UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

INSTITUTO DE MATEMÁTICA

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS RODRIGUES CARNEIRO

ANÁLISE DE COMPORTAMENTOS SOCIAIS EM AGENTES DE JOGOS

RIO DE JANEIRO

2018

LUCAS RODRIGUES CARNEIRO

ANÁLISE DE COMPORTAMENTOS SOCIAIS EM AGENTES DE JOGOS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Carla A. D. M. Delgado, D.Sc.

Co-orientador: Prof. João Carlos Pereira da Silva, D.Sc.

RIO DE JANEIRO

2018

CIP - Catalogação na Publicação

Carneiro, Lucas Rodrigues
C289a Análise de comportamentos sociais em agentes de jogos / Lucas Rodrigues Carneiro. -- Rio de Janeiro, 2018.
90 f.

Orientadora: Carla Amor Divino Moreira Delgado.
Coorientador: João Carlos Pereira da Silva.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2018.

1. Confiança e reputação. 2. Agentes sociai Jogos. I. Delgado, Carla Amor Divino Moreira, orient. II. da Silva, João Carlos Pereira, coorient. III. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a).

LUCAS RODRIGUES CARNEIRO

ANÁLISE DE COMPORTAMENTOS SOCIAIS EM AGENTES DE JOGOS

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 14 de setembro de 2018.

BANCA EXAMINADORA:

Prof. Carla A. D. M. Delgado, D.Sc.

Prof. João Carlos Pereira da Silva, D.Sc.

Prof. Geraldo Bonorino Xexéo, D.Sc.

Prof. Ana Luisa C. L. Duboc, D.Sc.

Prof. Eduardo F. M. de Brito, M.Sc.

AGRADECIMENTOS

Agradeço a minha família, e em especial a minha mãe, que sempre me ajudou durante todo o meu percurso de todas as maneiras possíveis e impossíveis; aos meus colegas de curso, que fizeram com que a vida universitária fosse mais suportável e divertida, e que também me ajudaram com todos os ensinamentos que eu não compreendi ou simplesmente perdi; a todos os geniais professores com que pude contar para moldar meu caráter e conhecimentos acadêmicos, além de se tornarem amizades bem-vindas; aos meu orientadores, Profa. Dra. Carla Amor Divino Moreira Delgado e Prof. Dr. João Carlos Pereira da Silva pelos ensinamentos, orientações, críticas e elogios feitos durante a caminhada deste projeto, e também por terem me proporcionado ótimas aulas durante minha graduação; à banca avaliadora, que se dispôs a participar deste projeto e enriqueceu meus estudos com as dicas, sugestões e críticas feitas durante a defesa.

Também preciso fazer um agradecimento especial ao Q4, pois sem estes grandes amigos que pude cultivar desde 2009 e que continuaram fazendo parte da minha caminhada acadêmica desde então, eu não teria conseguido chegar tão longe, e talvez até tivesse desistido deste caminho. Esta vitória não é só minha, mas de vocês também. Muito obrigado por tudo.

Por fim, um agradecimento a você leitor, que se dispôs a ler este trabalho. Espero que ele possa lhe acrescentar algo, da mesma maneira que fez comigo.

RESUMO

Jogos Eletrônicos costumam utilizar técnicas da área de Inteligência Artificial para melhorar o comportamento de personagens não jogáveis (NPCs), entregando uma experiência mais realística ao usuário. Entretanto, este comportamento não costuma levar em consideração as interações entre personagens e jogadores. Por conta disto, foi desenvolvida uma nova abordagem para estes agentes, que se utiliza de um viés social, criando uma estratégia que não se baseia apenas em vencer todos os jogadores, mas também em analisá-los e decidir se eles são aliados ou adversários. Foram usados modelos de confiança e reputação em conjunto com técnicas de Inteligência Artificial para permitir este comportamento social. O resultado obtido manteve a competitividade dos agentes, e conseguiu criar uma nova experiência para o jogador. Este trabalho continua esta proposta, analisando o viés social construído, propondo mudanças no seu funcionamento, e fazendo uma análise de dados em cima de simulações com agentes e jogadores humanos para concluir se foi possível ou não melhorar seus resultados, e entender a opinião dos jogadores sobre estes agentes.

Palavras-chave: Agentes sociais. Confiança e reputação. Jogos.

ABSTRACT

Video games normally use Artificial Intelligence techniques to improve NPC behavior, creating a more realistic experience for their players. However, this behavior does not consider interactions between player and bots. Because of that, a new approach for NPCs was proposed, which uses a social bias to mix the default strategy of finding best plays to win against all players with a player analysis to decide if they should be categorized as allies or foes. Trust and reputation models were used together with AI to implement this demeanor. The agents which were created not only maintained their competitiveness but also created a new player experience. This project continues the cited work, analyzing the social bias used, proposing improvements for it, and doing a data analysis on the results obtained from simulations with agents and players to conclude if it was possible or not to improve the results shown on the cited work, together with understanding players' opinions about these agents.

Keywords: Games. Social agents. Trust and reputation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

UFRJ	Universidade Federal do Rio de Janeiro
NPC	*Non-player Character*
IA	Inteligência Artificial
ETB	*Estimated Time to Build*
ETW	*Estimated Time to Win*
XML	*Extensible Markup Language*
CO	Cooperativo
AS	Antissocial
SO	Social

LISTA DE SÍMBOLOS

s	Segundos
KB	Kilobytes

SUMÁRIO

1 INTRODUÇÃO

Videogames são uma das fontes de entretenimento mais modernas da humanidade, além de ser uma das mais rentáveis. Existem diferentes tipos de gêneros de jogos, mas muitos deles tem uma característica em comum: Se utilizam de técnicas de Inteligência Artificial (IA) para introduzir NPCs (Non-Player Character) que podem cooperar com o jogador, fazer o papel de inimigos, ou simplesmente existir no jogo como cenário.

Por conta da quantidade de maneiras diferentes que pode ser usada dentro de jogos, a área de IA tem sido amplamente estudada com foco em jogos, e tem evoluído com o passar das gerações de jogos lançados. Essa evolução tem sido percebida pelos jogadores, e é vista com satisfação pelos mesmos.

Com os resultados obtidos pela indústria, seja em sucesso de crítica ou em retorno financeiro, tudo indica que o objetivo de criar uma inteligência mais realista que a atual não terminará tão cedo.

1.1 MOTIVAÇÃO

Dentro dos diversos gêneros de jogos existentes, também é possível notar uma tendência a jogos com dinâmicas sociais, aonde é necessária a interação entre diferentes jogadores para chegar num objetivo maior, seja em experiências competitivas ou cooperativas. Apesar disso, muitos jogos se utilizam de IA com estratégias focadas na vitória, como por exemplo os algoritmos Minimax [3] e *Monte Carlo Tree Search* [1].

Isso acaba afetando a experiência do jogador, que não sente muito realismo nas atitudes de NPCs nessas situações. Em um mesmo tipo de jogo, mas com outros jogadores humanos, é possível notar como as interações muitas vezes não são tomadas apenas com um viés lógico de obter a vitória, mas acabam sendo afetadas pela percepção do jogador sobre os outros participantes da partida.

Existe uma carência no mercado para jogadores artificiais que se aproveitem

desta característica social, que poderia transformar a experiência do jogador em algo mais realístico e mais interessante.

1.2 OBJETIVO

O objetivo deste projeto foi de continuar os estudos nessa área, se utilizando de sistemas de confiança e reputação para conseguir replicar o comportamento humano nesse tipo de situação. Para testar os resultados obtidos e fazer experimentos com jogadores humanos para analisar se foi possível conseguir um comportamento mais realístico ou não, foi utilizado o jogo *Settlers of Catan*[1], jogo de tabuleiro internacionalmente conhecido desenvolvido pelo alemão Klaus Teuber.

1.3 TRABALHOS CORRELATOS

Este projeto é uma continuação direta da dissertação de mestrado **Confiança e Reputação para Jogos** [2], se utilizando das mesmas tecnologias e experimentando diferentes maneiras para melhorar o desempenho dos agentes desta dissertação.

Além disso, também existem outros trabalhos na literatura que abordam o mesmo tema. Em Mascarenhas et al. [8], é desenvolvido um modelo para construção de agentes com uma configuração cultural, influenciando suas decisões e percepções sobre outros agentes e jogadores. Essa configuração cultural define o quão importante outros indivíduos são para um agente, usando fatores como relação pessoal, hierarquia, senso comum, e outros. Ao definir a importância de outros indivíduos, o agente é capaz de definir se tomará uma ação positiva ou negativa para cada um.

Em Osborne [9], é proposto um algoritmo de escolha de ações que define o comportamento de um agente por meio de três camadas: autopercepção, ambiente social e comunicação interpessoal. Para cada camada, existem várias opções de personalidades diferentes, que podem ser usadas em grupo ou individualmente. Logo, cada agente é composto por um conjunto de personalidades em cada camada, que definem

[1] https://www.catan.com/

suas atitudes. Desta maneira, é possível criar diferentes combinações que formem agentes que sejam antissociais, competitivos, neutros, e outros, inclusive permitindo agentes com comportamentos mistos.

Como último exemplo, podemos citar Tan e Cheng [10], que implementaram um *framework* chamado *Tactical Agent Personality*, onde agentes mudam de comportamento baseando-se na personalidade do jogador ou de outros agentes por meio de um sistema de aprendizado de recompensa e punição. Essa adaptação não ocorre a cada ação, mas sim em outros momentos do jogo onde o agente não está tomando ações. Isso faz com que o processamento do jogo seja menos impactado do que em casos onde é necessário se adaptar a cada ação, e por conta da possibilidade de se adaptar à personalidade do jogador (que é definida pelo conjunto de ações que ele toma durante o jogo entre suas opções possíveis), é possível fazer com que os agentes se comportem de maneira mais agradável para cada tipo de jogador.

1.4 ESTRUTURA

No capítulo 2, é explicado o conceito de sistemas de confiança e reputação, com foco no sistema utilizado no projeto, o *Beta Reputation System* [7]. Além disso, também é apresentada a implementação do mesmo, juntamente com o uso de memória.

O Capítulo 3 é focado em definir as regras e funcionamento do jogo Catan, ilustrando os pontos existentes de interação entre jogadores. Por fim, também é apresentado neste capítulo a implementação do jogo por meio do *software* JSettlers [11].

A seguir, temos no capítulo 4 uma apresentação geral dos agentes existentes no projeto, que engloba as atitudes de cada um diante dos diferentes pontos de interação do jogo, a implementação utilizada para cria-los, e as modificações feitas para melhorar o desempenho dos mesmos.

No Capítulo 5, todos os experimentos feitos são apresentados. Também são explicadas as métricas utilizadas, e é feita uma análise dos resultados obtidos. Es-

ses experimentos são divididos em duas categorias: simulações feitas apenas entre agentes, e simulações feitas com a presença de um jogador humano.

Finalizando, o capítulo 6 conclui o projeto, trazendo uma reflexão sobre o trabalho como um todo, e faz uma discussão sobre possíveis avanços futuros na área.

2 CONFIANÇA E REPUTAÇÃO

Confiança e Reputação são dois conceitos tão antigos quanto a noção de sociedade. Ambos são utilizados de maneira direta e/ou indireta nas relações entre diferentes pessoas, e são métricas importantes para definir se uma interação social é segura ou não. Essas interações podem acontecer em diferentes meios, incluindo situações digitais.

Antes de partimos para os sistemas de confiança e reputação existentes na área de computação, é necessário entender bem o que são esses dois conceitos. Numa definição de Jøsang [6], confiança é um relacionamento direcional entre dois agentes: quem confia (*trustor*) e quem é confiado (*trustee*), onde essa confiança é construída pelo agente que confia com base em informação externa e/ou experiências passadas. No caso de reputação, este conceito é diretamente ligado à confiança, e é definido como a qualidade ou caráter geral de um agente ao ser visto ou julgado por outros agentes. A diferença entre confiança e reputação pode ser ilustrada por meio de duas frases:

1. "Eu confio em você por causa da sua boa reputação."

2. "Eu confio em você apesar da sua má reputação."

Enquanto que no primeiro caso temos uma situação onde um agente conhece a reputação de outro e a utiliza para construir uma relação de confiança nele, o segundo caso ilustra uma situação onde um agente conhece a reputação do outro, e ela sozinha seria um motivo para não existir confiança entre os dois, só que por conta de outras informações e experiências que o agente possui (e que possuem um peso maior do que a reputação citada), ele considera que pode estabelecer confiança com o outro.

Dentro da área de computação, sistemas de confiança e reputação são uma maneira comum de utilizar estes conceitos para ajudar usuários a decidir se podem interagir de maneira positiva com outros usuários desconhecidos. Sistemas de confiança constroem uma nota que representa a credibilidade de um agente de maneira

subjetiva, enquanto que sistemas de reputação normalmente constroem uma nota pública sobre um agente por meio de eventos sofridos por todos os membros do sistema. Apesar disso, existem sistemas de confiança que possuem características de sistemas de reputação e vice-versa, o que faz com que essas duas definições se misturem em certos casos.

Neste projeto, o papel dos sistemas de confiança e reputação foi de, por meio das interações entre os jogadores em uma partida, fazer com que os agentes sociais tivessem uma métrica para decidir se cada um dos jogadores é confiável ou não, categorizando-os como aliados ou adversários. Os sistemas de recomendação e confiança utilizados neste projeto são o *Beta Reputation System*, e um simples sistema de confiança criado por do Couto [2]. Ambos foram escolhidos e utilizados por do Couto em sua tese, e continuaram a ser utilizados neste projeto.

2.1 BETA REPUTATION SYSTEM

Este sistema, proposto por Jøsang e Ismail [7], se baseia numa função de distribuição de probabilidade, que utiliza duas variáveis (α e β) em uma série de funções gama para definir as probabilidades dos valores da função de probabilidade beta. Isso é possível porque probabilidades de eventos binários (como vários casos de jogos, *e-commerce*, etc.) podem ser representadas por distribuições beta.

A esperança do valor da distribuição beta é dada por:

$$E(p) = \frac{\alpha}{\alpha + \beta} \tag{2.1}$$

Onde temos que:

$$p \neq 0, \text{ se } \alpha < 1 \tag{2.2}$$

$$p \neq 1, \text{ se } \beta < 1 \tag{2.3}$$

Para definirmos esses valores de uma maneira mais clara, temos que:

$$\alpha = r + 1 \tag{2.4}$$

$$\beta = s + 1 \tag{2.5}$$

Onde r é o número de "sucessos", e s é o número de "falhas". Num exemplo sobre interações sociais entre duas pessoas, temos que r seria o número de interações positivas entre os dois, enquanto que s seria o número de interações negativas. Então, usando essa substituição em α e β, podemos definir a esperança anterior como:

$$E(p) = \frac{r + 1}{r + s + 2} \tag{2.6}$$

O que nos dá um resultado variável entre o intervalo $]0, +1[$.

2.2 SISTEMA DE CONFIANÇA

O sistema de confiança deste projeto é o mais simples possível, consistindo apenas em definir se um jogador em questão é confiável ou não, sem retornar nenhum atributo extra. Para responder esta pergunta, utilizamos apenas o resultado de reputação de um usuário. Entretanto, por conta do seu intervalo ser $]0, +1[$, o mesmo é convertido para o intervalo $]-1, +1[$, fazendo com que possamos ter três possibilidades: confiança negativa, neutra e positiva. Com a conversão, a equação 2.6 é usada no cálculo de confiança da seguinte maneira:

$$\text{Confiança} = (E(p) - 0.5) \times 2 \tag{2.7}$$

Expandindo e fazendo algumas operações, obtemos a equação final:

$$\text{Confiança} = \frac{(r - s)}{(r + s + 2)} \tag{2.8}$$

2.3 IMPLEMENTAÇÃO

Para implementar estes sistemas de confiança e reputação, foram criadas duas classes no projeto: *BetaReputation* e *Reputations*. Ambas ficaram responsáveis pelo armazenamento de todas as interações entre jogadores, cálculo de reputação e conversão para confiança, além de possuírem funções que permitem que qualquer agente que vá utilizar confiança dentro da sua tomada de decisão possa facilmente obter e atualizar os valores de qualquer outro jogador.

Código 2.1: Classe *BetaReputation* (Java)

```
public class BetaReputation {
        /* Positive ratings */
        private int r;
        /* Negative ratings */
        private int s;

        public BetaReputation()
        {
                r = 0;
                s = 0;
        }

        public void Rate(boolean rating, int weight)
        {
                if (weight <= 0)
                        weight = 1;

                if (rating)
                        r += weight;
                else
                        s += weight;
        }

        public float ReputationLevel()
        {
                return ((float)r - (float)s)/((float)r + (float)s +
                    2);
        }
        public int getPositiveRating() {
                return r;
        }
        public int getNegativeRating() {
                return s;
        }
}
```

Como podemos ver na classe *BetaReputation*, uma instância dela é inicializada com nenhuma interação, mas pode facilmente ser modificada por meio da função *Rate*, que recebe dois parâmetros: o primeiro, chamado *rating*, define se a interação é positiva ou negativa, enquanto o segundo, que é chamado de *weight*, define o peso dessa interação (o que permite com que cada agente possua uma estratégia onde cada tipo de interação tem uma importância diferente para sua estratégia).

Além disso, ela possui três funções para a obtenção de dados: *ReputationLevel*, *getPositiveRating*, e *getNegativeRating*. Com a primeira função, é possível conseguir o valor de reputação já transformado pelo sistema de confiança para o intervalo $]-1, +1[$, enquanto que as outras duas funções permitem com que qualquer outro objeto possa obter a quantidade de interações positivas e negativas que já aconteceram. Estas últimas duas funções são usadas dentro do projeto para ser possível reproduzir um valor de confiança em outra partida e/ou em outro jogador, já que a classe não permite uma inserção de valor de confiança diretamente, apenas pelo uso da função *Rate* (que só aceita interações como parâmetro).

Código 2.2: Classe *Reputations* (Java)

```
public class Reputations {

        HashMap<String, BetaReputation> agentsReputation;
        String nome;

        public Reputations(String oNome)
        {
                agentsReputation = new HashMap<String,
                    BetaReputation>();
                nome = oNome;
        }

        public void AddAgent(String identifier)
        {
                agentsReputation.put(identifier, new BetaReputation
                    ());
        }

        public void RateAgent(String identifier, boolean rating,
            int weight)
        {
                if (agentsReputation == null)
                        return;

                if (!agentsReputation.containsKey(identifier))
                        return;

                agentsReputation.get(identifier).Rate(rating,
                    weight);
        }

        public void RateAgent(String identifier, boolean rating)
        {
                RateAgent(identifier, rating, 1);
        }
    //...
}
```

Enquanto que a classe *BetaReputation* possui a lógica necessária para o cálculo e consulta de reputações e confianças, a classe *Reputations* faz o controle a este acesso, além de possuir funções para adicionar jogadores na lista de confiança e retornar um texto formatado com os valores de confiança e interações para cada jogador.

Cada agente dentro da partida possui sua própria instância desta classe, e ela possui um *hashmap* de instâncias da classe *BetaReputation*, onde existe uma instância para cada um dos outros jogadores. Desta maneira, no caso máximo onde possuímos 4 agentes que fazem uso de confiança e reputação numa partida, teremos 12 instâncias da classe *BetaReputation*, divididas igualmente entre 4 instâncias desta classe.

Dentro da classe, temos duas funções para adicionar interações entre agentes, e ambas possuem o mesmo nome: *RateAgent*. Enquanto que uma versão aceita um parâmetro extra que define o peso da interação (*weight*), a outra só possui os dois parâmetros originais. Um é *identifier*, que define qual foi o jogador que participou da interação, e o outro é *rating*. Nesta versão, o valor de *weight* é considerado como 1. O funcionamento de ambas versões é igual, consistindo em encontrar dentro do seu *hashmap* a instância de *BetaReputation* correspondente ao jogador da interação e chamar sua função *Rate*.

Além disso, temos uma função que permite adicionar jogadores no seu *hashmap* de instâncias da classe *BetaReputation*, uma função para retornar o valor de confiança de um jogador específico e uma última função que retorna os valores de confiança e quantidade de interações de todos os jogadores, em uma *string* formatada para ser lida dentro do arquivo de log da partida.

É importante frisar que, nesta implementação, não existe nenhum tipo de comunicação ou consulta entre os jogadores para definir as reputações e confianças da partida, então cada agente que faz uso destes sistemas na sua estratégia só leva em consideração as interações que aconteceram com eles mesmos, sem verificar os valores de reputação e confiança que os outros agentes podem possuir armazenados.

Por conta disso, podemos afirmar que as reputações e confianças de cada um dos jogadores são independentes entre si.

2.4 MEMÓRIA

Em sistemas de reputação e confiança, nem sempre é relevante guardar informações antigas sobre um agente, já que o mesmo pode mudar seu comportamento durante o tempo [7]. Por conta disso, nessas situações é necessário implementar uma maneira de diminuir o peso de informações antigas, até o ponto onde elas se tornam irrelevantes. Este conceito de esquecimento pode ser usado em várias situações diferentes, além de possuir diferentes implementações, como por exemplo uma proposta por Jøsang [5].

Neste trabalho, também existe este conceito de esquecimento, mas ao invés de fazermos o uso de equações complementares no cálculo de reputação, ele acontece por conta da independência entre partidas, já que nenhuma informação sobre uma partida anterior é utilizada em uma partida seguinte. Por conta disso, foi proposta a ideia de implementar um conceito de memória, onde fosse possível começar uma partida com valores de reputação e confiança de uma partida anterior, ou até mesmo valores novos para fins de teste.

Esta memória pode ser implementada de várias maneiras. Algumas das possibilidades são:

- Ter uma função de inicialização de reputação que permita valores diferentes de 0;
- Armazenar o valor final de reputação de um jogador em um banco local ou arquivo de texto, que será lido em uma nova partida que contenha o mesmo jogador, e será usado como reputação inicial, como se se fosse uma continuação da partida anterior;
- Ao final de cada partida, verificar qual foi o posicionamento de cada jogador (positivo, negativo, neutro), e com base nisso, inicializar a reputação de cada

jogador com um valor padrão (ex: -1 para negativo, 0 para neutro, +1 para positivo).

Por questões de tempo e simplificação, a opção escolhida neste projeto foi criar uma função de inicialização de reputação. Podemos ver um trecho da sua implementação abaixo.

Código 2.3: Função *TurnReputationsOn* (Java)

```
public void TurnReputationsOn(int[] reputations)
{
  useReputation = true;
  this.reputations = new Reputations(name);

  // Clona o array, para nao modificar a ordem usada em todos os
     outros lugares do simulador
  SOCPlayer[] players = this.game.getPlayers().clone();

  // Ordena os jogadores pelo nome, ao inves de usar playerNumber
  Arrays.sort(players, new Sortbyname());

  int i = 0;
  for (SOCPlayer p : players){
    if (p.getPlayerNumber() != playerNumber){
      this.reputations.AddAgent(p.name);
      if (reputations.length == 3){
        if (reputations[i] != 0){
          boolean rating = reputations[i] > 0;
          this.reputations.RateAgent(p.name, rating, Math.abs(
             reputations[i]));
        }
        i++;
      } else { // length = 6
        if (reputations[i] != 0)
          this.reputations.RateAgent(p.name, true, reputations[i]);
        if (reputations[i+1] != 0)
          this.reputations.RateAgent(p.name, false, reputations[i
             +1]);
        i += 2;
      }
    }
  }
  System.out.println("[TRDEBUG:REPUTATIONSTATUS] " + this.
     reputations.getStatus());
}
```

Além de termos esta função que permite uma inicialização de reputação para os jogadores diferente de zero, foi decidido permitir que esses valores fossem passados por parâmetro ao executar o projeto. Para isto, basta passar um vetor de reputações como parâmetro, que contenha 3 ou 6 reputações, dependendo se a inicialização é feita apenas com um tipo de interação (positiva ou negativa) ou com ambos. Ao passar mais de um vetor, o programa irá inicializar as reputações do segundo agente da partida que utilize reputações, e assim sucessivamente, sempre respeitando a ordem dos agentes dentro da partida.

Por exemplo, num caso onde uma partida fosse iniciada com o vetor [10,-10,0], a lista de reputações do primeiro agente que utilize reputações será inicializada com 10 interações positivas em relação ao primeiro jogador da partida, 10 interações negativas em relação ao segundo, e não inicializará as interações do último jogador, mantendo o padrão de zero. Este vetor teria o mesmo efeito de passar por parâmetro o vetor [10,0,0,10,0,0], já que a ordem do vetor é interação positiva, seguida de interação negativa.

Desta maneira, é possível começar uma partida com reputações e confianças escolhidas de maneira arbitrária para fazer diferentes testes, ou mesmo começar uma partida com os mesmos valores de uma partida anterior, bastando apenas verificar no arquivo de log da partida anterior qual foi o valor final das reputações armazenadas por cada um dos agentes que fazem uso disso.

3 CATAN

Settlers of Catan, também conhecido apenas por Catan, é um jogo de tabuleiro alemão, criado em 1995 por Klaus Teuber. Ao participar de uma partida, os jogadores são levados para a ilha de Catan, que é composta de 19 terrenos rodeados pelo mar. O objetivo de cada jogador é colonizar a ilha e expandir seus territórios para se tornar o maior colonizador de Catan.

Dentro da ilha, existem 5 tipos diferentes de terrenos, onde cada um produz um tipo diferente de matéria-prima. Além desses terrenos, também existe um deserto, que não produz nada.

Terreno	**Recurso**
Floresta	Madeira
Colina	Tijolo
Pasto	Ovelha
Lavoura	Trigo
Montanha	Minério
Deserto	–

Tabela 1: Terrenos e Recursos Correspondentes.

No início do jogo, cada jogador começa com 2 aldeias e 2 estradas, onde cada aldeia vale 1 ponto. O primeiro jogador a chegar em 10 pontos na sua rodada vence o jogo. Para conseguir mais pontos, é preciso construir novas estradas e novas aldeias. Também é possível conseguir pontos transformando aldeias em cidades. Cada cidade corresponde a 2 pontos. Entretanto, para criar construções, é necessário obter as matérias-primas necessárias para cada tipo de construção.

Figura 1: Exemplo de tabuleiro de Catan.

A cada início de rodada, o jogador correspondente lança dois dados, que definem quais são os terrenos que vão produzir nesta rodada. Cada terreno tem um número para tal, variando entre todas as possibilidades de valores de uma dupla de dados (2 a 12). Se um jogador possuir uma aldeia ao redor de um terreno que produziu no turno em questão, ele irá receber um recurso correspondente ao tipo do terreno. No caso de possuir uma cidade, o jogador receberá dois recursos.

Caso o valor dos dados em conjunto dê o valor que representa o deserto (7), o jogador do turno em questão deve mover o ladrão, peça que começa a partida no terreno do deserto. O local que se tornar a nova moradia do ladrão não produz recursos se o seu número sair, e se no momento da movimentação do ladrão existirem quaisquer cidades ou aldeias de algum jogador, o jogador do turno poderá roubar aleatoriamente um recurso do dono da construção. Em caso de mais de um jogador possuir construções naquele terreno, o jogador do turno escolhe apenas uma pessoa para ser roubada. Além disso, todos os jogadores que possuírem 8 ou mais recursos no turno em que for tirado 7 deverão descartar metade dos seus recursos em mão, arredondando para baixo.

Na tabela abaixo, podemos ver as matérias-primas necessárias para cada tipo de

construção possível.

	Recursos				
Construção	**Madeira**	**Tijolo**	**Ovelha**	**Trigo**	**Minério**
Estrada	1	1	-	-	-
Aldeia	1	1	1	1	-
Cidade	-	-	-	2	3
Carta de Desenvolvimento	-	-	1	1	1

Tabela 2: Construções e Recursos Necessários.

Como pode ser visto na tabela 2, além das construções previamente citadas, também existem cartas de desenvolvimento, que podem ser compradas ao serem construídas e possuem diferentes efeitos possíveis. Os efeitos são:

- Cavaleiro: Permite mover o ladrão e ativar seu efeito de roubo;
- Monopólio: Permite escolher um recurso para que todos os jogadores entreguem a quantidade total que eles possuem dele ao dono da carta;
- Construção de Estradas: Permite construir gratuitamente duas estradas;
- Invenção: Permite o dono da carta escolher dois recursos para serem recebidos;
- Ponto: Cartas especiais que dão 1 ponto para seu dono, e permanecem ocultas até o jogador em questão (ou qualquer outro) ganhar.

Além destas cartas, também existem duas cartas especiais dentro jogo, que não podem ser compradas, só possuem uma cópia e dão dois pontos cada:

- Maior Estrada Comercial: Dada ao primeiro jogador que conseguir criar uma sequência de 5 estradas no tabuleiro (excluindo bifurcações);
- Maior Cavalaria: Dada ao primeiro jogador que utilizar três cartas de Cavaleiro.

Diferente das outras cartas, ambas podem mudar de dono, no caso que outro jogador consiga criar uma sequência de estradas maior ou que outro jogador utilize um número maior de cartas de cavaleiro, respectivamente. Apenas nestes dois casos que um jogador pode perder pontos, já que construções e cartas de ponto não podem ser destruídas nem roubadas.

Apesar das construções e cartas precisarem de diferentes recursos para serem criadas, muitas vezes o jogador irá ter dificuldade para obter todos os recursos necessários, já que não é fácil possuir aldeias ou cidades adjacentes a terrenos de todos os tipos, e dar a sorte de saírem números favoráveis ao jogador a cada turno. Por conta disso, os jogadores podem negociar recursos entre si a vontade, além de também poderem negociar diretamente com o banco do jogo, numa razão de 4:1, onde o jogador precisa oferecer quaisquer 4 recursos ao banco, e pode pegar 1 recurso de qualquer tipo em troca. Também é possível trocar com o banco numa razão de 3:1 ou 2:1 (com um tipo de recurso específico), se o jogador possuir uma aldeia ou cidade adjacente a um porto normal (para a razão 3:1), ou a um porto especial (para a razão 2:1).

Com este cenário e regras, *Settlers of Catan* é um jogo que, apesar de possuir sorte envolvida, faz com que o jogador também precise de uma boa estratégia e de fazer negociações com outros jogadores e o banco para se tornar vitorioso. Estes elementos fazem com que, além de ser um jogo divertido e amado por muitos, seja um bom ambiente para testar diferentes estratégias de IA, com ou sem abordagens sociais. Consequentemente, Catan foi escolhido como o jogo para implementar e testar os agentes sociais pela primeira vez.

3.1 INTERAÇÕES ENTRE JOGADORES

Considerando as regras do Catan, podemos observar que existem três interações possíveis entre jogadores, que podem ser usadas como informações relevantes para construir a reputação de um jogador.

- Negociação;

- Roubar Recurso;
- Mover Ladrão.

A negociação envolve ativamente dois jogadores, onde cada um tem um papel diferente. Enquanto que o jogador do turno faz uma proposta de troca de recursos, o jogador que recebe a proposta tem o papel de aceitá-la, recusá-la ou fazer uma contraproposta. O jogador do turno não possui limite de negociações, então é possível propor negociações com todos os outros jogadores, com quaisquer combinações de recursos, até o ponto que seja possível conseguir realizar uma troca, ou até o jogador desistir. No caso de uma negociação com o banco, não existe interação com outro jogador, nem a possibilidade de recusa pelo banco. Portanto, este tipo específico de negociação não é relevante para nenhum cálculo de reputação.

Numa situação de roubo de recursos, dois a quatro jogadores estão envolvidos na interação, mas apenas o jogador que está roubando os recursos é ativo, enquanto que os jogadores que precisam dar seus recursos não possuem escolha. Na maioria dos casos, um roubo envolve apenas dois jogadores, o que rouba e o que é roubado. Para isto acontecer, é necessário que o jogador do turno tire o número 7 ao rolar os dados (ativando a movimentação do ladrão), ou utilizando a carta de desenvolvimento Cavaleiro. A situação de um roubo coletivo acontece apenas no caso do uso da carta de desenvolvimento Monopólio, que pode roubar recursos de zero a três jogadores.

A movimentação do ladrão também é um ponto de interação, onde além de existir a possibilidade de causar um roubo de recursos, também envolve outra interação, que é o posicionamento do ladrão no jogo. Se o jogador a mover o ladrão colocá-lo em um terreno que possui aldeias ou cidades adjacentes, aquelas construções não irão produzir recursos mesmo se o número do terreno em questão sair nos turnos subsequentes, até o momento que o ladrão seja movido novamente. Nesta situação, temos novamente um caso onde apenas o jogador do turno é ativo, enquanto que os jogadores prejudicados não possuem escolha. Além disso, visto que um terreno pode ter um máximo de três cidades ou aldeias adjacentes, é possível que aconteçam interações com todos os jogadores numa mesma ação.

3.1.1 Implementação

Existem algumas classes que atuam dentro do projeto para controlar o comportamento e funcionamento dos diferentes *bots* existentes. A classe *SOCRobotBrain* é responsável por controlar o raciocínio de todos os *bots*, mas dependendo do tipo de estratégia deles, a resposta para cada tipo de ação varia. Por conta disso, para controlar os pontos de interação do jogo e definir como cada um dos *bots* criados neste projeto deveriam se comportar, foi necessário modificar as funções correspondentes a cada uma das ações que possuem pontos de interação com jogadores, para que o comportamento escolhido fosse feito com base no tipo de estratégia específica daquele agente.

Código 3.1: Trecho da Função *moveRobber* (Java)

```
protected void moveRobber() {

    //...
    int strategy = getRobotParameters().getStrategyType();
    //PONTO DE DECISAO
    if (strategy == SOCRobotDM.COOP_STRATEGY ||
        strategy == SOCRobotDM.SMART_COOP_STRATEGY)
    {
        moveRobberCOOP();
        return;
    }
    if (strategy == SOCRobotDM.COMP_STRATEGY ||
        strategy == SOCRobotDM.SMART_COMP_STRATEGY)
    {
        moveRobberCOMP();
        return;
    }
    if (strategy == SOCRobotDM.SOCIAL_STRATEGY ||
        strategy == SOCRobotDM.SOCIAL2_STRATEGY ||
        strategy == SOCRobotDM.SMART_SOCIAL_STRATEGY ||
        strategy == SOCRobotDM.SMART_SOCIAL2_STRATEGY)
    {
        moveRobberSOCIAL();
        return;
    }

    D.ebugPrintln("[" + ourPlayerData.getPlayerNumber() + "] !!!
       MOVING ROBBER !!!");
    client.moveRobber(game, ourPlayerData, bestHex);
    pause(2000);
}
```

Neste trecho de código, podemos ver como funciona esta diferenciação de estratégia. Ao utilizar uma própria função da classe *SOCRobotBrain*, é possível obter um número inteiro que representa qual o tipo de estratégia daquele agente. Com base

nisso, foi possível implementar facilmente uma maneira de adicionar novos *bots*, no projeto, sem afetar o funcionamento dos *bots* originais. Considerando este código, temos que, em um caso onde o *bot* em questão possuir uma estratégia com algum tipo de viés social, ele irá terminar seu raciocínio para movimentar o ladrão dentro de uma função específica para tal, enquanto que no caso onde o mesmo seja um *bot* original do projeto, ele simplesmente vai seguir com a sua estratégia rotineira.

3.2 JSETTLERS

JSettlers[1], que significa Java Settlers, é um projeto de mesma linguagem que permite com que jogadores joguem partidas de Catan localmente ou via internet com outras pessoas ou *bots* [11]. Por possuir código aberto[2], e já ter uma estrutura que suporta IAs, inclusive possuindo 2 *bots* com estratégias diferentes [4], esse projeto foi escolhido para ser usado como ambiente para o teste de novos agentes na dissertação original deste trabalho em [2]. Consequentemente, foi decidido continuar com o mesmo ambiente.

[1] http://nand.net/jsettlers/

[2] https://github.com/jdmonin/JSettlers2/

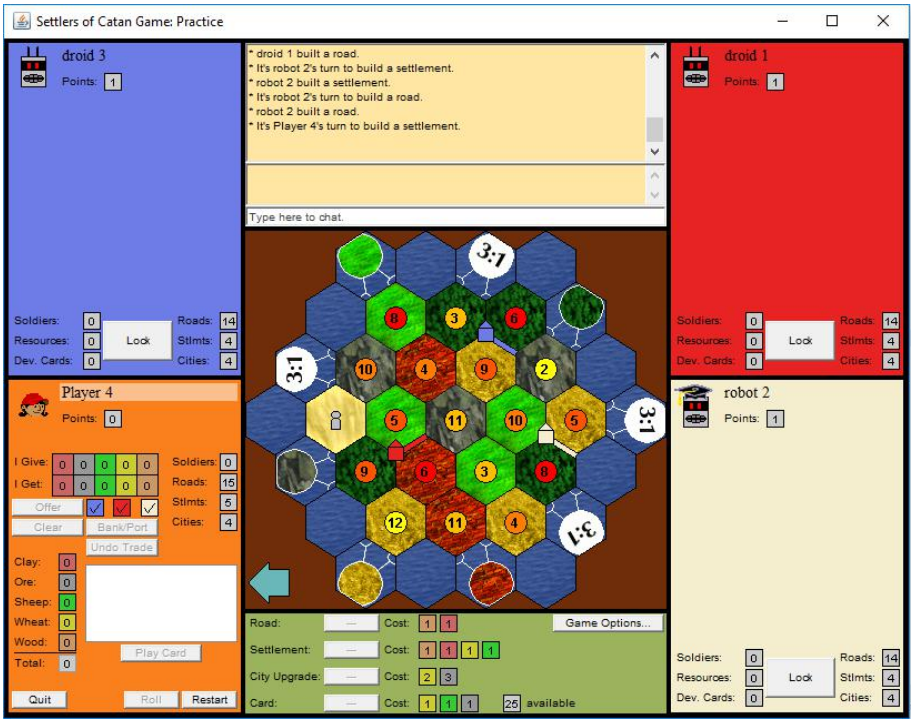


Figura 2: Interface do JSettlers.

3.2.1 Tipos de Agentes

Ao implementar o JSettlers em sua tese de doutorado, Thomas [11] criou 2 estratégias diferentes para jogar Catan, que possuem o mesmo nome dos seus *bots*: *Fast* e *Smart* [4].

A estratégia do agente *Fast* é minimizar seu ETB, que significa *Estimated Time to Build*. Logo, a cada rodada este agente calcula qual a construção que ele está mais próximo de construir, e toma todas as ações possíveis para concluir esta construção no menor tempo possível. Por conta disso, este agente não costuma comprar cartas de desenvolvimento, e não necessariamente foca em conseguir pontos, já que estradas não dão pontos (apenas no caso do jogador em questão conseguir construir a maior rota comercial da partida).

No caso do *bot Smart*, temos um agente que tem como objetivo minimizar seu ETW (*Estimated Time to Win*). Ou seja, ao invés de ter uma estratégia focada em construir o mais rápido possível, este *bot* prioriza a pontuação, seja qual for

a maneira de obtê-la (construção, carta de desenvolvimento, maior exército, etc). Apesar de possuir uma estratégia mais vitoriosa que o agente *Fast*, o processamento necessário para atingir este desempenho é aproximadamente 30 vezes maior do que o tempo que o agente *Fast* leva para tomar ações [4].

Apesar das diferenças, ambos os agentes também possuem várias familiaridades nos seus comportamentos. Ambos evitam negociar com adversários que estejam mais próximos de ganhar, e, ao mesmo tempo, tentam prejudicá-los de todas as maneiras possíveis, para que seja possível ultrapassá-los ou se manter na frente dos mesmos.

		Proporção de Construções			
Agente	**Vitórias**	**Estrada**	**Aldeia**	**Cidade**	**Carta de Desenvolvimento**
Smart	**33.8%**	33%	14%	14%	39%
Fast	22.1%	46%	25%	28%	1%

Tabela 3: Simulação com 1 agente *Smart* e 3 agentes *Fast*. Adaptado de [4].

A tabela 3 exibe os resultados obtidos em uma simulação feita por Guhe e Lascarides [4], na qual 10000 partidas entre um agente *Smart* e três agentes *Fast* foram analisadas. Como podemos ver, o agente *Smart* conseguiu ser mais vitorioso, e a maior diferença entre os dois foi o número de cartas de desenvolvimento compradas. Um dado interessante desta simulação é que, enquanto que o agente *Fast* obteve em média 66.5 recursos de terrenos, o agente *Smart* obteve em média 53.5 recursos, o que sugere que o agente *Smart* possui uma estratégia que utiliza recursos de maneira mais eficiente que a estratégia *Fast* [4].

Neste projeto e no seu antecessor [2], nenhuma estratégia de jogo foi criada, então todos os *bots* deste projeto utilizam uma mistura de uma destas duas estratégias com um dos vieses sociais criados por do Couto [2]. Portanto, as únicas mudanças feitas nos agentes criados por este projeto aconteceram nas estratégias sociais dos agentes originais, com o objetivo de melhorar seus desempenhos.

4 AGENTES

Em sua dissertação de mestrado, do Couto [2] implementou três agentes que possuíam contextos sociais diferentes, representando estereótipos humanos simplificados: agente Cooperativo, Antissocial/Competitivo e *Social*. Para fins de abreviação, os dois primeiros agentes serão chamados ao longo deste documento de *Coop* e *Comp*, respectivamente.

Apesar de todos estes três *bots* terem seu comportamento baseado em um viés social, os agentes *Coop* e *Comp* não se utilizam de nenhum cálculo de reputação e confiança, fazendo com que seus comportamentos não sejam influenciados pela maneira como seus adversários se relacionam com eles. Apenas o agente *Social* verifica a reputação de outros jogadores para definir se pode confiar (se tornando um aliado) ou não (se tornando um adversário) em outro jogador. Considerando a estratégia de vitória dos três *bots*, todos utilizavam o comportamento do *bot Fast*.

O foco principal deste trabalho foi modificar estes agentes, de maneira a melhorar seus desempenhos e criar uma experiência mais realística ao usuário, focando principalmente no *bot Social*. Abaixo podemos ver o funcionamento dos agentes originais, e todas as modificações feitas para cada uma das versões novas de cada agente.

4.1 AGENTE COOPERATIVO

O agente *Coop* tenta sempre ajudar o maior número de jogadores, e prejudicar o menor número possível. Este *bot* sempre propõe negociações a todos os jogadores, e também aceita qualquer proposta, independente de quem seja, e se é vantajosa ou não. Ao mover o ladrão, o *bot* sempre escolhe o terreno que possui o menor número de construções, minimizando a quantidade de recursos coletivos perdidos. No caso de roubar recursos, já que não é possível não roubar nenhum, o *bot* prejudica alguém aleatoriamente, já que em uma quantidade grande de interações, todos os outros jogadores seriam prejudicados de maneira parecida, o que poderia ser visto

como algo justo.

Ao analisar o código do *bot*, foi possível mapear seus comportamentos para cada tipo de interação com jogadores:

- Propor Negociação: Função *SOCRobotNegotiator.makeOfferAux*
  - Funcionamento idêntico ao do bot padrão do JSettlers. Propõe negociações a todos os jogadores, exceto os que estão próximos de ganhar.
- Aceitar/Recusar proposta: Função *SOCRobotNegotiator.considerOffer2*
  - Aceita qualquer proposta de negociação recebida, independente de quão bom ou ruim seja para o *bot*, e ignorando o quão perto ou distante o outro jogador esteja de ganhar.
- Roubar recurso: Função *SOCRobotBrain.chooseRobberVictim*
  - Rouba do primeiro jogador que faz parte da lista de escolhas possíveis. Por conta da estratégia deste agente ao mover o ladrão, é extremamente improvável que seja necessário para o *bot* escolher algum jogador.
- Mover ladrão: Função *SOCRobotBrain.moveRobberCOOP*
  - Mapeia o tabuleiro, e escolhe o terreno que tenha o menor número de cidades e aldeias possível. É extremamente improvável que esta função retorne um terreno que tenha mais de um jogador, e na maioria dos casos existirá um terreno na partida que não possui nenhuma construção.

4.1.1 Propostas

4.1.1.1 Smart Coop

Esta versão do *bot Coop* tem o mesmo comportamento social do que seu antecessor, mas se utiliza da estratégia *Smart* para planejar sua vitória. Por conta disso, ao invés de possuir uma estratégia baseada no menor ETB possível, seu objetivo é sempre minimizar seu ETW.

4.2 AGENTE ANTISSOCIAL/COMPETITIVO

O *bot* antissocial age de maneira caótica, tentando prejudicar o maior número possível de jogadores, ao mesmo tempo que tenta se beneficiar sempre que possível. Este agente só propõe negociações quando precisa de um recurso, visando diminuir ao máximo o número de interações com outros jogadores. Se algum jogador fizer qualquer proposta a ele, a mesma será recusada, independente de ser vantajosa ou não. Ao mover o ladrão, o *bot* escolhe o terreno com o maior número de construções possível, tentando causar o máximo de impacto na partida. No caso de roubar recursos, o agente *Comp* sempre escolhe o jogador que possui a maior pontuação no momento do roubo.

Ao analisar o código do *bot*, foi possível mapear seus comportamentos para cada tipo de interação com jogadores:

- Propor Negociação: Função *SOCRobotNegotiator.makeOfferAux*
  - Com o mesmo comportamento que o agente *Fast*, propõe negociações a todos os jogadores que não estejam próximos de ganhar. Sua única diferença é que não verifica se as ofertas são plausíveis de serem aceitas, sempre propondo independente do resultado.
- Aceitar/Recusar proposta: Função *SOCRobotNegotiator.considerOffer2*
  - Recusa qualquer proposta de negociação recebida, independente de quão bom ou ruim seja para o *bot*, e ignorando o quão perto ou distante o outro jogador esteja de ganhar.
- Roubar recurso: Função *SOCRobotBrain.chooseRobberVictim*
  - Mantém a estratégia do *bot Fast*, escolhendo o jogador que estiver mais próximo de ganhar entre as escolhas possíveis de roubo.
- Mover ladrão: Função *SOCRobotBrain.moveRobberCOMP*

– Mapeia o tabuleiro, e escolhe o terreno que tenha o maior número de cidades e aldeias possível. Desta maneira, o *bot Comp* consegue diminuir ao máximo o número de recursos possíveis a se ganhar por rodada.

4.2.1 Propostas

4.2.1.1 Smart Comp

Esta versão do *bot Comp* tem o mesmo comportamento social do que seu antecessor, mas se utiliza da estratégia *Smart* para planejar sua vitória. Por conta disso, ao invés de possuir uma estratégia baseada no menor ETB possível, seu objetivo é sempre minimizar seu ETW.

4.3 AGENTE SOCIAL

O agente *Social* é o principal *bot* do trabalho de do Couto [2], sendo o único que utiliza sistemas de reputação e confiança para afetar seu comportamento. Para isso, ele categoriza todas as interações entre ele e outro jogador como positivas ou negativas, fazendo com que a confiança que o agente tenha em cada jogador seja positiva, neutra ou negativa. Quando o agente *Social* interage com jogadores que possuam confiança positiva ou neutra, ele os trata de maneira similar com que um jogador humano trataria um aliado, e no caso de uma interação com um jogador que possua confiança negativa, o tratamento seria similar ao dado a um adversário.

Ao receber uma proposta de qualquer outro jogador, o *bot Social* sempre considera a proposta feita como uma interação positiva do jogador em questão, sem verificar se a proposta é vantajosa ou não para ele, e independente do agente aceitar ou não a proposta. No momento que o agente faz propostas para outros jogadores, ele verifica a confiança de cada jogador, e ele apenas faz propostas para jogadores que possuem confiança positiva e não estão próximos de vencer, fazendo com que seu viés social não fique acima da sua estratégia de vitória. Se um jogador aceitar a proposta feita, essa interação será vista como positiva, e no caso de onde a proposta

for recusada, a reputação do jogador será afetada de maneira negativa.

Quando um jogador move o ladrão, temos duas possibilidades. Na situação onde o jogador da rodada move o ladrão para um terreno que não possua nenhuma aldeia ou cidade do agente *Social*, a ação é vista como positiva mesmo não tendo uma interação direta com o agente *Social*, já que o jogador da rodada poderia tê-lo prejudicado. Logo, a única interação direta neste caso é negativa, quando o jogador da rodada move o ladrão para um terreno com aldeias e/ou cidades do *bot Social*. Já que esta interação pode ter um impacto variável para o agente, dependendo da quantidade de construções que ele tenha no terreno que irá receber o ladrão, esta interação possui peso n, onde n é igual ao número de construções que o *bot Social* possui no terreno em questão. Logo, se o jogador da rodada mover o ladrão para um terreno onde existe uma cidade do agente *Social*, essa ação terá o peso de uma interação negativa, enquanto que num cenário onde o terreno possuísse duas aldeias e uma cidade, a ação teria o peso de três interações negativas. Quando o agente *Social* move o ladrão, ele escolhe o terreno em que ele consiga prejudicar o maior número de jogadores com confiança negativa, ao mesmo tempo que não prejudique nenhum jogador que possua reputação positiva.

Se o jogador da rodada roubar recursos de um jogador que não seja o agente *Social*, não temos uma interação, enquanto que se o jogador roubar um recurso do agente, essa interação diminuirá a reputação do jogador da rodada. Uma diferença neste caso ao comparar com a movimentação do ladrão é que, por conta do roubo de recursos sempre ser limitado a apenas um recurso de outro jogador, essa interação sempre tem o peso fixo de 1. Quando o *bot Social* precisa roubar um recurso, ele sempre faz isto de maneira a prejudicar o jogador mais próximo de ganhar que possua confiança negativa, exceto no caso onde ele não tenha recursos disponíveis.

Ao analisar o código do *bot*, foi possível mapear seus comportamentos para cada tipo de interação com jogadores:

- Propor Negociação: Função *SOCRobotNegotiator.makeOfferAux*
  - Com o mesmo comportamento que o agente *Fast*, propõe negociações a

todos os jogadores que não estejam próximos de ganhar. Sua única diferença é que não propõe negociações para jogadores que possuam confiança negativa.

- Aceitar/Recusar proposta: Função *SOCRobotNegotiator.considerOffer2*
  - Aceita qualquer oferta que seja feita por um jogador com confiança positiva, e recusa qualquer oferta que seja feita por um jogador com confiança negativa, independente das ofertas serem vantajosas ou não.
- Roubar recurso: Função *SOCRobotBrain.chooseRobberVictim*
  - Rouba recurso do jogador que estiver mais próximo de ganhar e que não tenha confiança positiva. Se todos os jogadores possuírem confiança positiva, rouba do primeiro jogador da lista, independente dele ser o mais próximo a ganhar ou não.
- Mover ladrão: Função *SOCRobotBrain.moveRobberSOCIAL*
  - Move o ladrão para o terreno onde for possível causar o maior prejuízo possível, com base em algumas regras. Para calcular o prejuízo em cada terreno, é somado a reputação de todos os jogadores que tem aldeias/cidades naquele local. No caso de um terreno que possua aldeias/cidades de um jogador que tem confiança positiva, o prejuízo do terreno é visto como 0, independente dos jogadores com confiança negativa que estiverem nele.

Além disso, também foi possível mapear todos os seus cálculos de reputação:

- Receber Proposta: Função *SOCPlayer.UpdateReputationOfferMade*
  - 1 interação positiva para o jogador que fez a proposta.
- Aceitarem Proposta: Função *SOCPlayer.UpdateReputationOfferAccepted*
  - 1 interação positiva para o jogador que aceitou a proposta.
- Recusarem Proposta: Função *SOCPlayer.UpdateReputationOfferRejected*

  - 1 interação negativa para o jogador que rejeitou a proposta, apenas no caso em que ele realmente tinha recursos suficientes para aceitá-la.

- Recurso Roubado: Função *SOCPlayer.UpdateReputationResourceStealed*
  - 1 interação negativa para o jogador que roubou recursos do agente.

- Ladrão Movido: Função *SOCPlayer.UpdateReputationRobberMoved*
  - 1 interação positiva para o jogador que moveu o ladrão, se nenhuma aldeia/cidade do agente for prejudicada. n interações negativas para o jogador que moveu o ladrão, onde n é igual ao número de recursos que o agente está deixando de ganhar por conta do posicionamento do ladrão.

4.3.1 Propostas

Por ser o foco do trabalho e o único agente que utilizava sistemas de recomendação e confiança, o *bot Social* é o que mais possui variações, sofrendo algumas mudanças no seu cálculo de reputação e na sua estratégia de jogo.

4.3.1.1 Smart Social

Esta versão do *bot Social* tem o mesmo comportamento social do que seu antecessor, mas se utiliza da estratégia *Smart* para planejar sua vitória. Por conta disso, ao invés de possuir uma estratégia baseada no menor ETB possível, seu objetivo é sempre minimizar seu ETW.

4.3.1.2 Social 2

O agente *Social 2* possui várias mudanças no seu cálculo de reputação, criadas após uma análise detalhada de todos os cálculos do *bot* original, que permitiu identificar possíveis melhorias no seu funcionamento.

O primeiro passo em identificar mudanças que pudessem ser feitas no cálculo de reputação do agente foi analisar o comportamento da versão original, nos casos onde

o *bot* modificava a reputação dos outros jogadores que estivessem interagindo com ele, de maneira positiva ou negativa. Além disso, também foi importante analisar os efeitos da reputação nas ações do agente, visto que existem comportamentos diferentes dependendo se o jogador alvo é visto como um aliado ou adversário.

No caso onde o agente recebia uma proposta, faria mais sentido deixar de aumentar a reputação do jogador para qualquer proposta feita, e passar a verificar se a proposta possui uma razão entre recursos dados e recursos recebidos de 1:1, 2:1 ou 3:1 em diante. Com base nisso, o agente poderia decidir se é uma proposta aceitável para ser considerada positiva, se é uma proposta mediana, e deve ser vista como neutra, ou se é uma proposta ruim, que será vista como negativa.

Além desta interação, outro momento analisado que se mostrou passível de sofrer modificações foi a situação onde o *bot Social* decidia se iria aceitar ou não uma proposta recebida. Na versão original, o agente aceitava qualquer proposta feita por um jogador que tivesse reputação positiva. Este comportamento permite com que o agente seja prejudicado por um jogador manipulador em relação a razão da quantidade de recursos recebidos e dados, além de que não necessariamente o *bot* estaria interessado nos recursos que irá receber em troca, prejudicando sua estratégia de vitória, que provavelmente não iria aceitar a proposta recebida se não fosse por conta de seu viés social. Portanto, seria melhor deixar de aceitar qualquer oferta feita por um jogador com confiança positiva, verificando se ela realmente vale a pena. No caso de recusar ofertas quando o jogador em questão tem confiança negativa, nenhuma modificação se mostrou necessária.

Também foram feitas análises para todas as outras interações possíveis entre o agente *Social* e outros jogadores, mas apenas os dois cenários explicados acima foram considerados fracos e escolhidos para serem modificados. Portanto, o passo seguinte foi implementar estas duas mudanças, para que elas pudessem ser testadas em prática com todos os outros *bots*.

O cálculo de reputação ao receber propostas de negociação foi mudado para verificar a razão entre o número de recursos a serem dados e o número de recursos a serem recebidos. A razão é representada por:

$$\text{Razão} = \frac{\text{Quantidade de recursos dados}}{\text{Quantidade de recursos recebidos}} \tag{4.1}$$

Sendo usada para definir três possibilidades de negociação:

- Razão $<$ 1.5: 1 reputação positiva ao jogador que propôs a negociação
- 1.5 $<=$ Razão $<=$ 2.5: reputação permanece a mesma
- Razão $>$ 2.5: 1 reputação negativa ao jogador que propôs a negociação

Após ser implementado, o código foi depurado em situações reais de simulação, para verificar quais razões eram mais comuns entre as outras. Foi notado que, por conta da estratégia dos *bots*, que costumam fazer ofertas com razão 1:1 ou 2:1 (em caso de recursos mais críticos), essa implementação acabou não afetando o cálculo de reputação deste agente em comparação com o agente original no cenário de partida apenas com *bots*. Estas duas razões acabam dando, respectivamente, razões de valor 1 e 0.5, ambas caindo no caso de incrementar a reputação do jogador que propôs a oferta. Portanto, esta modificação no código só tem impacto real em casos de *bots* futuros que venham a ter uma estratégia de negociação diferente, ou em casos de jogadores que, vendo uma oportunidade vantajosa para explorar um agente, decidam propor uma negociação com razão muito alta.

A modificação no cenário de aceitar negociações também foi implementada, de maneira que o *bot* utilize seu viés racional para definir se a negociação é vantajosa ou não para ele, e quando for, ele aceite apenas no caso que o jogador que fez a proposta possuir confiança positiva. Em qualquer caso que a confiança for negativa, a proposta será recusada, da mesma maneira que o agente *Social* funciona.

4.3.1.3 Smart Social 2

Este agente nada mais é que um misto das versões *Smart Social* e *Social 2*. Seu comportamento social é idêntico ao *bot Social 2*, mas se utiliza da estratégia *Smart* para planejar sua vitória. Por conta disso, ao invés de possuir uma estratégia baseada no menor ETB possível, seu objetivo é sempre minimizar seu ETW.

4.4 VISÃO GERAL

Nesta seção, temos um resumo dos comportamentos de cada agente, por meio de tabelas que explicam cada um dos principais tipos, junto com suas variações.

	Comportamento	
Ação	**Cooperativo**	**Antissocial**
Propor Negociação	Propõe a todos, exceto os que estão próximos de ganhar	
Aceitar Proposta	Aceita **qualquer** proposta	-
Recusar Proposta	-	Recusa **qualquer** proposta
Roubar Recurso	Rouba do primeiro jogador possível, se necessário. Na maioria dos casos, não rouba de ninguém	Rouba do jogador mais próximo de ganhar
Mover Ladrão	Escolhe o terreno com o **menor** número de aldeias e cidades possível	Escolhe o terreno com o **maior** número de aldeias e cidades possível

Tabela 4: Comportamento dos agentes Cooperativo, Antissocial e suas variações.

	Comportamento	
Ação	**Social, Smart Social**	**Social 2, Smart Social 2**
Propor Negociação	Propõe a jogadores que não estejam próximos de ganhar e que não possuam confiança negativa	
Aceitar Proposta	Aceita **qualquer** proposta feita por jogadores que não possuam confiança negativa	Aceita propostas vantajosas feitas por jogadores que não possuam confiança negativa
Recusar Proposta	Recusa **qualquer** proposta feita por jogadores com confiança negativa	
Roubar Recurso	Rouba do jogador mais próximo de ganhar que possua confiança negativa	
Mover Ladrão	Escolhe o terreno com o **maior** número de aldeias e cidades de jogadores com reputação negativa possível, desde que não possua nenhuma aldeia ou cidade de um jogador com reputação positiva	

Tabela 5: Comportamento dos agentes *Social*, *Social 2* e suas variações.

Exclusivamente no caso dos agentes do tipo *Social*, também temos as definições de cálculos de reputação.

	Cálculo de Reputação	
Interação	**Social, Smart Social**	**Social 2, Smart Social 2**
Receber Proposta	+1 Interação Positiva	Interação varia entre positiva, negativa e neutra dependendo da razão da negociação
Aceitarem Proposta	+1 Interação Positiva	
Recusarem Proposta	+1 Interação Negativa, se o jogador tivesse recursos para aceitar a proposta	
Recurso Roubado	+1 Interação Negativa	
Ladrão Movido	+1 Interação Positiva se nenhuma aldeia ou cidade do agente foi prejudicada, +1 Interação Negativa para cada aldeia ou cidade prejudicada do agente	

Tabela 6: Cálculos de reputação dos agentes *Social*, *Social 2* e suas variações.

5 EXPERIMENTOS

Neste capítulo, todos os experimentos feitos neste trabalho são explicados. Já que o objetivo deste projeto é propor uma nova abordagem para IAs que permita tornar a experiência de jogadores algo mais realístico do que a abordagem tradicional, não é possível apenas implementar agentes e considerar o objetivo como conquistado, também é necessário achar maneiras de mensurar o desempenho destes *bots* e analisar o quão perto ou longe estamos de chegar no resultado proposto. Para isso, diferentes simulações foram feitas, analisando desde o desempenho dos novos agentes num cenário exclusivo com *bots*, até a emoção e experiência de jogadores humanos ao enfrentá-los.

5.1 SIMULAÇÕES ENTRE AGENTES

A maior parte das simulações foram feitas exclusivamente entre agentes, tendo como objetivo testar o desempenho de cada uma das variações de agentes criadas em relação aos seus antecessores. Já que estas simulações não tinham a participação de jogadores humanos, cada uma delas possuem um número muito maior de partidas do que as simulações com jogadores, foram mais fáceis de serem produzidas pois não dependiam de ninguém, e puderam ser automatizadas.

Cada uma das simulações (tirando o primeiro experimento feito) é composta por 1000 partidas. Originalmente, cada uma delas era executada no JSettlers com interface gráfica e com uma quantidade de *delay* considerável nos *bots*, tentando reproduzir o tempo de raciocínio de um jogador humano. Essas características fizeram com que, originalmente, o tempo necessário para cada partida acontecer fosse de cerca de 7 minutos. Já que isso faria com que fosse impossível executar todas as simulações existentes neste projeto num tempo aceitável, foi necessário modificar o código do JSettlers para permitir com que fosse possível executar uma partida sem interface gráfica, e sem o *delay* nos agentes. Remover a interface gráfica foi uma tarefa simples, já que no projeto feito por do Couto [2], já existia um parâmetro que permitia a execução de uma partida sem interface gráfica, bastando

utilizá-lo para conseguir tal efeito. Já a remoção do tempo extra de raciocínio dos agentes precisou de implementação em código. Esta implementação foi relativamente simples, já que apenas foi necessário modificar as classes que faziam o raciocínio dos *bots* para deixarem de chamar funções de *sleep* dentro das tomadas de decisão do agente. Estas duas modificações fizeram com que o tempo médio de partida caísse de 7 minutos para aproximadamente 5 segundos.

Além disso, para facilitar a automatização, o funcionamento do fluxo de execuções e análises de simulações foi modificado. Na dissertação de mestrado de do Couto [2], todo este processo de simulação acontecia dentro de outro programa, criado por ele mesmo, chamado SOCSim. Este projeto escrito em C# era o encarregado de executar cada uma das partidas do JSettlers, ler os logs de cada partida, e depois criar um arquivo XML com algumas métricas de cada partida, que posteriormente poderia ser transformado em conjunto com outros arquivos XML em um único arquivo de planilha, contendo uma análise geral das métricas da simulação. Neste projeto, o fluxo foi modificado, onde um arquivo *batch* foi o encarregado de executar todas as partidas do JSettlers e salvar seus resultados em arquivos log, ao invés de deixar este controle para o SOCSim. Ao executar todas as partidas de uma simulação, o mesmo executa o SOCSim, gerando os arquivos XML, e por fim o arquivo de planilha. Esta modificação foi importante pois, além de simplificar o processo de execução de simulações, também ajudou a deixá-lo mais modularizado, o que foi importante para facilitar a reprodução de alguns trechos de simulação, já que ambos os programas possuem alguns *bugs* que acabam criando partidas e arquivos de resultado corrompidos.

Todos os testes realizados foram feitos dentro do sistema operacional Windows 10, e para executar o programa JSettlers, foi utilizada a versão 1.8.0_161 do Java. Além disso, já que o foco foi testar o desempenho dos novos agentes ao enfrentar os antigos, e garantir que eles seriam um desafio igual ou maior do que os *bots* originais para jogadores humanos, a única métrica utilizada foi o número de vitórias.

5.1.1 Simulações

5.1.1.1 Reprodução das simulações aleatórias originais

Esta primeira simulação foi feita com o objetivo de comparar os resultados obtidos na dissertação de do Couto [2] com resultados obtidos utilizando os agentes disponibilizados neste projeto, já que as implementações feitas na dissertação de mestrado não foram armazenadas com nenhum tipo de sistema de versionamento. Por conta do objetivo de comparação, este experimento possui uma simulação com número baixo de partidas em relação ao resto do projeto, reproduzindo o número de partidas da dissertação. Além disso, apesar do risco da aleatoriedade entre partidas ser o fator de diferença entre os resultados do experimento, apenas os experimentos aleatórios de do Couto foram reproduzidos. Esta decisão foi tomada por conta da impossibilidade de saber quais foram os valores de variáveis utilizados nos experimentos determinísticos do mesmo.

Infelizmente a dissertação [2] não possui muitos dados de experimentos aleatórios, já que foca em testar casos determinísticos. Existem apenas dois dados apresentados: quantidade de vitórias de cada agente, e análise entre quantidade de propostas rejeitadas e quantidade de propostas aceitas entre vários *bots*. Por conta disso, para comparar os dois resultados, foram utilizados apenas o número de vitórias.

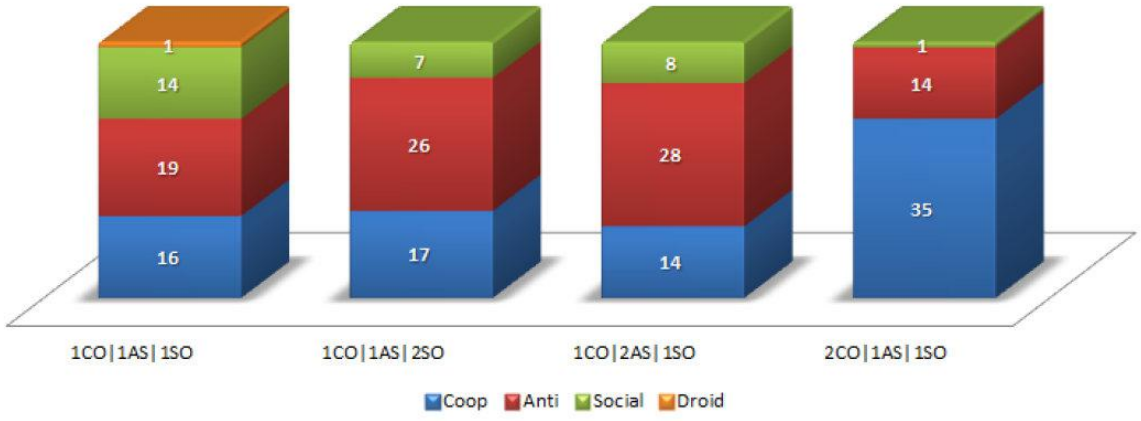


Figura 3: Simulação de 200 partidas sobre o número de vitórias por agente. Adaptado de [2].

Na simulação de do Couto [2], temos 4 cenários diferentes, sempre variando entre 4 *bots*: Cooperativo, Antissocial, Social e Droid, onde temos os três agentes sociais da dissertação e o *bot Fast* padrão do JSettlers, respectivamente. A legenda abaixo de cada cenário na figura 3 representa o número de agentes sociais participantes de tal. As siglas "CO", "AS"e "SO" representam, respectivamente, Cooperativo, Antissocial e Social.

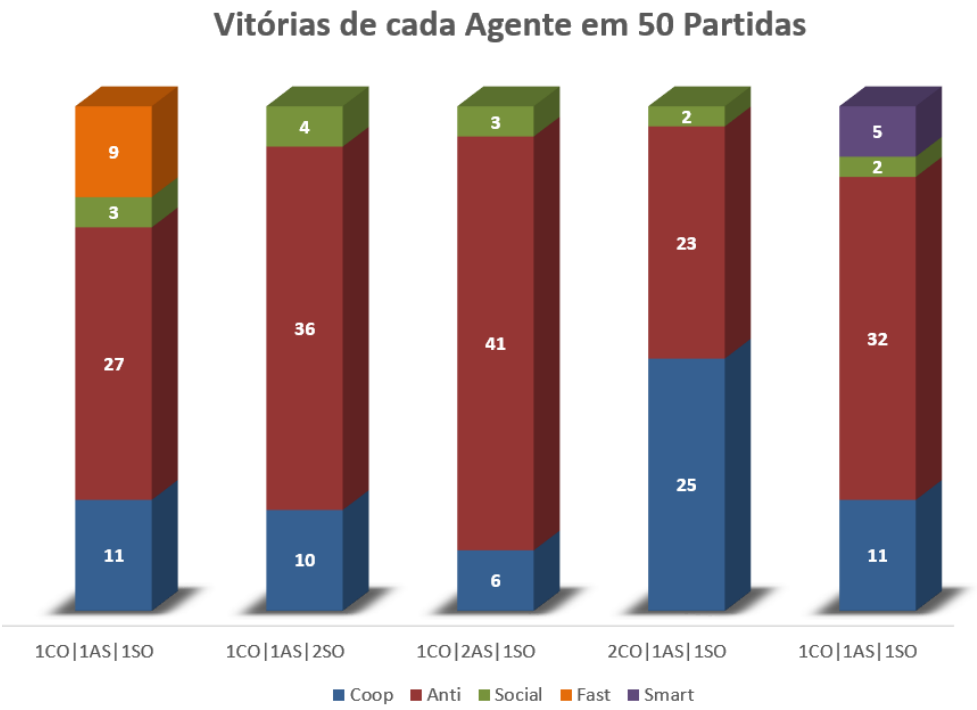


Figura 4: Simulação com 250 partidas.

Na simulação feita neste projeto, o mesmo cenário da figura 3 foi realizado, junto com um caso extra: uma simulação de 50 partidas com os agentes originais e o *bot Smart* padrão do JSettlers. Como podemos perceber comparando as figuras 3 e 4, apesar das quantidades de vitórias serem diferentes em cada simulação, o comportamento dos agentes em cada simulação é levemente parecido, com o agente antissocial se destacando dos demais, e o cooperativo terminando em segundo lugar, exceto no caso onde existe mais de um cooperativo jogando, quando consegue ser mais vitorioso que o antissocial. Para verificar mais a fundo se os resultados realmente são parecidos, foram testados casos maiores, dessa vez com 200 partidas por caso, e não 50 como anteriormente.

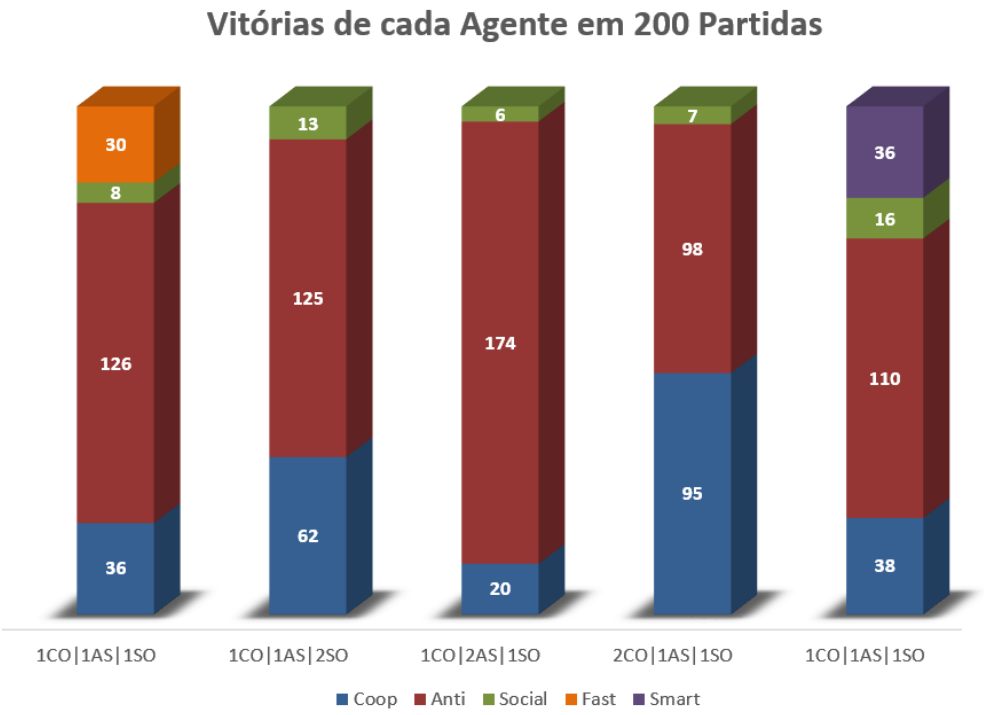


Figura 5: Simulação com 1000 partidas.

Nesta simulação, a hegemonia do agente antissocial foi bem similar aos resultados da figura 4, só que desta vez conseguindo ser mais vitorioso que dois *bots* cooperativos juntos. Também vemos que apesar de termos testado os *bots* num ambiente com 4 vezes mais partidas, a porcentagem de vitórias foi quase idêntica, com uma variação de apenas 0.3%. Enquanto que na simulação de 50 vitórias o agente antissocial obteve vitória em 63.6% das partidas, nesta a porcentagem foi de 63.3%. Já nos dados de do Couto (representados na figura 3), o *bot* antissocial tinha conseguido 43.5% de vitórias.

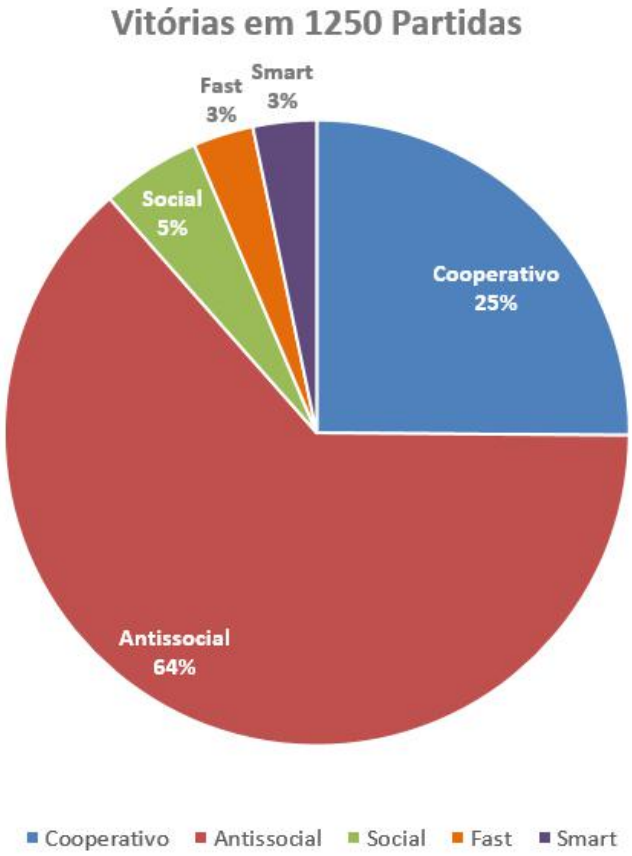


Figura 6: Junção dos dados das simulações de 250 e 1000 partidas.

Comparando a porcentagem de vitórias de cada agente, podemos perceber mais claramente a diferença entre os dados da figura 3, e os que foram produzidos nesta simulação. A diferença de desempenho entre os *bots* Cooperativo e Antissocial, que na figura 3 era desprezível, se tornou considerável ao calcular a porcentagem de vitórias dos agentes na figura 6. As vitórias do *bot* Antissocial subiram de 43% para 64%, enquanto que as do Cooperativo caíram de 41% para 25%. O desempenho do *bot* Social, assim como o Cooperativo, mostrou uma queda, reduzindo de 15% para 5%. Já o *bot Fast* não mostrou uma diferença notável, subindo de 1% para 3%.

Analisando estes dados, vemos que apesar das posições em cada caso serem muito similares as quais o do Couto [2] apresentou em sua dissertação, a variação de vitórias sugere uma diferença no comportamento dos *bots* entre as duas versões.

5.1.1.2 Teste de desempenho dos agentes Smart

Foram executadas 1000 partidas, com o objetivo de comparar os desempenhos de cada um dos *bots Smart* com suas versões originais, que se utilizam da estratégia *Fast*.

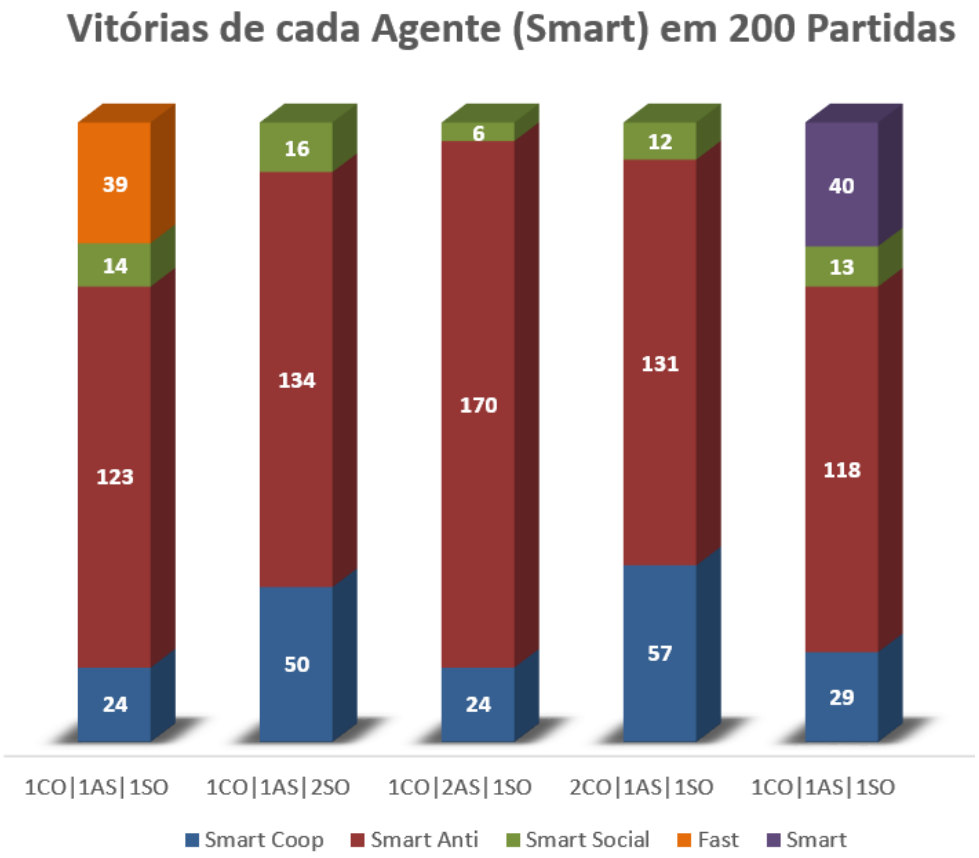


Figura 7: Simulação com 1000 partidas entre os agentes versão *Smart*.

Comparando os resultados da figura 7 com os obtidos na simulação anterior, os desempenhos foram relativamente parecidos, com exceção do caso onde tínhamos dois *bots* Cooperativos, um *bot* Antissocial e um *bot* Social jogando ao mesmo tempo. Nesta simulação, o *bot* Antissocial conseguiu se destacar e vencer os dois *bots* Cooperativos, vencendo em 65.5% das partidas, enquanto que na simulação *Fast* (representada na figura 5), este mesmo caso foi bem mais equilibrado, onde o *bot* Antissocial conseguiu apenas 49% das vitórias.

Analisando o percentual de vitórias de todos os agentes nas figuras 6 e 7, con-

seguimos ver que não aconteceram diferenças consideráveis nas vitórias dos *bots*, exceto pelo caso do agente Cooperativo, que teve 7% de vitórias a menos (caindo para 18%), principalmente pela diferença de rendimento no cenário onde tinham dois *bots* Cooperativos. O agente Antissocial conseguiu 4% a mais de vitórias (subindo para 68%), enquanto que os agentes *Social*, *Fast* e *Smart* conseguiram 1% a mais de vitórias.

Apesar da diferença de desempenho conhecida entre os *bots Fast* e *Smart* do JSettlers [4], essa mudança de estratégia na parte racional dos agentes acabou não tendo mudanças de desempenho expressivas.

5.1.1.3 Teste de desempenho do novo agente Social

Foram executadas 2000 partidas, divididas em 1000 partidas com *bots Social 2* e *bots* versão *Fast*, e outras 1000 partidas com agentes *Smart Social 2* e agentes versão *Smart*. O objetivo deste experimento foi testar o desempenho dos novos agentes em situações similares as que foram utilizadas anteriormente para testar os *bots Social* e *Smart Social*.

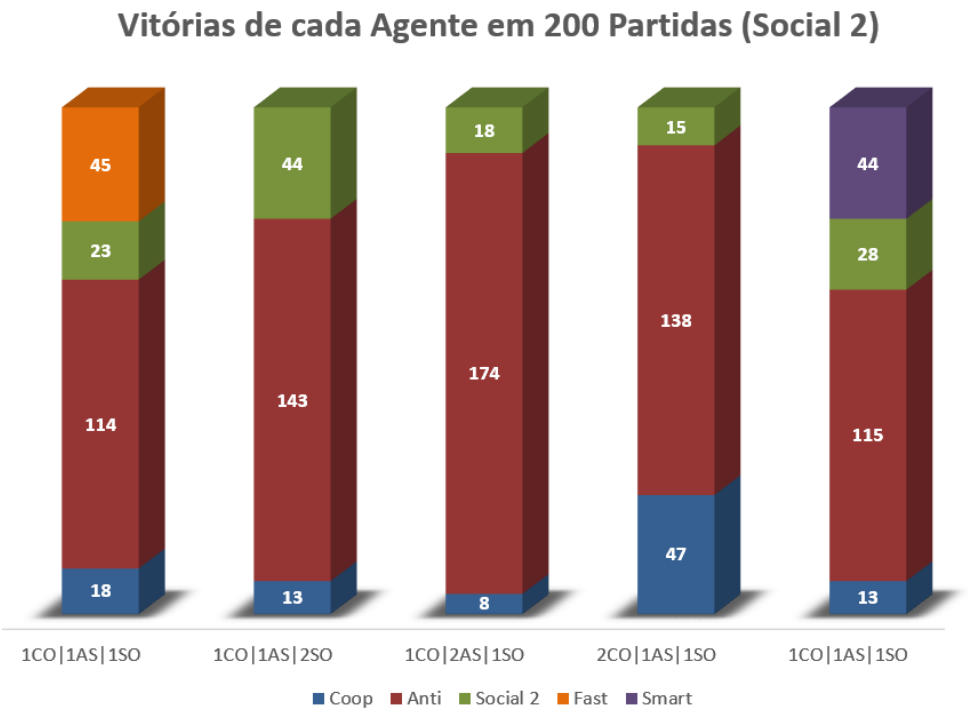


Figura 8: Simulação com 1000 partidas entre o agente *Social 2* e agentes versão *Fast*.

Na primeira simulação deste experimento, ao analisar o gráfico de vitórias por configuração de jogadores, é possível ver como a quantidade de vitórias do agente *Social 2* foi mais expressiva do que com sua estratégia original, ilustrada na figura 6. Também podemos notar que, diferentemente da figura 6, o *bot* conseguiu ter uma quantidade de vitórias maior que o *bot* Cooperativo em todos os casos, exceto quando existem 2 agentes Cooperativos na mesma partida. Ao pegar os mesmos dados e olhar em conjunto para analisar a porcentagem de vitórias geral, conseguimos notar de maneira mais clara como o resultado do agente *Social 2* se saiu melhor que seu antecessor. O *bot* conseguiu vencer em 13% das partidas, enquanto que sua versão anterior conseguiu apenas 5% de vitórias.

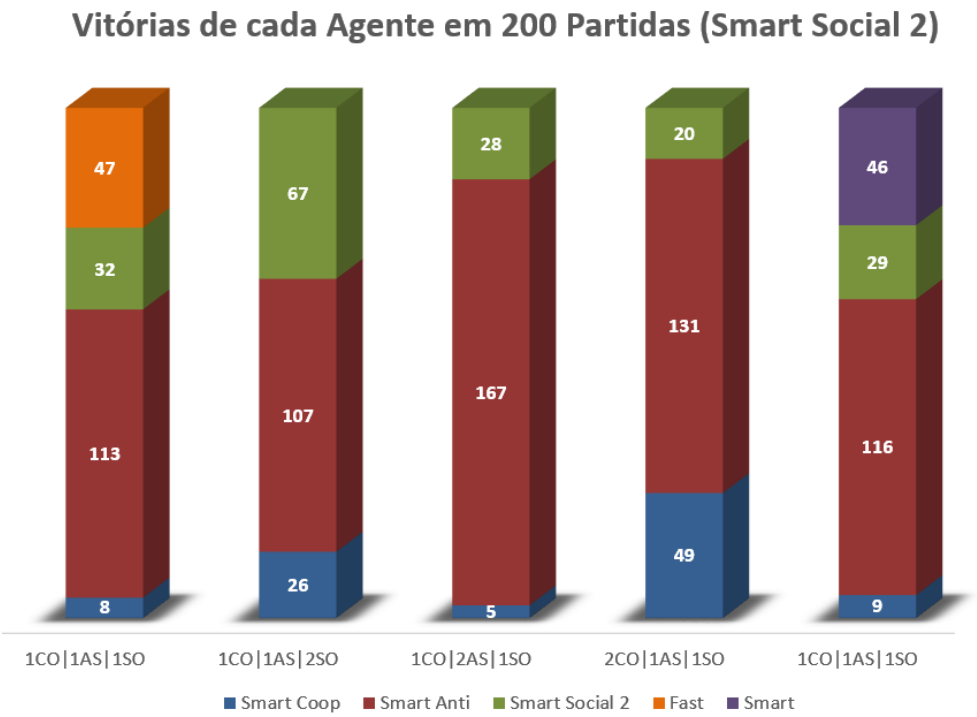


Figura 9: Simulação com 1000 partidas entre o agente *Smart Social 2* e agentes versão *Smart*.

Na segunda simulação, também é possível notar uma melhora do *bot Smart Social 2* em comparação com sua versão original (analisada na figura 7), que chega a ser maior do que a melhora observada entre os resultados das figuras 6 e 8. Com o gráfico de vitórias por configuração de jogadores, vemos que o agente conseguiu ganhar mais vezes em todas as configurações, e que sua estratégia fez com que o agente Cooperativo tivesse mais dificuldade de ganhar. Essa situação é parecida com os resultados obtidos com o agente Social original e o agente Cooperativo na figura 6, mas de maneira invertida. O *bot* Cooperativo quase não conseguiu ganhar na maioria das configurações, só obtendo resultados melhores quando existiam dois agentes Sociais ou Cooperativos juntos.

Corroborando o que foi visto no gráfico de vitórias por configuração de jogadores, a análise da porcentagem de vitórias geral nos permite perceber que o *bot Smart Social 2* conseguiu vencer 18% das partidas, e que o *bot Smart Coop* só venceu 10%. No experimento anterior com agentes *Smart* (figura 7) o agente *Smart Coop* tinha

conseguido obter 18% de vitórias, e o agente *Smart Social* teve um desempenho três vezes pior, com apenas 6% de vitórias.

Em ambas as simulações o agente Antissocial continuou sendo o grande vencedor, com sua estratégia de prejudicar a todos e só negociar quando lhe convém. Sua porcentagem de vitórias teve uma variação de 4 a 5% em ambas as simulações, o que não muda sua hegemonia ao enfrentar diversos agentes ao mesmo tempo.

Por fim, com base em todos os dados apresentados, é possível concluir que a mudança na estratégia de aceitação de negociações dos *bots* Sociais foi eficaz, aumentando sua competitividade sem fazer com que seu comportamento seja menos social.

5.1.1.4 Teste de desempenho com memória

Finalizando a implementação da memória por meio de inicialização de reputação, o próximo passo foi testar seu desempenho, para descobrir como uma mudança de reputação no início de uma partida poderia influenciar a estratégia do agente *Smart Social 2* durante a partida. As simulações feitas foram:

- Reputação +5 para todos os jogadores
- Reputação -5 para todos os jogadores
- Reputação +1000 para todos os jogadores
- Reputação -1000 para todos os jogadores
- Reputação +1000 para agentes Cooperativos e -1000 para Antissociais
- Reputação igual ao da partida anterior, mantendo memória em 10 partidas

Para fins de diferenciação, chamaremos o agente *Smart Social 2* com inicialização positiva de *Smart Social 2+*, enquanto que no caso negativo, chamaremos de *Smart Social 2-*. Além disso, o agente com inicialização positiva e negativa ao mesmo tempo será chamado de *Smart Social 2+-*.

Para as duas primeiras simulações, foi escolhido usar um valor pequeno de reputação para permitir com que o agente não demorasse para mudar de opinião sobre outros agentes. Afinal, em um caso onde a reputação inicial fosse muito alta ou muito baixa, seria possível que o *bot Smart Social 2+* jogasse uma partida inteira considerando que os outros participantes da partida fossem todos positivos ou negativos, o que faria com seu comportamento se tornasse bem similar a um agente Cooperativo, ou a um agente Antissocial.

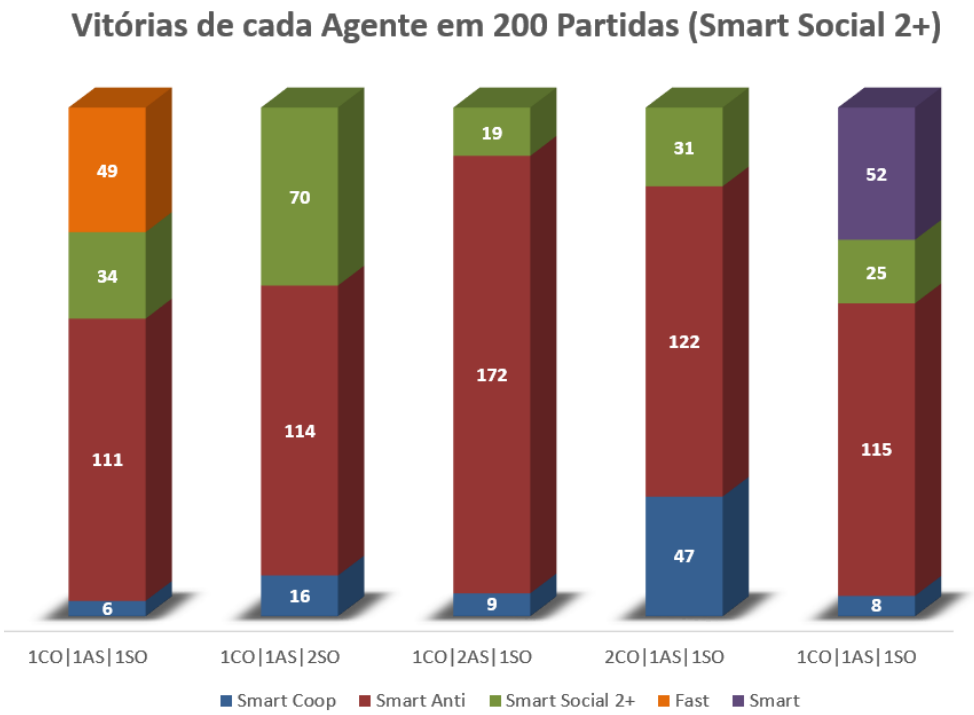


Figura 10: Simulação com 1000 partidas entre o agente *Smart Social 2+* e agentes versão *Smart*.

Nesta simulação, a quantidade de vitórias do agente *Smart Social 2+* em comparação ao agente *Smart Social 2* na figura 9 foi quase idêntica, tendo uma variação negativa de apenas 1.7%. Os únicos casos que tiveram uma mudança considerável foram o caso "1CO|2AS|1SO" e o caso "2CO|1AS|1SO". No primeiro, o desempenho do *bot Smart Social 2+* foi 47.3% melhor que o *bot Smart Social 2*, enquanto que no segundo tivemos uma situação contrária, onde o novo agente teve um desempenho pior que sua versão anterior, obtendo 35.4% vitórias a menos. Com os resultados obtidos, podemos afirmar que essa inicialização de reputação acabou não causando

mudanças consideráveis, e o seu desempenho se manteve igual no geral.

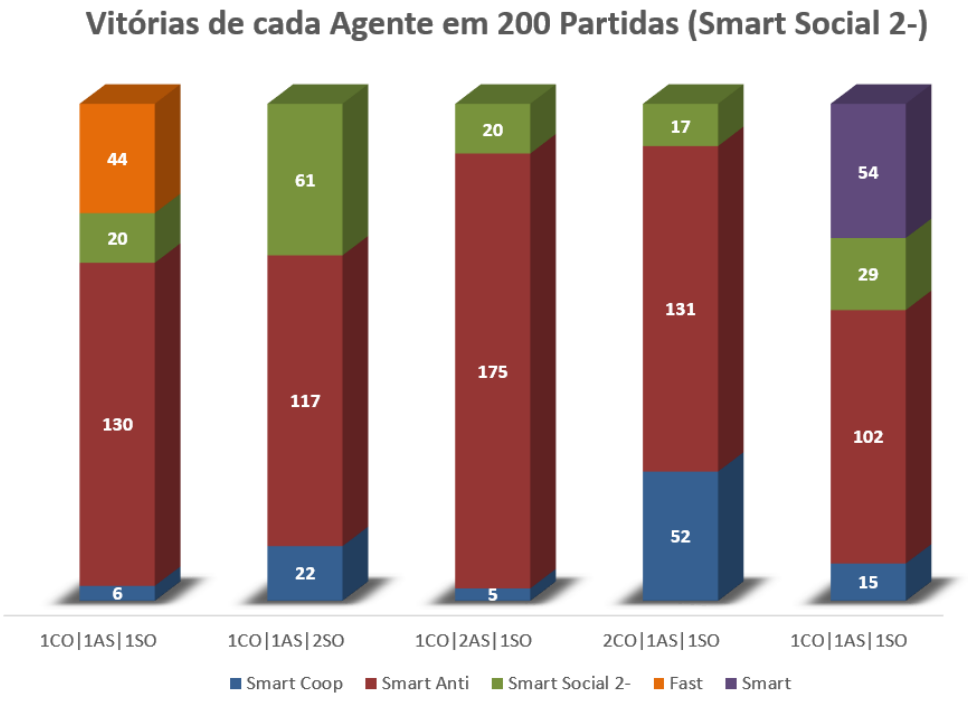


Figura 11: Simulação com 1000 partidas entre o agente *Smart Social 2-* e agentes versão *Smart*.

Diferentemente da simulação anterior, o experimento com o agente *Smart Social 2-* apresentou uma diferença considerável do agente *Smart Social 2*. De todos os casos testados, em apenas um deles ("1CO|1AS|1SO" com *bot Smart*) o desempenho foi igual ao resultado da figura 9, e em todos os outros, o resultado foi inferior. Ao analisar o número de vitórias geral, foi possível ver o tamanho da discrepância. Enquanto que o agente *Smart Social 2* tinha tido 179 vitórias na sua simulação, o agente *Smart Social2-* só conseguiu obter 147 vitórias, representando uma queda de 17.8%. Este comportamento acabou seguindo nos outros experimentos com reputação negativa, mostrando o quão ruim é para as versões do *bot Social* considerar todos os jogadores de uma partida como adversários.

Após estes dois experimentos com reputações baixas, foi o momento de testar como o *bot Smart Social 2* se saía ao ser inicializado com reputações muito altas, a ponto de não ser possível mudar a confiança referente a um jogador de positiva para

negativa e vice-versa dentro de apenas uma partida. Para isso, as reputações de todos os jogadores foram iniciadas com o valor 1000, primeiramente em um experimento exclusivamente positivo, e depois em um negativo. O principal objetivo destes dois experimentos foi ver na prática o quão próximo o comportamento dos agentes versão Social estão dos agentes Cooperativo e Antissocial em um cenário onde o *bot* categorize todos os outros jogadores como aliados (Cooperativo) ou adversários (Antissocial).

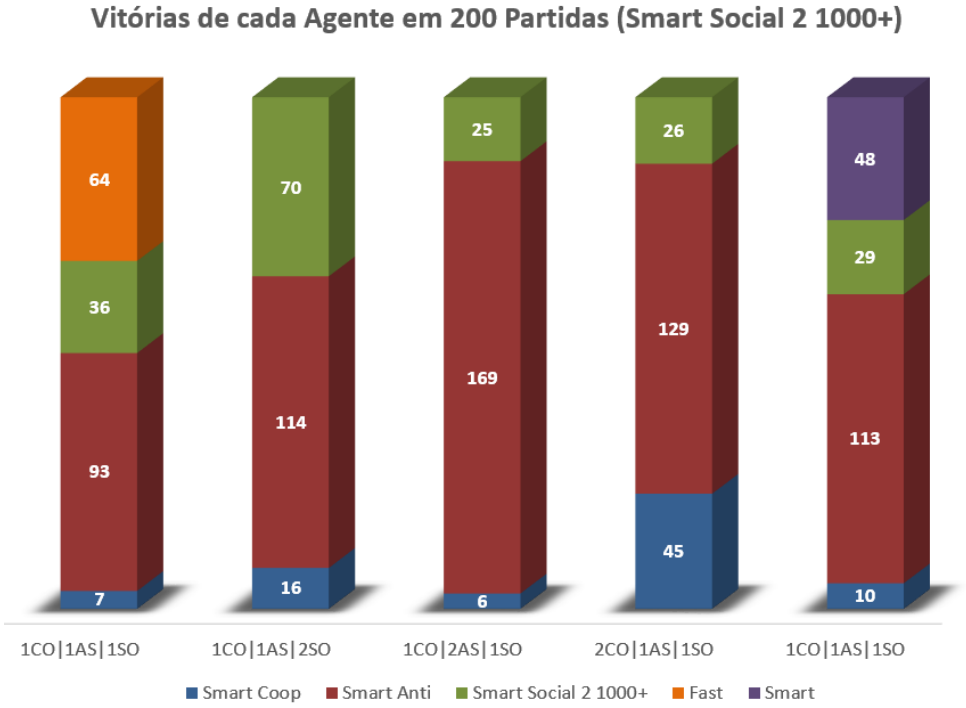


Figura 12: Simulação com 1000 partidas entre o agente *Smart Social 2 1000+* e agentes versão *Smart*.

A primeira simulação feita foi com o agente *Smart Social 2* sendo inicializado com reputação positiva igual a 1000 para todos os outros jogadores. De maneira parecida com o experimento feito com o agente *Smart Social 2+*, esta simulação teve resultados melhores que a versão do *bot* sem inicialização de reputação na figura 7, tendo um desempenho pior apenas no caso "1CO|2AS|1SO", onde tivemos uma redução de 10.7% no número de vitórias. Por fim, o resultado geral foi de 186 vitórias, representando um desempenho 3.9% melhor. Este resultado, apesar

de apresentar uma melhora, é baixo demais para representar um ganho real de desempenho, podendo ter acontecido apenas pela aleatoriedade do jogo, e não pela mudança feita.

Também é importante frisar que, apesar desta versão de agente ter tido suas reputações inicializadas de uma maneira que poderia levar a crer que seu resultado e comportamento seria parecido com o do agente Cooperativo, isto não ocorreu. A análise que podemos fazer disso é que, apesar das reputações terem feito com que o *bot Smart Social 2* tenha visto todos os outros jogadores como aliados (da mesma maneira que o *bot* Cooperativo faz), seu comportamento não é igual a ele, já que este agente não aceita qualquer proposta de negociação feita por um aliado. Esta diferença foi o que mais pesou na diferença de desempenho, e é justamente o motivo que faz com que o agente Cooperativo tenha dificuldade de conseguir muitas vitórias nos cenários estudados.

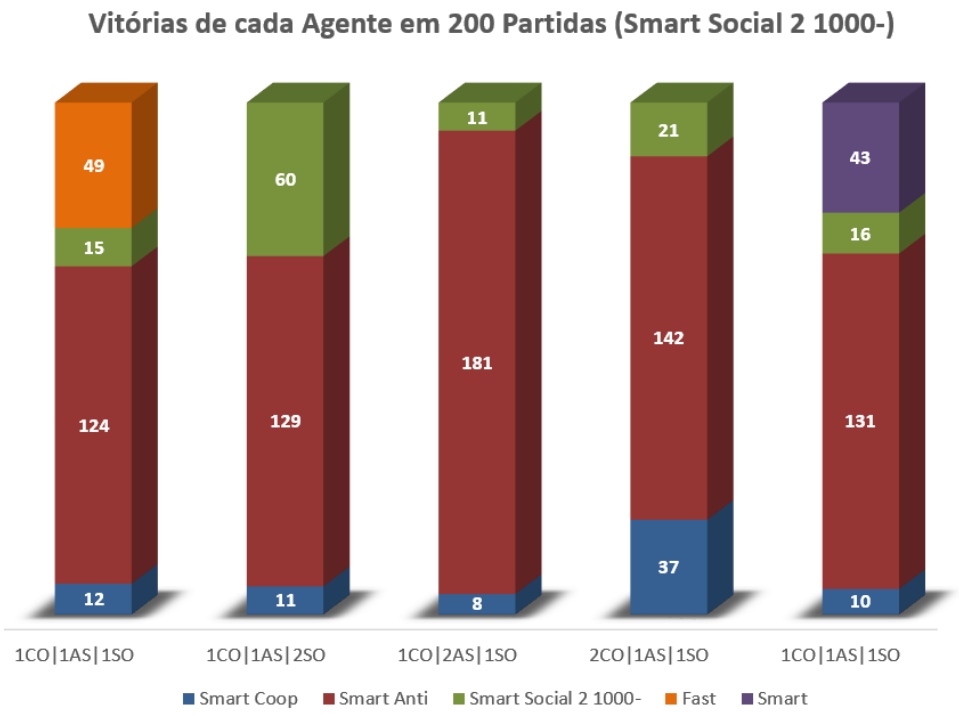


Figura 13: Simulação com 1000 partidas entre o agente *Smart Social 2 1000-* e agentes versão *Smart*.

Nesta simulação, o resultado foi parecido com o outro experimento que foi feito com reputações negativas, tendo um resultado consideravelmente mais baixo do que o desempenho do agente *Smart Social 2* na figura 7. Apenas o caso "2CO|1AS|1SO" teve um resultado acima do que o original, enquanto que todos os outros 4 casos tiveram desempenhos bem menores, com o pior caso ("1CO|2AS|1SO") tendo um desempenho 60.7% pior. Ao todo, esta versão do agente conseguiu apenas 123 vitórias, o que representa um desempenho geral 31.2% menor que a versão sem inicialização de reputação.

Além disso, ao analisar o comportamento do agente, podemos entender o que aconteceu para que o desempenho dele fosse tão diferente do desempenho que o agente Antissocial, que é o melhor *bot* de todas as simulações feitas, possui normalmente. A principal característica do agente *Smart Social 2* que fez com que seu comportamento fosse diferente do agente Antissocial é a maneira com que ele lida com negociações. Já que ele sempre recusa propostas feitas por jogadores com

confiança negativa, e também não faz propostas para esses jogadores, o *bot Smart Social 2* acabou não negociando com nenhum outro jogador neste cenário, tendo que fazer negociações exclusivamente com o banco. Esse comportamento é diferente do *bot* Antissocial, que apesar de recusar propostas feitas por qualquer jogador, não deixa de fazer propostas para todos, garantido recursos sem ter que depender do banco. Isso também mostra que, ao depender exclusivamente do banco e não fazer negociações com nenhum outro jogador, o desempenho do jogador em questão será bem menor do que se ele estivesse disposto a negociar. Catan é um jogo muito dependente deste tipo de interação já que muitas vezes um jogador não irá conseguir todos os recursos que precisa para fazer suas construções ou comprar cartas de desenvolvimento, então o resultado encontrado neste experimento faz todo o sentido.

Após fazer simulações com reputações com valores altos para situações positivas e negativas, o passo seguinte foi testar como um agente *Smart Social 2* iria se comportar se, por conta de conhecermos o perfil social dos agentes Cooperativo e Antissocial, os categorizarmos como aliado e adversário, respectivamente. No caso dos *bots Fast* e *Smart*, a reputação permaneceu sendo iniciada com 0, já que ambos *bots* não possuem nenhuma característica social.

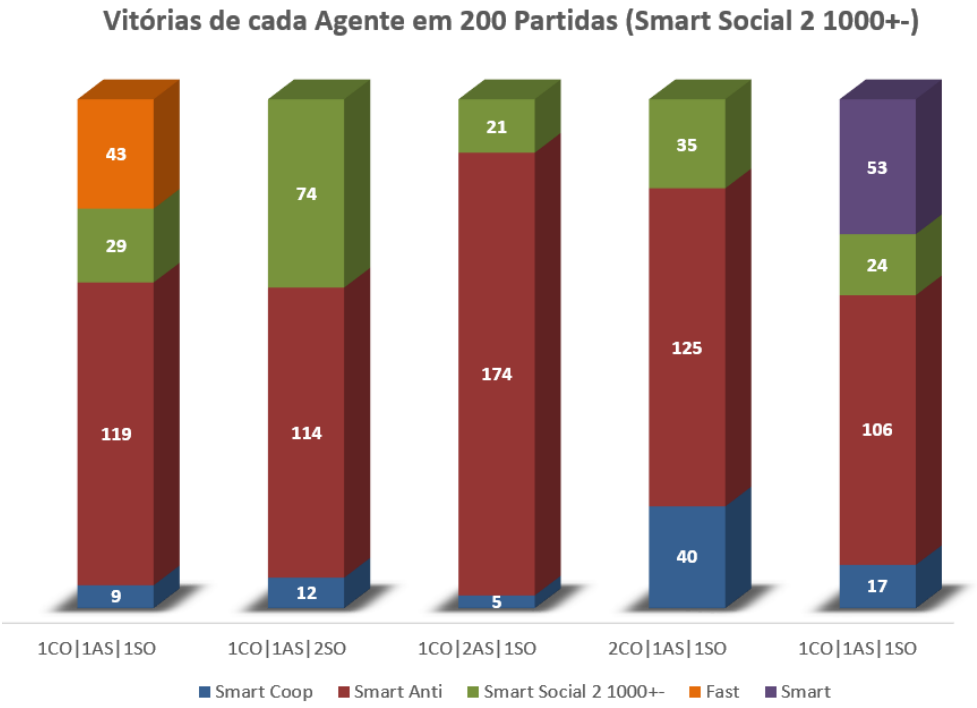
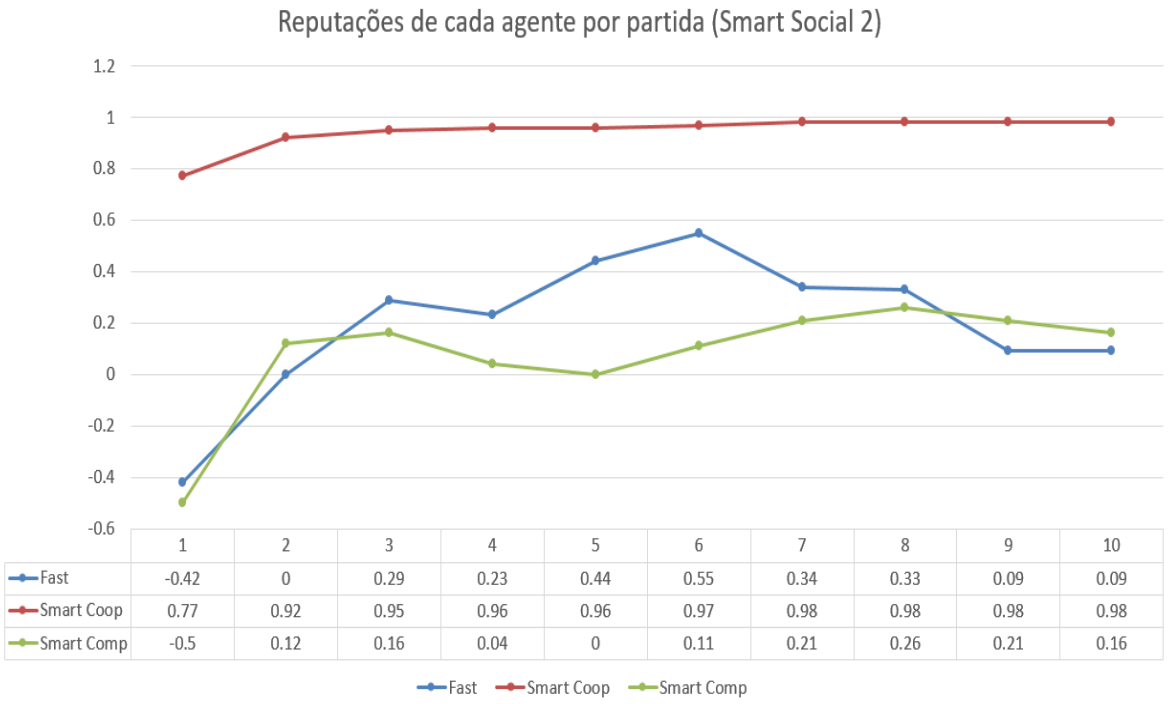


Figura 14: Simulação com 1000 partidas entre o agente *Smart Social 2 1000+-* e agentes versão *Smart*.

Nesta simulação, a quantidade de vitórias gerais obtidas pela variação do *bot Smart Social 2* foi levemente superior a sua versão original, obtendo um aumento de desempenho de 2.2%. O melhor cenário deste agente foi o caso com 2 *bots* Cooperativos, com um aumento de desempenho de 75%, enquanto que o pior cenário foi o caso com 2 *bots* Antissociais, com uma diminuição de desempenho de 25%. É interessante notar que, apesar desta versão do agente *Smart Social 2* não fazer nenhuma negociação com o agente Antissocial, o mesmo ainda consegue os recursos que precisa por meio de outros jogadores, e continua se mantendo como o *bot* com melhor desempenho. Até mesmo no cenário com dois agentes *Smart Social 2*, o agente Antissocial ainda consegue desbancá-los na maior parte do tempo, e se mantém com um número de vitórias parecido com o que teve na simulação sem memória.

Finalmente, a última simulação feita com memória teve o objetivo de entender como que as confianças dos jogadores variam para o agente *Smart Social 2* após

um conjunto pequeno de partidas, onde cada partida recebe como entrada as reputações obtidas na partida anterior, de modo a reconstruir as confianças finais de cada jogador na última partida. No caso da primeira partida, não houve nenhuma inicialização de reputação, de modo a deixar com que o próprio agente as construísse e atualizasse-as a cada partida.



	1	2	3	4	5	6	7	8	9	10
Fast	-0.42	0	0.29	0.23	0.44	0.55	0.34	0.33	0.09	0.09
Smart Coop	0.77	0.92	0.95	0.96	0.96	0.97	0.98	0.98	0.98	0.98
Smart Comp	-0.5	0.12	0.16	0.04	0	0.11	0.21	0.26	0.21	0.16

Figura 15: Simulação com 10 partidas entre o agente *Smart Social 2* com memória e agentes versão *Smart*.

Como é possível ver na imagem 15, apesar dos agentes *Fast* e *Smart Comp* terem tido uma reputação negativa na primeira partida, após ela todos os resultados foram positivos ou neutros, o que caracteriza com que o agente *Social* não os viu como adversários durante a maior parte da simulação. No caso do agente *Smart Coop*, tivemos o resultado esperado, onde o agente foi visto como aliado durante toda a partida, sem ter nenhuma interação negativa durante a simulação.

Considerando os perfis sociais de cada um dos agentes da partida, temos que o resultado do agente Cooperativo foi igual ao esperado, e o do *bot* Antissocial

foi o contrário do esperado (reputações negativas na maior parte do tempo). Isso provavelmente aconteceu pelo fato de que, apesar do *bot* Antissocial recusar todas as negociações recebidas, ele ainda acaba fazendo várias propostas ao agente *Smart Social 2*, o que é categorizado como uma interação positiva. No caso do agente *Fast*, não existia nenhuma previsão para sua reputação, já que sua estratégia não possui nenhum viés social.

5.1.1.5 Experimento fixo com diferentes reputações

Este experimento, que é a única simulação fixa apenas com *bots* deste trabalho, tem como objetivo analisar como diferentes reputações afetam o jogo do agente *Social*. Para analisar isto da melhor forma, foi decidido que o experimento seria determinístico, permitindo que cada uma das partidas da simulação tivessem como única diferença o valor inicial de reputação de cada jogador. Desta maneira, é possível testar diferentes reputações em um ambiente controlado, sem ter nenhuma outra variável afetando a comparação. Foram escolhidos os seguintes casos para serem testados:

- Normal (Reputação 0 para todos)
- Reputação positiva para todos (1000)
- Reputação negativa para todos (1000)
- Reputação positiva com *Coop*, negativa com *Comp* e neutra com *Fast* (1000)
- Memória oriunda dos resultados de reputação e confiança da partida Normal

Nestes cenários, o foco foi em analisar a quantidade de negociações que cada jogador fez com o banco e com outros jogadores, para entender como a mudança de confiança do agente *Smart Social 2* em relação aos outros *bots* mudou o decorrer das negociações.

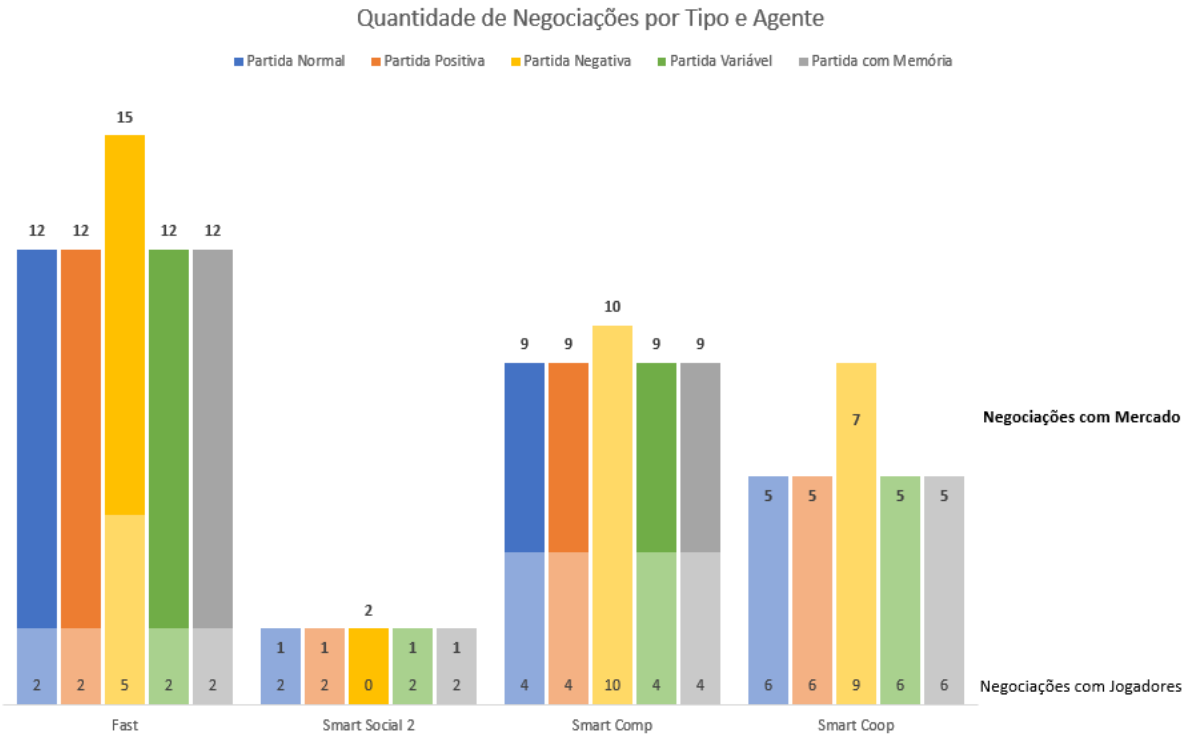


Figura 16: Simulação com 5 partidas idênticas entre o agente *Smart Social 2* com diferentes reputações e agentes versão *Smart*.

Na figura 16, temos uma representação da quantidade de negociações feitas por cada agente em cada um dos cenários de reputação aplicados no agente *Smart Social 2*. Cada conjunto de barras representa todos os resultados de um tipo específico de agente, diferenciando os cenários da simulação por meio de cores. Os valores apresentados representam o número de negociações com o mercado (representado pelas barras opacas, com seus valores na parte superior), e o número de negociações com outros jogadores (representado pelas barras translúcidas, com seus valores na parte inferior).

É bem interessante notar como que as variações de reputação acabaram não afetando a quantidade de negociações por partida na maioria dos cenários. O único caso que teve uma mudança na sua disposição de negociações foi no caso da partida onde o agente *Smart Social 2* foi inicializado com reputação negativa para todos os outros jogadores. Neste caso, houve 7 negociações a mais com o mercado, e 10

negociações a mais entre os jogadores, totalizando 17 negociações extras. O número de negociações extras foi bem maior do que o número de negociações que o agente *Smart Social 2* deixou de fazer em comparação com as outras partidas, o que nos leva a crer que, ao precisarem de recursos específicos que não iriam mais ser dados pelo *bot Smart Social 2* nesta partida, os outros agentes tiveram que negociar de outras maneiras ou até mesmo ficar sem eles, o que acabou modificando a estratégia deles ao ponto de precisarem de mais negociações para chegarem no objetivo da vitória.

A maior parte das partidas acabou tendo a mesma quantidade de negociações, justamente pelo fato de que apesar de termos tido 5 estratégias diferentes, 4 delas acabaram criando os mesmos resultados. Na partida normal, todos os três agentes acabaram sendo categorizados como aliados, e ao fazer o uso de memória, os três jogadores se mantiveram como aliados, com pequenas variações nas suas confianças. Por conta disso, a partida positiva acabou tendo o mesmo resultado, já que independente dos valores de confianças serem diferentes nesses 3 casos, todos ainda eram positivos, então tiveram a mesma estratégia durante a partida. Já no caso da partida variável, o único agente que ficou com confiança negativa foi o agente Antissocial, mas por conta de que na partida normal não houve nenhuma negociação entre o *bot Smart Social 2* e o *bot* Antissocial, a mudança de sua categorização de aliado para adversário não teve como ter impacto no número de negociações da partida.

Todas as partidas foram vencidas pelo agente Antissocial, que além de ter o melhor aproveitamento de terrenos (um pouco acima do segundo colocado, que foi o agente *Fast*), também é o agente que tem o melhor desempenho em partidas. A mudança no número de negociações causada pela mudança de reputações armazenadas no agente *Smart Social 2* conseguiu retardar esta vitória em 4 rodadas, mas não foi o suficiente para com que outro *bot* conseguisse triunfar. Para testar melhor este tipo de situação, o ideal seria simular uma partida fixa onde cada um dos jogadores tivesse um aproveitamento de terrenos relativamente parecido, e que o agente *Smart Social 2* fizesse negociações com todos os outros jogadores, garantido que qualquer mudança de reputação afetaria cada um dos cenários jogados.

5.1.2 Conclusão

Apesar dos resultados iniciais dos agentes deste projeto nos terem levado a conclusão de que as versões dos mesmos que foram usadas não eram as últimas versões desenvolvidas por do Couto [2], foi possível modificá-los de maneira a melhorar seus desempenhos. O agente do tipo Social foi o que teve a melhor evolução, com sua última versão (*Smart Social 2*) que possuía modificações na estratégia de vitória a longo prazo (estratégia *Smart*) e no comportamento social. Já os outros dois agentes não tiveram uma evolução expressiva. O agente do tipo Antissocial (em sua versão *Smart Comp*) teve um desempenho similar, enquanto que o agente do tipo Cooperativo (em sua versão *Smart Coop*) teve um desempenho reduzido ao ser testado com outros agentes que tivessem a mesma estratégia de vitória a longo prazo.

5.2 SIMULAÇÕES COM JOGADOR

Para poder testar os *bots* com jogadores de verdade, foi necessário primeiramente estruturar o projeto e decidir como permitir com que várias pessoas diferentes jogassem, gerassem resultados e passassem-os de volta para análise.

Durante este projeto, a maneira mais comum de fazer todo este processo era executar o SOCSim com os parâmetros necessários para gerar uma partida, e após isso, chamá-lo novamente para pegar o resultado anterior e analisá-lo com algumas métricas. Estas duas ações foram executadas por meio de um arquivo *batch* enquanto as simulações eram feitas apenas com *bots*, e depois foram manualmente executadas quando as simulações foram feitas com um jogador humano. Entretanto, este procedimento não poderia ser feito pelos jogadores, já que isso exigiria mais conhecimento do que o que um usuário comum tem, diminuindo o conjunto de possíveis participantes. Além disso, executar o SOCSim diretamente envolveria outros problemas, como exigir mais memória física disponível, além da probabilidade de que aconteçam erros durante o processo de pegar a saída do JSettlers e transformar em arquivos para leitura do SOCSim.

Por conta destes motivos, a melhor solução encontrada foi passar o mínimo pos-

sível de arquivos para os usuários que fossem testar, junto com o menor número possível de instruções necessárias para executar uma simulação e gerar resultados. Para isto, foi feito um arquivo zip, que consiste de 4 arquivos:

- JSettlers.jar
- Instruções.txt
- Jogar Partida 1.bat
- Jogar Partida 2.bat

Este conjunto mínimo ocupa pouco espaço de memória (um pouco menos de 900 KB), e apesar de precisar de um arquivo de instruções, é bastante intuitivo e simples. Seu único requerimento para executar com sucesso é possuir o Java 8 ou superior instalado na máquina.

O arquivo JAR contém o JSettlers, e é necessário para executar o jogo. Ele seria o mínimo necessário para testar os *bots* em qualquer situação possível.

O arquivo de instruções explica como executar o jogo, mas na verdade é focado em explicar um pouco sobre o JSettlers, já que o simulador não é muito conhecido e pode acabar sendo confuso para jogadores de Catan que nunca o experimentaram anteriormente. Além disso, ele também possui todas as instruções necessárias sobre como enviar os dados gerados por ele mesmo.

Os arquivos *batch* são os inicializadores do jogo, e os responsáveis por fazer com que o *output* do JSettlers seja salvo em arquivos log, que serão enviados de volta para análise dos resultados. Por conta do experimento feito conter duas partidas, foi necessário ter dois arquivos separados, onde cada um tem parâmetros diferentes para inicializar o jogo.

Para executar todos estes arquivos, é necessário utilizar alguma versão do sistema operacional Windows, além de precisar do Java 8 ou superior, mas por conta da diferença de configurações entre os participantes das simulações, não são conhecidas as versões específicas usadas.

Por conta do objetivo das simulações com jogadores humanos ser para ouvir as opiniões dos mesmos, e entender o quão humano foram os comportamentos dos agentes que possuem vieses sociais, as métricas deste tipo de simulação tiveram que ser bem diferentes das utilizadas nas simulações feitas exclusivamente com *bots*. Enquanto que nessas simulações as métricas eram focadas em vitórias e alguns outros atributos, como quantidade de negociações feitas e aproveitamento de terrenos, estas simulações não focaram em nenhuma estatística do jogo, e sim em um questionário[1] que fazia uma série de perguntas para entender como foi a experiência do jogador humano. Além disso, para não criar nenhum tipo de influência nos jogadores ao responder sobre as partidas, não foi explicada a diferença entre os agentes de cada partida.

No início do formulário existem duas perguntas sobre o sexo e idade do participante, apenas para dar uma ideia do público participante. Após isso, temos duas perguntas relacionadas ao jogo Catan, onde é perguntado se o participante tem alguma experiência com Catan ou não, e se o mesmo já jogou Catan anteriormente com *bots*.

A parte seguinte do questionário foca nas partidas jogadas pelo participante, e é composta por 5 perguntas. A primeira pergunta questiona qual das duas partidas foi a preferida do participante, e após isso, a segunda pergunta oferece 4 opções de possíveis motivos principais para a preferência. Além disso, é dada uma quinta opção chamada "Outro", que vem acompanhada da terceira pergunta, que pede para o participante explicar qual foi este outro motivo de preferência. Com isso, o participante deve responder mais duas perguntas, ambas com o mesmo conteúdo, mas sobre cada uma das partidas. Estas duas perguntas são sobre o quão realístico estas duas partidas foram, comparado com experiências anteriores do participante com Catan entre pessoas, ou, no caso onde o participante nunca havia jogado Catan antes, comparado com experiências anteriores com outros jogos de tabuleiro.

Por fim, a última pergunta é opcional, e apenas por sugestões, críticas ou comentários a fazer sobre os *bots*, as partidas, ou quaisquer outros aspectos do experimento.

[1] `https://goo.gl/forms/KJO2tcG6qiKAHX6n2`

5.2.1 Simulações

Por conta de tempo e disponibilidade de participantes, apenas uma simulação com pessoas foi feita, e o número de participantes foi pequeno, sendo de apenas 6 pessoas. Além disso, já que a maioria dos jogadores nunca teve contato com o JSettlers e, portanto, nunca teve uma experiência de jogo com os seus *bots*, foi decidido que, para melhorar a análise sobre os resultados das partidas, cada jogador que participasse do experimento iria jogar duas partidas:

- Partida com 2 *bots Fast*, e 1 *Smart*
- Partida com 1 *bot Smart Social 2*, 1 *Smart Coop* e 1 *Smart Comp*

Dessa maneira, se tornou mais fácil do jogador expressar as diferenças de emoção que ele sentiu com os *bots* sociais no formulário, já que existe uma comparação direta com os *bots* padrões do JSettlers. Além disso, foi decidido fixar o mesmo tabuleiro e todas as outras variáveis aleatórias nos dois casos para que a estratégia do jogador pudesse ter algum nível de similaridade com sua partida anterior, com o objetivo de criar mais familiaridades entre as duas partidas para facilitar a comparação do jogador na hora de responder a enquete sobre os jogos. Logo, este experimento determinístico utilizou os mesmos valores de dados, terrenos e outras variáveis aleatórias do que as usadas no experimento determinístico com variação de inicialização de reputação.

Infelizmente, por questões de tempo e disponibilidade de participantes, apenas 6 pessoas puderam participar do experimento. Apesar disso, foi possível colher dados interessantes sobre a experiência dos mesmos.

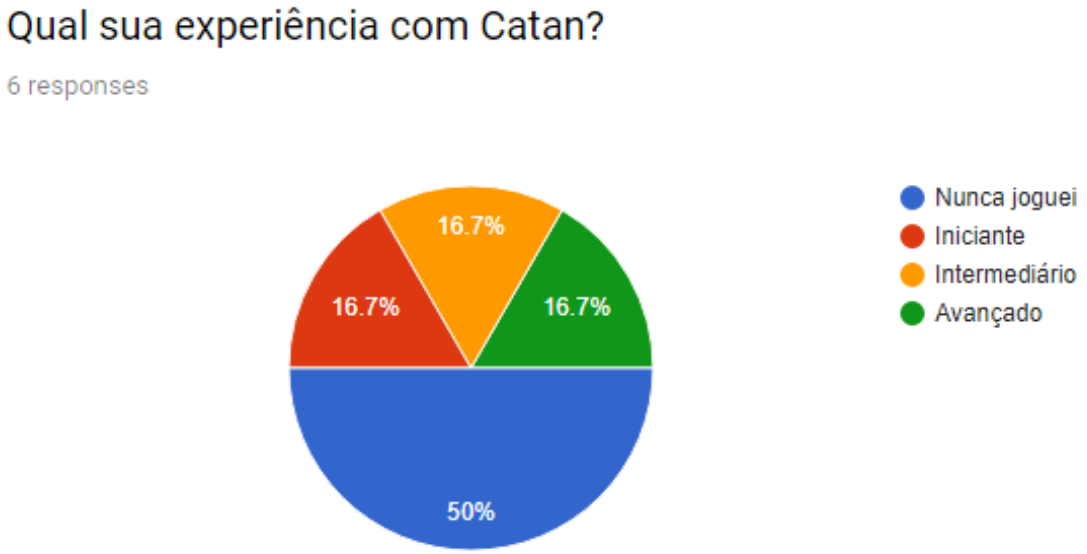


Figura 17: Respostas dadas na primeira pergunta do questionário.

Os participantes do experimento estavam bem divididos. Enquanto que metade não tinha tido contato com Catan anteriormente, a outra metade possuía diferentes níveis de habilidade, com 1 participante para cada um dos níveis: iniciante, intermediário e avançado.

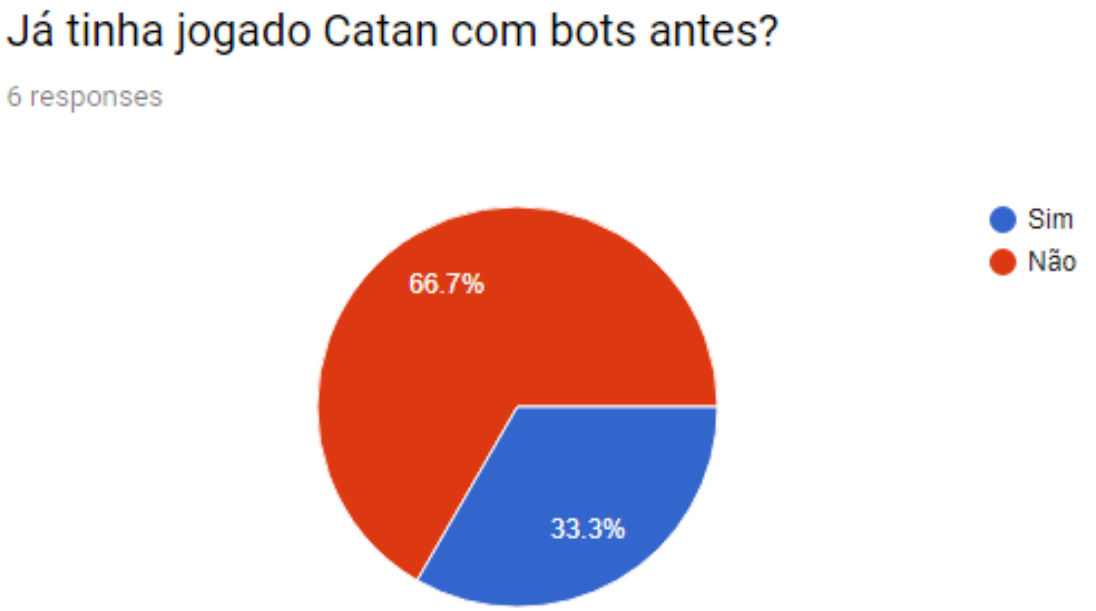


Figura 18: Respostas dadas na segunda pergunta do questionário.

Apesar de metade dos participantes terem tido experiência anterior com Catan, apenas 2 já tinham jogado com *bots* anteriormente, seguindo a linha de que a maioria dos jogadores que já tiveram contato com Catan fizeram por meio do jogo de tabuleiro, e não em jogos de computador.

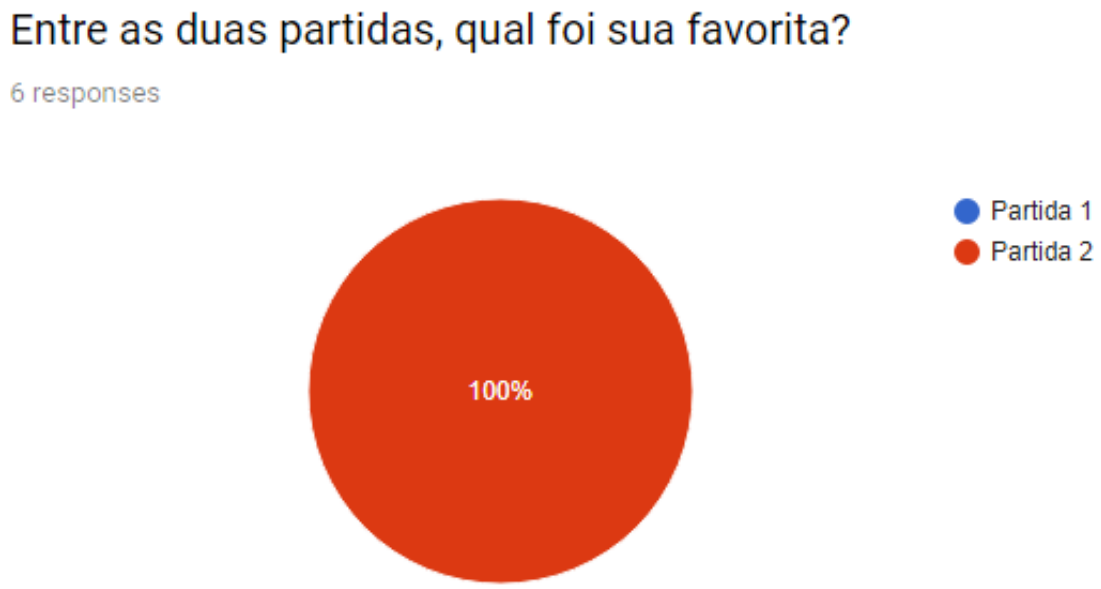


Figura 19: Respostas dadas na terceira pergunta do questionário.

No momento de analisar os resultados das duas partidas, um fato interessante ocorreu: todos os participantes preferiram a segunda partida, com os agentes versão *Smart*, e a última versão do agente *Social*. Apesar do objetivo do trabalho ser com que a segunda partida fosse mais interessante para os participantes, não era sabido se isto realmente iria acontecer, ou se iria acontecer com uma porcentagem alta dos participantes.

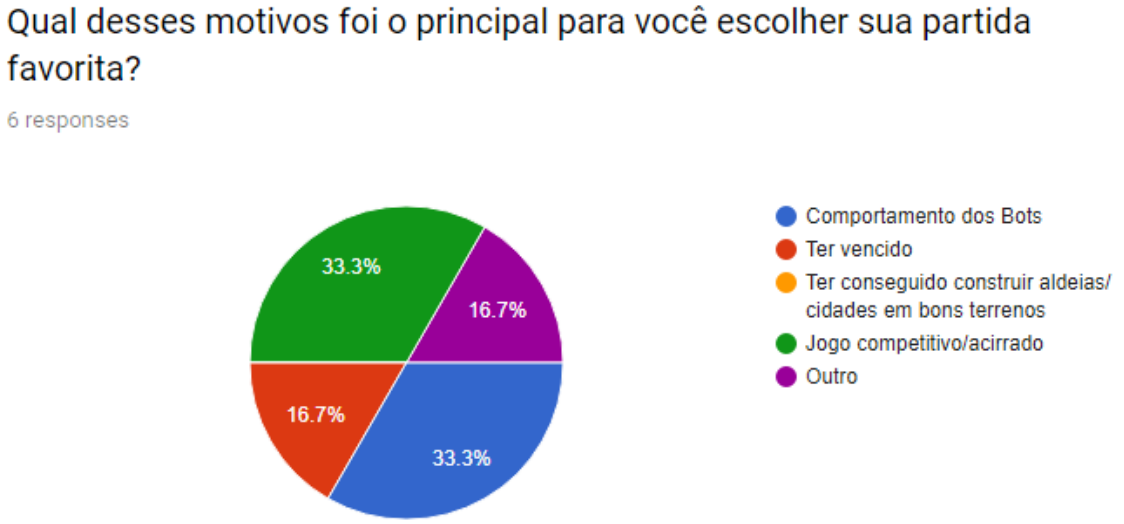


Figura 20: Respostas dadas na quarta pergunta do questionário.

A pergunta seguinte era diretamente relacionada a anterior, com o objetivo de entender quais foram os principais motivos que a segunda partida foi a mais escolhida, e entender se o motivo foi relacionado ao desempenho dos novos agentes sociais, ou se foi por motivos não relacionados. Pelo o que podemos ver, 5 dos resultados foram relacionados aos *bots*, onde 2 foram por conta do comportamento dos mesmos, outros 2 foram por conta dos *bots* novos terem conseguido proporcionar uma experiência acirrada ao jogador, e apenas 1 foi por um misto entre os dois motivos citados anteriormente (resposta dada em "Outro").

Esta e a terceira pergunta foram muito importantes, pois mostraram como a partida com *bots* sociais fez diferença na experiência dos jogadores, ainda que de maneiras diferentes. Entretanto, ainda é necessário analisar melhor se essa experiência social foi mais realística ou não, por meio das próximas duas perguntas.

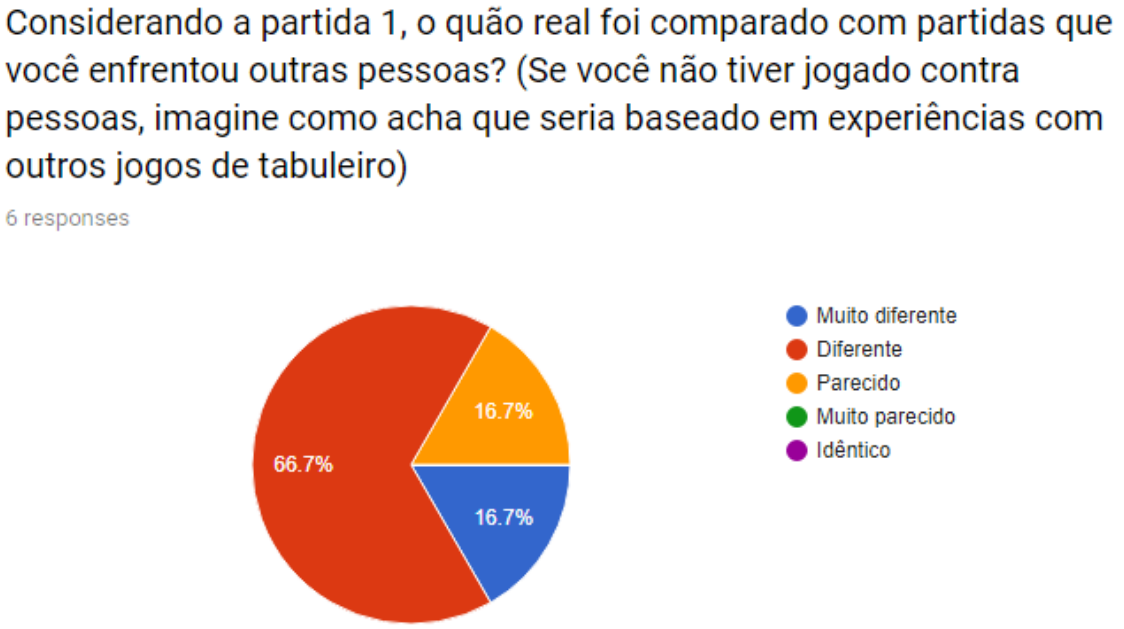


Figura 21: Respostas dadas na quinta pergunta do questionário.

Antes de analisar se a segunda partida foi realística comparada com experiências entre jogadores humanos, foi necessário verificar se a primeira partida ficou distante disso, pois considerando que o objetivo principal da abordagem social é criar um novo tipo de experiência para jogadores, isso não poderia ser atingido se a partida com *bots* normais já proporcionasse isto.

Como podemos ver pelas respostas dadas, a experiência com *bots* normais não se aproximou de um jogo entre humanos. Cinco participantes consideraram a partida em questão como diferente ou muito diferente de uma experiência real, enquanto que apenas 1 dos participantes considerou a experiência parecida. Ninguém considerou a primeira partida como muito parecida ou idêntica a um jogo entre jogadores humanos.

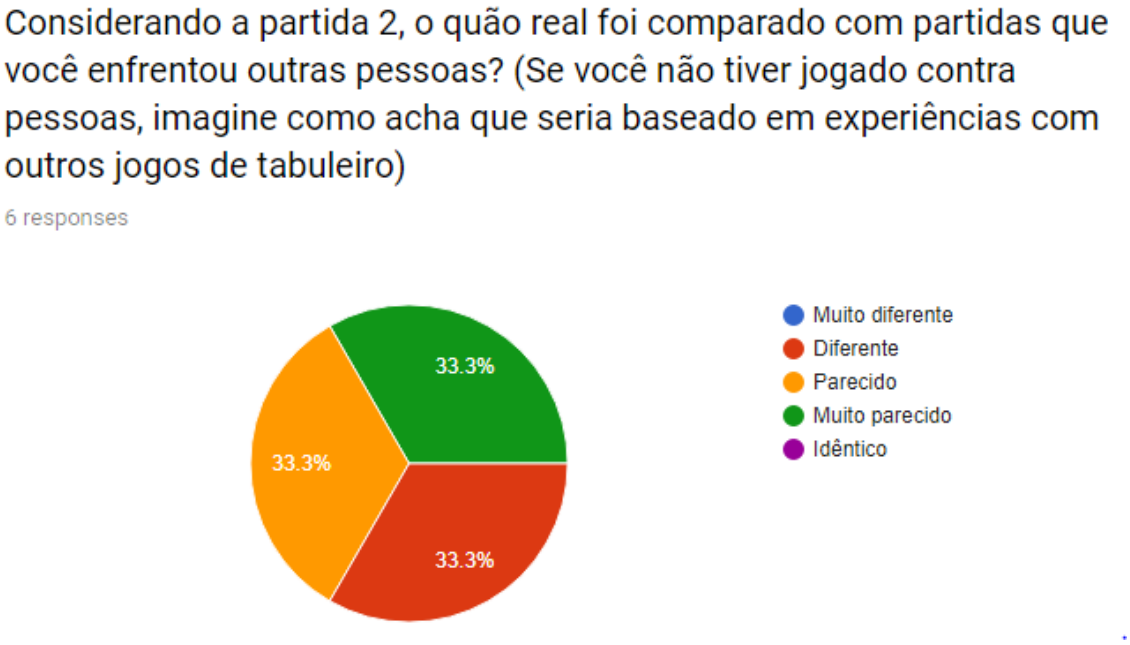


Figura 22: Respostas dadas na sexta pergunta do questionário.

Por fim, temos a análise sobre quão próximo a experiência da partida 2 ficou de um jogo entre pessoas. O resultado foi bastante positivo, o que era de se esperar considerando os resultados encontrados nas perguntas 3 e 4. Quatro participantes do experimento consideraram a partida com *bots* sociais parecida ou muito parecida com experiências contra jogadores humanos, e apenas 2 consideram-na diferente, sem nenhum participante considerando a experiência como muito diferente. Com isso, podemos afirmar que a maior parte dos participantes sentiu uma diferença ao enfrentar os agentes com viés social, o que os lembrou de partidas contra outros jogadores, e ao mesmo tempo tornou a experiência deles mais interessante e divertida. Entretanto, é importante frisar que, dos 3 participantes que possuíam experiência com Catan, apenas 1 deles achou a experiência parecida com enfrentar outras pessoas.

5.2.2 Conclusão

Apesar deste experimento ter sido feito com um número muito baixo de pessoas, e por conta disso ter um resultado que exige uma análise mais detalhada e longa, as

respostas obtidas foram positivas, e levam a crer que realmente existe um ganho com a abordagem proposta neste trabalho. Apesar deste ganho não aparentar ser sobre uma proximidade da jogabilidade dos agentes sociais ao comportamento humano, os resultados obtidos foram positivos quanto a diversão proporcionada pelos agentes em comparação com *bots* tradicionais. Considerando tudo que foi feito neste trabalho até este ponto, a melhoria mais significativa para este projeto seria expandir este experimento, colhendo mais dados de uma gama maior de jogadores, e também adicionar outras perguntas no questionário, com o objetivo de entender a opinião de jogadores sobre cada um dos *bots* testados.

6 CONCLUSÃO

6.1 VISÃO GERAL

Dentro da indústria de *videogames*, muitos jogos se fazem do uso de IA para acrescentar uma camada de raciocínio nos seus *bots* por diferentes motivos, que vão desde criar um personagem mais complexo para melhorar uma história, até criar um adversário desafiador para um jogador que quer mais dificuldade. Do Couto [2] viu neste mercado uma oportunidade para estudar maneiras de criar *bots* que tivesse uma abordagem diferente da rotineira na indústria, e dentro da sua dissertação de mestrado, implementou diferentes agentes que faziam esta tarefa, além de identificar pontos que poderiam ser melhorados no seu trabalho.

Este projeto continuou a proposta desta dissertação de mestrado, primeiramente estudando os sistemas de reputação e confiança utilizados na dissertação, para que fosse possível entender como cada um dos agentes com viés social funciona. Estes sistemas são comumente utilizados em outras áreas que precisam garantir a confiança de dois agentes diferentes que estão interagindo entre si, como por exemplo em *e-commerce*. Apesar disso, eles se mostraram úteis no contexto deste projeto, servindo como uma ótima maneira de analisar outros jogadores e permitir com que os agentes sociais conseguissem decidir como se comportar em cada situação.

A implementação destes agentes aconteceu dentro do jogo de tabuleiro conhecido como Catan. Este jogo tem uma dinâmica e regras que fazem com negociações entre jogadores sejam um fator crucial para vencer, além de permitir com que os jogadores interajam de outras maneiras também. Este ambiente cheio de pontos de interação se mostrou adequado para testar o desempenho dos agentes criados, além de possuir uma implementação *open-source* [11] em Java que já possuía *bots* com boas estratégias de jogo.

Já existiam três agentes criados na dissertação de mestrado: Social, Cooperativo e Antissocial. Além desses três *bots*, também exitiam outros dois, provenientes do JSettlers: *Fast* e *Smart*. Após um estudo de funcionamento dos três agentes sociais

e das diferenças de estratégia entre os *bots* padrões do JSettlers, foi possível criar variações dos *bots* sociais, com diferentes abordagens não somente na estratégia de jogo, mas também em seus vieses sociais: *Smart Coop*, *Smart Comp*, *Social 2*, *Smart Social* e *Smart Social 2*.

Cada um destes agentes foram testados em diferentes experimentos. A maior parte deles foi feita apenas entre *bots*, com o intuito de comparar o desempenho das novas versões com os *bots* originais do projeto. Além disso, também houve experimentos para testar como a inicialização de reputação e memória influenciam o comportamento do agente Social em diferentes cenários, desde inicializações positivas e negativas em partidas aleatórias, até a variação de reputação e o uso de memória em partidas determinísticas, analisando o impacto das mudanças no decorrer do jogo. Por fim, também houve experimentos com jogadores, pois não seria possível mensurar se o objetivo do projeto foi alcançado sem a opinião de pessoas que interagissem com os agentes.

Com todo o estudo e resultados obtidos neste projeto, foi possível alcançar o objetivo proposto, criando uma experiência mais interessante para jogadores humanos, e também melhorar os resultados apresentados na dissertação de mestrado utilizada de base neste projeto. Apesar disso, ainda existem muitas possibilidades nesta área, não faltando caminhos possíveis para melhorar o que foi apresentado neste trabalho, na sua dissertação de mestrado antecessora e no mercado de jogos eletrônicos.

6.2 TRABALHOS FUTUROS

Existem diversas oportunidades em aberto neste trabalho para melhorias, seja por falta de tempo, recursos ou puro desconhecimento. Dentro do que foi proposto, seria possível fazer novos experimentos, principalmente na área de inicialização de reputação e memória, visando compreender melhor como reputações podem afetar o cálculo de confiança em diferentes situações. Além disso, também seria interessante fazer experimentos mais detalhados e com um número maior de participantes na área de partidas com jogadores humanos, com o objetivo de entender mais a fundo quais

aspectos implementados neste trabalho foram positivos e quais foram negativos.

Saindo do escopo apresentado neste projeto, também existem várias boas oportunidades de estudo. Poderiam ser feitas implementações destes agentes em outros jogos, permitindo com que fosse possível testar seus desempenhos juntamente com estratégias totalmente diferentes das que são usadas em Catan, e com potenciais novos pontos de interação para serem analisados e calculados em forma de reputação. Também seria possível procurar outros perfis sociais para NPCs na literatura, com o objetivo de criar um novo tipo de agente que não se encaixe em Social, Antissocial e Cooperativo. Ainda dentro deste aspecto, seria interessante testar como um agente misto, que fosse se adaptando durante o jogo entre os perfis sociais deste trabalho, se comportaria, e como seu estilo de jogo é mais próximo ou distante da experiência de um jogador humano.

6.3 CONSIDERAÇÕES FINAIS

Apesar das limitações de tempo e conhecimento, este trabalho conseguiu com sucesso continuar os estudos feitos por do Couto [2], implementando novos agentes, e também conseguindo testá-los em um ambiente com jogadores, o que permitiu agregar valor a este projeto. Além disso, foi possível estudar IA, sistemas de recomendação e confiança e metodologia científica, permitindo usar alguns conceitos aprendidos em sala de aula de maneira mais prática, e também adquirir novos conhecimentos que não puderam ser ensinados durante os anos de graduação. Por fim, a continuação dos estudos feitos neste projeto possuem um grande valor, seja para a indústria dos jogos, que poderá alcançar novos consumidores que estão atualmente insatisfeitos com a experiência proporcionada por *bots* em jogos, seja para os atuais consumidores de jogos, que esperam ter experiências melhores do que suas anteriores por conta da evolução da tecnologia e dos novos estudos gerados dentro da área.

REFERÊNCIAS

[1] Chaslot, G., Bakkes, S., Szita, I., e Spronck, P. Monte-carlo tree search: A new framework for game ai. In *AIIDE* (2008).

[2] do Couto, F. S. Confiança e reputação para jogos. Tese de Mestrado, Programa de Pós-Graduação em Informática da Universidade Federal do Rio de Janeiro, 2013.

[3] Encyclopedia of Mathematics. Minimax principle. `https://www.encyclopediaofmath.org/index.php/Minimax_principle`.

[4] Guhe, M., e Lascarides, A. Game strategies for the settlers of catan. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on* (2014), IEEE, pp. 1–8.

[5] Jøsang, A. *Modelling Trust in Information Security*. PhD thesis, Norwegian University of Science and Technology, 1998.

[6] Jøsang, A. Trust and reputation systems. In *Foundations of security analysis and design IV*. Springer, 2007, pp. 209–245.

[7] Jøsang, A., e Ismail, R. The beta reputation system. In *Proceedings of the 15th bled electronic commerce conference* (2002), vol. 5, pp. 2502–2511.

[8] Mascarenhas, S., Prada, R., Paiva, A., e Hofstede, G. J. Social importance dynamics: A model for culturally-adaptive agents. In *International Workshop on Intelligent Virtual Agents* (2013), Springer, pp. 325–338.

[9] Osborne, F. A new approach to social behavior simulation: the mask model. In *International Conference on Interactive Digital Storytelling* (2011), Springer, pp. 97–108.

[10] Tan, C. T., e Cheng, H.-l. Personality-based adaptation for teamwork in game agents. In *AIIDE* (2007), pp. 37–42.

[11] Thomas, R. S. *Real-time decision making for adversarial environments using a plan-based heuristic*. PhD thesis, Northwestern University, 2003.