources and staticResources
left	Neighbor	Resources and trading information for the player sitting to the left
right	Neighbor	Resources and trading information for the player sitting to the right

Figure 14: *Class* ResourcesManager *member variables*

4 Neighbor

The class *Neighbor* is a subclass of class *ResourcesManager*. *Neighbor* maintains the resources owned by the neighboring player and the associated prices of purchasing both the raw resources (wood, stone, ore, brick) and manufactured resources (glass, loom, papyrus).

Name	Type	Description
resources	ArrayList<Resources>	Resources granted from ResourceBonus effects
rawPrive	int	Cost in coins to purchase wood, stone, ore, or brick resources
manufacturedPrice	int	Cost in coins to purchase glass, loom or papyrus resources

Figure 15: *Class* Neighbor *member variables*

5 Conflict

The *Conflict* class is a helper to class *Player*. *Conflict* manages military conflicts between a player and his two neighbors. At the end of each age, each player enters military conflict with his two neighbors. During this part of the turn, player's shields which represent a player's military strength are compared. Having more shields than an opponent constitutes a victory while having less shields constitutes a defeat. Each defeat is wroth -1 victory points while a victory is worth 1, 3 or 5 victory points if the victory was achieved at the end of Age I, II, or III respectively.

Name	Type	Description
defeatCount	int	Number of defeats earned from losing military conflicts with neighbors
points	int	Points generated from victories and defeats during military conflicts
shields	int	Earned from ShieldBonus effects. Represents a player's military might.
victoryCount	int	Number of victories earned from winning military conflicts with neighbors

Figure 16: *Class* Conflict *member variables*

6 ScienceSymbols

The *ScienceSymbols* class is helper to class *Player*. *ScienceSymbols* manages all science symbols earned from ScienceBonus effects and points generated from combinations of science symbols. At the end of the game, science symbols generate 7 *victory points* for each set of scientific symbols and $x^2 + y^2 + z^2$ *victory points* where $x, y,$ and z is the number of each scientific symbol. The recursive algorithm *permuteScience* determines how *wild* symbols should be allocated to maximize points earned from scientific symbols. Figure 17 shows a sample configuration of science symbols and their associated scores.

Figure 17: *An example of the final scoring of a set of science symbols. Two full sets os symbols are each worth 7 points, while having 2 compass, 2 gear and 4 table symbols are worth 4, 4, and 16 points respectively for a total score of 38 points. The symbols with purple backgrounds indicate 'wild' symbols and were treated as'compass' and 'tablet' to maximize the science points.*

Name	Type	Description
compass	int	Number of SymbolA owned by the player
gear	int	Number of SymbolB owned by the player
tablet	int	Number of SymbolC owned by the player
points	int	Points earned from science symbols
wild	int	Number of symbols owned by the player which can be a gear, compass, or tablet

Figure 18: *Class* ScienceSymbols *member variables.*

Procedure 1 permuteScience($compass, gear, tablet, wild, maxScore$)

Input: The number of compass, gear, table, and wild science symbols earned by the player.

Output: Maximized points earned from science symbols.

```
if (wild = 0) then
    score ← 0
    score ← score + (compass)^2 + (gear)^2 + (tablet)^2 + (minimum(compass, gear, tablet) * 7)
    return score
end if
for i ← 0 to 2 do
    tempCompass ← compass
    tempGear ← gear
    tempTablet ← tablet
    if (i = 0) then
        tempCompass ← tempCompass + 1
    end if
    if (i = 1) then
        tempGear ← tempGear + 1
    end if
    if (i = 2) then
        tempTablet ← tempTablet + 1
    end if
    max = maximum(permuteScience(tempCompass, tempGear, tempTablet, wild − 1), max)
end for
return max
```

7 Constants

The constants interface *Constants* houses the set of enumerations used by the game: *Action*, *Adjacency*, *Age*, *Agent*, *Side*, *CardColor*, *EffectType*, *ObjectType*, *Science*, and *Trade*.

Actions

Actions available to the player. During each turn, he player may perform any one of the following actions: (i) construct a building from his hand (CONSTRUCT_STRUCTURE), (ii) construct the next stage of his winderboard (CONSTRUCT_STAGE), or (iii) discard a card from his hand (DISCARD_CARD).

Name	Value
Action	CONSTRUCT_STRUCTURE
	CONSTRUCT_STAGE
	DISCARD_CARD

Figure 19: *An enumeration of the* Action *object*

Adjacency

Determines which players to consider when determining effect resolution. Some effects which grant victory points or coins to the player determine the magnitude of their bonus based on the number of specific objects owned by a subset of the following three players: (i) the player (SELF), (ii) player's left neighbor (LEFT), and (iii) player's right neighbor(RIGHT).

Name	Value
Adjacency	LEFT
	RIGHT
	SELF

Figure 20: *An enumeration of the* Adjacency *object*

Age

Identifies the current Age. With only 3 Ages in a single game, the types of cards available each round, the direction hands are passed at the end of the turn, and the number of victory points received from military conflicts are determined by the current Age.

Name	Value
Age	ONE
	TWO
	THREE

Figure 21: *An enumeration of the* Age *object*

Agent

Classifies players based on the artificial intelligence which dictates their next chosen action. Each of the agents implement the following architectures: (i) rules-based expert system, (ii) perceptron, as well as (iii) a statiscal analysis of past game configurations.

Name	Value
Agent	HUMAN
	PERCEPT
	RANDOM
	RBAI_1
	RBAI_2
	STATISTIC

Figure 22: *An enumeration of the* Agent *object*

Side

Identifies which side of the wonderboard is currently in use. Each wonderboard has an 'A' and 'B' side. Though both sides are unique, both sides of a board cannot both be in play.

Name	Value
Side	A
	B

Figure 23: *An enumeration of the* Side *object*

CardColor

Identifies a card's and structure's color.

Name	Value
CardColor	BLUE
	BROWN
	GRAY
	GREEN
	PURPLE
	RED
	YELLOW

Figure 24: *An enumeration of the* CardColor *object*

EffectType

Identifies the type of benefit or bonus granted by an effect. Most effects grant a specific bonus to the player;such as, shields, coins, science symbols, resources, or victory points. Other effects influence the number of actions they can take per round or modify trade costs.

Name	Value
EffectType	BUILD_DISCARDED_CARD
	COPY_GUILD
	FREE_BUILD
	GOLD_BONUS
	PLAY_LAST_CARD
	RESORCE_BONUS
	SCIENCE_BONUS
	SHIELD_BONUS
	TRADE_BONUS
	VICTORY_POINT_BONUS

Figure 25: *An enumeration of the* EffectType *object*

ObjectType

Types of objects soe victory point and coin bonuses from effects are dependent on. Most effects give a static bonus while there are others whose bonus is dependent on the number of objects owned by a subset of the players in the game.

Name	Value
ObjectType	BLUE_CARD
	BROWN_CARD
	DEFEAT_TOKEN
	GRAY_CARD
	GREEN_CARD
	PURPLE_CARD
	RED_CARD
	VICTORY_TOKEN
	WONDER_STAGE
	YELLOW_CARD

Figure 26: *An enumeration of the* ObjectType *object*

Science

Types of science symbols granted by effects. Most effects grant a single specific symbol; while, some grant any one scientific symbol. This 'wild' symbol may count as any of the scientific symbols at the end of the game during scoring resolution.

Name	Value
Science	COMPASS
	GEAR
	TABLET
	WILD

Figure 27: *An enumeration of the* Science *object*

Trading

Types of trading oriented bonus granted by effects.

Name	Value
Trading	RAW_DISCOUNT
	MANUFACTURED_DISCOUNT

Figure 28: *An enumeration of the* Trading *object*

8 Structure

The *Structure* class is the object representation of the Age cards which form the game library, discard, and player hands. In figure 29, an example of the *Chamber of Commerce* card can be seen.

Figure 29: *The* Chamber of Commerce *card is a YELLOW structure with a resouce cost of [BRICK, BRICK, PAPYRUS] and provides a 2 coin and 2 victory point bonus for each GRAY structure previously built by the player.*

Name	Type	Description
id	int	Internal identification number
name	String	Structure name
age	Age	Age in which the card represented by this structure is added to the game library
color	CardColor	Structure color
resourceCost	Resources	Construction cost of the structure
effects	ArrayList<Effects>	Bonus and benefits provided to the player when this structure is constructed
frequency	ArrayList<Frequency>	Indicates the amount of copies of the card represented by this structure is added to the library
freeBuild	Structure	The structure if constructed mitigates the cost of the construction cost of this structure

Figure 30: *Class* Structure *member variables*

9 StructureDatabase

The class *StructureDatabase* holds all object data for each card within the *Seven Wonders* base game.

```
ChamberOfCommerce = new Structure(0x3F, "Chamber Of Commerce",Age.THREE, CardColor.YELLOW,
        new Resources.Builder().brick(2).papyrus(1).build());
ChamberOfCommerce.addEffect(new GoldBonus(2, ObjectType.GRAY_CARD, Adjacency.SELF));
ChamberOfCommerce.addEffect(new VictoryPointBonus(2, ObjectType.GRAY_CARD, Adjacency.SELF));
ChamberOfCommerce.addFrequency(6);
structures.add(ChamberOfCommerce);
```

Figure 31: *Object representation of the* Chamber of Commerce *card depicted in the previous section.*

Name	Type	Description
structures	ArrayList<Structures	Structures loaded from the database
guilds	ArrayList<Structure>	Guilds, structures with a card color of purple, loaded from the database

Figure 32: *Class* StructureDatabase *member variables*

10 Resources

The class *Resources* is the object representation of all construction costs and resources granted by Resource-Bonus effects.

Name	Type	Description
coins	int	Coins granted by this resource or required for this construction cost
wood	int	Wood granted by this resource or required for this construction cost
stone	int	Stone granted by this resource or required for this construction cost
brick	int	Brick granted by this resource or required for this construction cost
ore	int	Ore granted by this resource or required for this construction cost
glass	int	Glass granted by this resource or required for this construction cost
loom	int	Loom granted by this resource or required for this construction cost
papyrus	int	Papyrus granted by this resource or required for this construction cost

Figure 33: *Class* Resources *member variables*

11 Builder

A subclass of both the *Resources* and *ResouseBonus* classes, *Builder* is a helper class which allows for a simplified construction of construction costs and resources granted by ResourceBonus effects.

Name	Type	Description
resourceBonus	Resources	The construction cost or resources bonus by a ResourceBonus effect given by the structure or wonder stage
dynamic	boolean	Dictates if the ResourceBonus effect grants a choice of one resource from a subset of resources
tradable	boolean	Dicates if player's neighbors can purchase the resources granted by this ResourceBonus effect
wood	int	Wood granted by this ResourceBonus effect or required for this construction cost
stone	int	Stone granted by this ResourceBonus effect or required for this construction cost
brick	int	Brick granted by this ResourceBonus effect or required for this construction cost
ore	int	Ore granted by this ResourceBonus effect or required for this construction cost
glass	int	Glass granted by this ResourceBonus effect or required for this construction cost
loom	int	Loom granted by this ResourceBonus effect or required for this construction cost
papyrus	int	Papyrus granted by this ResourceBonus effect or required for this construction cost

Figure 34: *Class* Builder *member variables*

12 Wonderboard

The class *Wonderboard* is the object representation of the wonderboard. The wondeboard serves as the base of the player's city. Each wonderboard gives an initial resource to the player, and offers uniqe set of wonder stages. Figure 35 is an example of the Éphesos wonderboard. This wonderboard provides the player a unit of the *papyrus* resource and has three wonder stages.

Figure 35: *Side A of the Éphesos wonderboard.*

Name	Type	Description
id	int	Internal identification number
name	String	Wonderboard name
side	Side	Wonderboard side
resource	Resources	Resource provided by the wonderboard
stages	ArrayList<Stage>	Wonderboard stages

Figure 36: *Class* Wonderboard *member variables*

13 Stage

The class *stage* is an object representation of the wonders belonging to each wonderboard. Each wonder has a construction cost, and provides a bonus when constructed. Each wonder may only be constructed once and must be constructed in order: starting with the left most wonder and ending with the right most wonder. In theÉphesos wonderboard, the three wonders have the following cost and effect: (1) has a cost of 2 *stone* and gives 3 *victory points*, (2) has a cost of 2 *wood* and provides 9 *gold*, and (3) has a cost of 2 *papyrus* and provides 7 *victory points* when constructed.

Name	Type	Description
id	int	Internal identification number
resourceCost	Resources	Construction cost of the wonder stage
effects	ArrayList<Effect>	Bonus and benefits provided to the player when this wonder stage is constructed
built	boolean	Dictates whether the stage has been constructed

Figure 37: *Class* Stage *member variables*

14 StageDatabase

The class *StageDatabase* holds all object data for each wonder within the *Seven Wonders* base game.

```
EphesosA1 = new Stage(0x01, new Resources.Builder().stone(2).build());
EphesosA1.addEffect(new VictoryPointBonus(3));

EphesosA2 = new Stage(0x02, new Resources.Builder().wood(2).build());
EphesosA2.addEffect(new GoldBonus(9));

EphesosA3 = new Stage(0x03, new Resources.Builder().papyrus(2).build());
EphesosA3.addEffect(new VictoryPointBonus(7));
```

Figure 38: *Object representation of the wonders belonging to the* Éphesos *side A wonderboard depicted in the previous section.*

Name	Type	Description
stages	ArrayList<Stage	Stages loaded from the database

Figure 39: *Class* StageDatabase *member variables*

15 WonderboardDatabase

The class *WonderboardDatabase* holds all object data for each wonderboard within the *Seven Wonders* base game.

```
Wonderboard EphesosA = new Wonderboard(0x05, BoardSide.A, "Ephesos",
        new Resources.Builder().papyrus(1).build());
EphesosA.addStage(StageDatabase.EphesosA1);
EphesosA.addStage(StageDatabase.EphesosA2);
EphesosA.addStage(StageDatabase.EphesosA3);
wonderboards.add(EphesosA);
```

Figure 40: *Object representation of the* Éphesos *wonderboard depicted in the previous section.*

Name	Type	Description
wonderboards	ArrayList<Wonderboard	Wonderboards loaded from the database

Figure 41: *Class* WonderboardDatabase *member variables*

16 Effect

The *Effect* abstract class is the abstract representation of any bonuses and abilities granted by satisfying the construction cost of cards or wonders. There are many effect types: *BuildDiscardedCard*, *CopyGuild*, *FreeBuild*, *GoldBonus*, *PlayLastCard*, *ResourceBonus*, *ScienceBonus*, *ShieldBonus*, *TradeBonus*, and *VictoryPointBonus*, each of which is an extension of the *Effect* class. Common to all effects, each effect can only be used once.

Name	Type	Description
type	EffectType	Internal identification
effectTriggered	boolean	Denotes if the effect has been activated

Figure 42: *Class* Effect *member variables*

Figure 43: *The* Lighthouse*, an example of a card with a coin bonus effect. This structure gives one coin for each YELLOW structure constructed by the player.*

Name	Type	Description
adjacency	Adjacency	Player to check for the number of owned objects
coinBonus	int	Amount of coins given by the effect
multiplier	int	Amount of coins for each object given by the effect
object	ObjectType	Object type the effect's coin bonus is dependent on

Figure 44: *Class* CoinBonus *specific member variables*

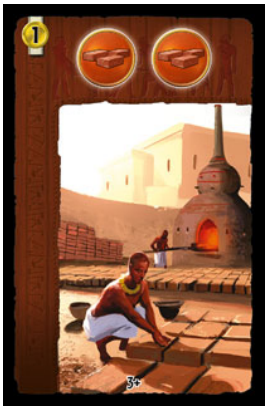

Figure 45: *The* Brickyard*, an example of a card with a resource bonus effect. This structure gives two BRICK resources, which are purchaseable by the player's neighbors.*

Name	Type	Description
resourceBonus	Resources	Resource bonus given by the effect
choice	boolean	Denotes if the resouce bonus grants only a single resource of a subset of resources
tradable	boolean	Denotes if the player's neighbors can purchase the resources granted by this effect

Figure 46: *Class* ResouceBonus *specific member variables*

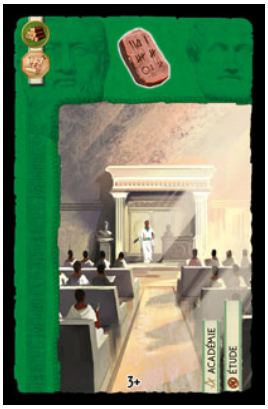

Figure 47: *The* School*, an example of a card with a science bonus effect. This structure gives a tablet science bonus.*

Name	Type	Description
scienceTypeBonus	Science	Science symbol bonus given by the effect

Figure 48: *Class* ScienceBonus *specific member variables*

Figure 49: *The* Walls*, an example of a card with a shield bonus effect. This structure gives three shields.*

Name	Type	Description
shieldBonus	int	Shields given by the effect

Figure 50: *Class* ShieldBonus *specific member variables*

Figure 51: *The* East Trading Post*, an example of a card with a trade bonus effect. This structure reduces the cost of purchasing raw materials from his right neighbor by one coin.*

Name	Type	Description
typeBonus	Trade	Trade bonus given by the effect
adjacency	Adjacency	Player to apply the trade discount to

Figure 52: *Class* TradeBonus *specific member variables*

Figure 53: *The* Senate*, an example of a card with a victory point bonus effect. This structure gives five victory points.*

Name	Type	Description
adjacency	Adjacency	Player to check for the number of owned objects
pointBonus	int	Amount of victory points given by the effect
multiplier	int	Amount of victory points for each object given by the effect
object	ObjectType	Object type the effect's point bonus is dependent on

Figure 54: *Class* VictoryPointBonus *specific member variables*

B. Artificial Intelligence

In this project, the following architecture's were considered: (i) rules-based expert systems, (iii) statistical analysis, (ii) perceptron, (iii) parallel alpha-beata search algorithms, and (iv) Monte Carlo tree searching. The algorithms and specifications for each algorithm is as follows.

1 Randomized Cards

To serve as a starting point to test the effectiveness of future algorithms, *RANDOM* was designed to perform randomized actions. *RANDOM* will construct a random structure from a set of playable cards. If no cards are playable, *RANDOM* will attempt to construct his wonder, if unable to do so, he will discard a random card.

Procedure 2 RANDOM(long seed, ArrayList<Structure> hand)

Input: A seed value to fix the random number generation to ensure results are reproduceable.

Output: A proposed **action** and **structure**.

```
for (Structure s : hand) do
    if s.isPlayable() then
        playable.add(s)
    end if
end for
if (!playable.isEmpty()) then
    shuffle(playable)
    action ← Action.CONSTRUCT_STRUCTURE
    structure ← playable.getFirstElement()
else if (wonderBoard.getNextStage().isPlayable) then
    action ← Action.CONSTRUCT_STAGE
    structure ← playable.getFirstElement()
else
    shuffle(hand)
    action ← Action.DISCARD_CARD
    structure ← playable.getFirstElement()
end if
return
```

2 Rules-Based Expert System

The first attempt at developing an artificial intelligence, the *Rules-based Expert System* offers a simple series of nested IF THEN statements which serves as a set of rules based on expert advice. Each rule serves as a hierarchy of conditions stressing the construction of a subset of structures:

Rule 1: $(\forall c)(\exists r_1)(\exists r_2)\big[\text{Card}(c) \wedge \text{Resource}(r_1, c) \wedge \text{Resource}(r_2, c) \wedge r_1 \neq r_2\big]$

Rule 2: $(\forall c)(\forall n)(\exists r)\big[\text{Card}(c) \wedge \text{Neighbor}(n) \wedge \text{Resource}(r, c) \wedge \neg\text{Produce}(r) \wedge \neg\text{Produce}(n, r)\big]$

Rule 3: $(\forall c)(\forall n)\big[\text{Card}(c) \wedge \text{Neighbor}(n) \wedge \text{Shields}(x) \wedge \text{Shields}(n, z) \wedge \text{Shields}(c, y) \wedge (x + y) > z\big]$

Rule 4: $(\exists c_1)(\forall c_2)\big[\text{Card}(c_1) \wedge \text{Card}(c_2) \wedge \mathit{Points}(c_1, x) \wedge \mathit{Points}(c_2, y) \wedge x \geq y\big]$

Rule 5: $(\forall c)\big[\text{Card}(c) \wedge \text{Color}(c, x) \wedge x = \text{`GREEN'}\big]$

Procedure 3 Rules-based Expert System(long seed, ArrayList<Structure> hand)

Input: A seed value to fix the random number generation to ensure results are reproduceable.

Output: A proposed **action** and **structure**.

```
for (Structure s : hand) do
    if s.isPlayable() then
        playable.add(s)
    end if
end for
if (!playable.isEmpty()) then
    structure ← RuleOne(playable)
    if (RuleOne(playable) != null) then
        action ← Action.CONSTRUCT_STRUCTURE return
    end if
    structure ← RuleTwo(playable)
    if (RuleTwo(playable) != null) then
        action ← Action.CONSTRUCT_STRUCTURE return
    end if
    structure ← RuleThree(playable)
    if (RuleThree(playable) != null) then
        action ← Action.CONSTRUCT_STRUCTURE return
    end if
    structure ← RuleFour(playable)
    if (RuleFour(playable) != null) then
        action ← Action.CONSTRUCT_STRUCTURE return
    end if
    structure ← RuleFive(playable)
    if (RuleFive(playable) != null) then
        action ← Action.CONSTRUCT_STRUCTURE return
    end if
end if
if (wonderBoard.getNextStage().isPlayable) then
    action ← Action.CONSTRUCT_STAGE
    structure ← playable.getFirstElement()
else
    shuffle(hand)
    action ← Action.DISCARD_CARD
    structure ← playable.getFirstElement()
end if
return
```

3 Statistical Analysis

During experimentation, certain structures were observed to have been consistently constructed. Furthermore, these structures were oftentimes constructed by players who scored particularly high in the game. Our last agent monitors the results of past games and assigns a weight to each structure indicating its desireability. A structure constructed by the winner will receive a higher level of desireability compared to lower level players. During future playthoughs, the agent will select the structures with large desireability values.

Procedure 4 Learner(long seed, ArrayList<Structure> hand, $ArrayList < Integers > desireabilityIndex$)

Input: A desireability index which indicates an order of precedence of which structures to first construct

Output: A proposed **action** and **structure**.

```
for (Structure s : hand) do
    if s.isPlayable() then
        playable.add(s)
    end if
end for
maxIndex ← 0
if (!playable.isEmpty()) then
    for (Structure s : playabale) do
        if (s.getDesireability() ≥ maxIndex) then
            structure ← s
        end if
    end for
    action ← Action.CONSTRUCT_STRUCTURE
else if (wonderBoard.getNextStage().isPlayable) then
    action ← Action.CONSTRUCT_STAGE
    structure ← playable.getFirstElement()
else
    shuffle(hand)
    action ← Action.DISCARD_CARD
    structure ← playable.getFirstElement()
end if
return
```

IV. Results

The following data serves as an evaluation of the performance for each agent. Statistical information regarding scores as well as placement information is shown.

Agent	Mean	Low	High	Standard Deviation	1st	2nd	3rd	4th	5th
RBAI	38	10	64	7.6	0.19	0.22	0.22	0.20	0.17
LEARNER	38	11	69	8.0	0.45	0.24	0.20	0.17	0.15

Figure 55: *Agent scoring information over 10,000 trials, 1 Agent vs 4 RANDOM. Both agents performed comparably when facing* RANDOM *opponents.*

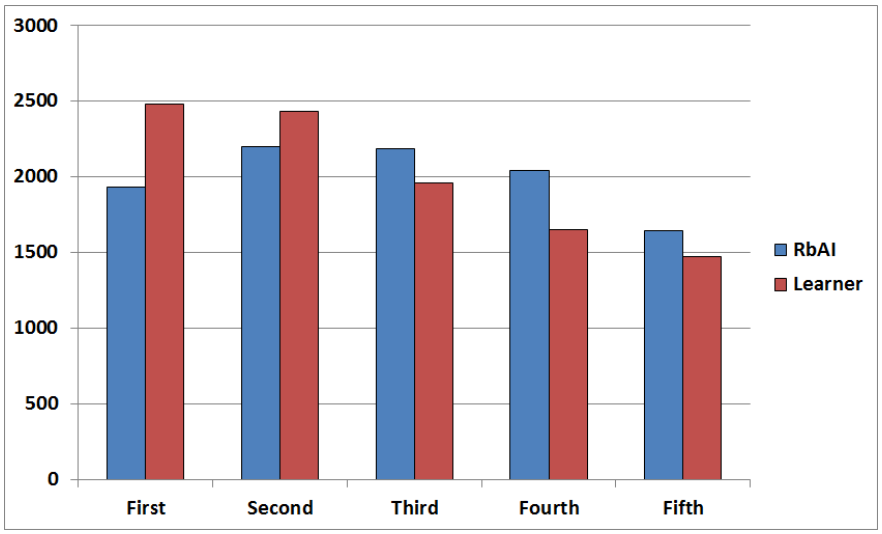

Figure 56: *Agent scoring information over 10,000 trials, 1 Agent vs 4 RANDOM. Both agents performed comparably when facing* RANDOM *opponents.*

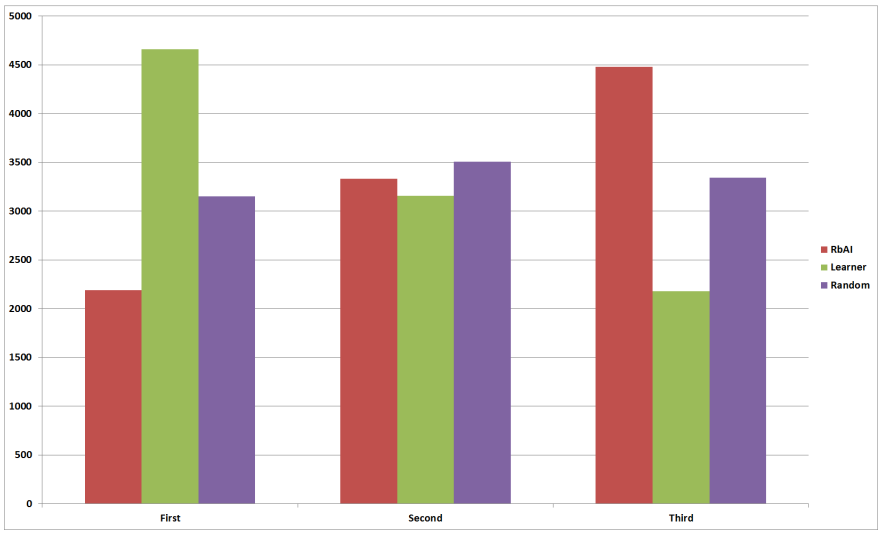

Figure 57: *Agent placement results over 10,000 trials, RBAI vs LEARNER vs RANDOM.*