CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA – CEFET/RJ

Amê: Framework para Avaliação de Algoritmos para Jogos de Competição Baseados em Cartas

Ana Beatriz Cruz
Sabrina Pontes Serique

Prof.º Orientador: Eduardo Ogasawara, D.Sc

Rio de Janeiro
fevereiro/2014

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA – CEFET/RJ

Amê: Framework para Avaliação de Algoritmos para Jogos de Competição Baseados em Cartas

Ana Beatriz Cruz
Sabrina Pontes Serique

Projeto Final II apresentado em cumprimento às normas do Departamento de Educação Superior do CEFET-RJ, como parte dos requisitos para obtenção do título de Tecnólogo em Sistemas para Internet

Prof.º Orientador: Eduardo Ogasawara, D.Sc

Rio de Janeiro
fevereiro/2014

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

C957 Cruz, Ana Beatriz

Amê : framework para avaliação de algoritmos para jogos de competição baseadas em cartas / Ana Beatriz Cruz [e] Sabrina Pontes Serique.—2014.

35f. + apêndices : il.color. , grafs. , tabs. ; enc.

Projeto Final (Tecnólogo) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2014.

Bibliografia : f. 34-35

Orientador : Eduardo Ogasawara

1. Framework (Programa de computador). 2. Algoritmos. 3. Jogos de carta. 4. Inteligência artificial. I. Serique, Sabrina Pontes. II. Ogasawara, Eduardo (Orient.). III. Título.

CDD 005.3

Dedicamos a todos que,

de alguma forma,

tornaram este trabalho possível.

AGRADECIMENTOS

Agradecemos as nossas famílias pela compreensão para conosco, nossos pais pela dedicação, apoio e amor nos dados. Aos nossos amigos pelos momentos de distração, palavras e sugestões que contribuíam direta ou indiretamente no desenvolvimento do trabalho. Ao nosso orientador, Prof.º Eduardo Ogasawara pela dedicação, contribuição e compreensão, para a concretização desse trabalho. Ao colega, amigo e colaborador Leonardo de Souza Preuss, pela ajuda, paciência, conhecimento e disponibilidade. A todos os professores que direta ou indiretamente nos auxiliaram na realização deste trabalho. Aos colegas e amigos Anderson Luiz M. dos S. Silva, Fábio da Mota Rosa e Viktor Hugo Barreiro dos Santos pelo auxilio durante a realização do experimento e a todos os voluntários que se dispuseram a participar do experimento. Ao CEFET pela busca incessante em qualificar os seus alunos, oferecendo suporte e conhecimento, os quais foram essenciais para o desenvolvimento desse projeto. Finalmente, agradecemos à FAPERJ e ao CNPq pelo fomento e bolsas PIBIC concedidas.

"Always in motion the future is."

Yoda

RESUMO

Este trabalho visa o estudo e implementação do *framework* Amê, que possibilita a comparação de diferentes algoritmos de Inteligência Artificial (IA) no contexto de jogos de competição baseados em cartas. A fim de viabilizar tal avaliação, buscou-se um jogo com alto dinamismo e com regras bem definidas que fosse capaz de se tornar um exemplo prático para os diferentes níveis de complexidade de cada algoritmo usado, tornando-os assim mensuráveis entre si. Para tanto, adotadou-se um jogo de cartas tradicional japonês, denominado Hanafuda. O jogo escolhido tem como diferencial a possibilidade de se poder ganhar a partir do uso de diferentes estratégias, tornando-se um contexto propício para avaliação de diferentes algoritmos de IA. Sendo assim, quanto mais eficiente for o algoritmo, mais difícil se torna para o usuário final vencer do computador. A partir desta ótica, o *framework* foi desenvolvido e avaliado na plataforma móvel Android. Os nossos resultados indicaram que o Amê está apto a apoiar diferentes algoritmos.

Palavras-chave: Framework, Jogos, Inteligência Artificial, competição.

ABSTRACT

This work targets to study and develop Amê framework, which makes possible the comparison of different algorithms for Artificial Intelligence (AI) in the context of competitive card-games. In order to execute this evaluation, we used a dynamic and well-defined card-game with rules that makes it possible to implement different levels of complexity of each algorithm used, thereby making them measurable together. Therefore, we adopted a traditional Japanese card game, named Hanafuda. The selected game has the possibility for the player to win using different strategies, which makes it adequate to measure the performance of different AI algorithms. To this extend, the more efficient is the algorithm, the harder it is for the final user to challenge the computer. Under such approach, the framework was developed and evaluated on top of Android mobile platform. Our results indicated that Amê was able to support different versions of algorithms.

Keywords: Framework, Game, Artificial Intelligence, competition.

SUMÁRIO

I Introdução 1

II Jogos de Competição Baseados em Baralhos 3

2.1 Hanafuda 3

2.1.1 As Cartas 4

2.1.2 Como Jogar 6

2.1.3 Regras Gerais do Hanafuda 7

2.1.4 O início 7

2.1.5 A progressão do jogo 7

2.1.6 O Fim 8

2.1.7 Pontuação das Cartas e Combos 9

2.1.8 Pontuação de cartas separadas 9

2.1.9 Yaku (combos especiais) 10

2.1.10 Jogos Fortes 11

2.2 Algoritmos de Inteligência Artificial 12

2.2.1 Algoritmo Aleatório 12

2.2.2 Algoritmo Guloso (Greedy) 12

2.2.3 Algoritmo Minmax 13

2.3 Trabalhos relacionados 14

III Framework Amê 17

3.1 Android 17

3.2 Arquitetura 20

3.2.1 Estruturação da Camada *Model* 21

3.2.2 Estruturação da Camada *View* 23

3.2.3 Estruturação da Camada *Controller* 24

3.3 Projeto de Interface Gráfica 25

IV Experimentação 27

4.1 GQM 27

4.1.1 Nível Conceitual 28

4.1.2 Nível Operacional 28

4.1.3 Nível Quantitativo 30

V Conclusão 33

REFERÊNCIAS BIBLIOGRÁFICAS 34

APÊNDICE A – Manual de Instalação do *framework* Amê 36

APÊNDICE B – Experimentação 40

LISTA DE FIGURAS

Figura 1 - Jogo Hanafuda 6
Figura 2 - Camadas do Software Android (Android Developers 2013a) 17
Figura 3 - Ciclo de vida de uma Atividade (Android Developers 2013c) 19
Figura 4 - Diagrama de Classes Conceitual 20
Figura 5 - Camada *Model* 21
Figura 6 - Camada *View* 23
Figura 7 - Camada *Controller* 24
Figura 8 – Projeto de Interface – (a) tela inicial, (b) tela do jogo; (c) tela de pontuação 25
Figura 9 – Estrutura hierárquica do modelo GQM 27
Figura 10 - Tela de formulário com perguntas sobre o jogo 30
Figura 11 - Gráfico probabilístico de saldos de pontuação 31
Figura 12 - Percepção do Player 1 quanto ao oponente mais difícil 32

LISTA DE TABELAS

Tabela 1 - Famílias das Cartas....4
Tabela 2 - Pontuação das Cartas....9
Tabela 3 - Pontuação das Combos de três cartas....10
Tabela 4 - Combinações que formam Jogos Fortes....11
Tabela 5 - Hipóteses....28
Tabela 6 - Relação de cor de Android e Algoritmo....29

LISTA DE SIGLAS

ADT – Android Development Tools

API – Application Programming Interface

AVD – Android Virtual Device

GQM – Goal - Question - Metric (Objetivo - Perguntas - Métrica)

GUI – Graphic User Interface

IA – Inteligência Artificial

IDE – Integrated Development Environment (Ambiente Integrado de Desenvolvimento)

MVC – Model - View - Controller (Modelo - Visão - Controle)

QEMU – Quick EMUlator

SDK – Software Development Kit (Kit de Desenvolvimento de Software)

SQLite – Structured Query Language Lite

UML – Unified Modeling Language (Linguagem de Modelagem Unificada)

USB – Universal Serial Bus

XML – eXtensible Markup Language

I Introdução

Um dos grandes objetivos da Inteligência Artificial (IA) consiste em trabalhar a racionalidade do pensamento humano na resolução de problemas (Russell e Norvig 2009). Na área de jogos de competição nos quais o adversário do usuário é o computador, a utilização de técnicas de inteligência artificial serve como grande estímulo para a elaboração de algoritmos. Note-se que há muitos jogos desenvolvidos visando explorar diferentes algoritmos de IA.

Os jogos nos quais os cenários ficam totalmente abertos, como xadrez e dama, acabam deixando o jogo um pouco monótono na perspectiva do usuário. Em contra partida, os jogos nos quais a imprevisibilidade faz parte presente, tornam-se muito atraentes para os usuários. Porém, alguns jogos que tem este teor de imprevisibilidade são tão sofisticados e envolvem vários outros aspectos como usabilidade, computação gráfica, dentre outros que se pode acabar perdendo o foco da implementação do algoritmo de IA.

Um grupo de jogos interessantes sob a ótica de IA são os jogos de competição baseados em cartas. Estes jogos, onde se tem o conceito de pegar cartas do monte faz com que as possiblidade mudem a cada rodada. O pensamento humano precisa estar em constante adaptação para produzir bom desempenho. Todavia, jogos mais interessantes, como, por exemplo, buraco, demandando quatro jogadores. Isto novamente requer uma interface gráfica rica e algoritmos de rede para possibilitar tão interação. Novamente, pode-se perder o foco da elaboração do algoritmo de IA propriamente dito.

Para piorar, a maioria dos jogos de cartas envolvendo apenas dois jogadores tem regras tão simplórias que geram poucas possibilidades de espaço de busca. A implementação de algoritmos para eles acaba ficando bem limitada, pois o espaço de busca é muito restrito. Felizmente, neste contexto, restam dois jogos interessantes para se trabalhar. O primeiro é o *Poker*, amplamente conhecido no mundo e muito rico em termos de possiblidade. Entretanto, a possibilidade de blefe traz características da psicologia humana que exigem o desenvolvimento de algoritmos que fazem percepção do comportamento do usuário (Billings et al. 2002), tornando os algoritmos complexos demais.

O segundo é o Hanafuda, que é um jogo de baralho japonês. Este jogo é tradicionalmente jogado no Japão, mas que foi introduzido e disseminado no Brasil pela imigração. Ele possui 48 cartas, que são divididas em doze grupos de quatro (cada grupo representando um mês do ano). O jogo tem como característica marcante a possibilidade de uso de diferentes estratégias de competição, torna-se uma boa base para o estudo de

algoritmos de Inteligência Artificial (IA) para jogos de carta nos quais o usuário joga contra o computador. Um jogo baseado no Hanafuda possibilita a adoção de diferentes estratégias para se conseguir ganhar. Sendo assim, algoritmos que implementam diferentes heurísticas podem ser elaborados para tornar o computador um oponente desafiador para o usuário. Para a exploração destes algoritmos foi implementado um *framework*, denominado Amê, capaz de avaliá-los.

Optou-se pela plataforma Android para servir de arena para avaliação dos algoritmos, uma vez que é uma plataforma já muito utilizada na criação de aplicativos. Ela viabiliza variedade de recursos de interatividade e facilita o desenvolvimento de aplicativos por meio do seu ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) do Eclipse. O jogo traz ainda, como diferencial, a inclusão da cultura japonesa, o que o torna bem estimulante para que os alunos possam pesquisar e expandir novos algoritmos a partir das primitivas oferecidas pelo *framework*

Neste trabalho o objetivo principal foi implementar e avaliar uma versão inicial do *framework* Amê para desenvolvimento de algoritmos para jogos de competição utilizando o Android, de forma a criar um jogo de competição baseado no Hanafuda. Para uma primeira avaliação do *framework*, foram implementados dois algoritmos iniciais: aleatório (Russell e Norvig 2009) e guloso (Cormen et al. 2009). As principais contribuições deste projeto final são: (i) aprendizado das regras e estratégias básicas do Hanafuda; (ii) analise e estudo da plataforma Android; (iii) elaboração dos algoritmos aleatório e o guloso; (iv) avaliação do *framework* Amê.

Este trabalho está organizado em mais quatro seções, além desta introdução. A Seção II apresenta tópicos importantes sobre o jogo Hanafuda, sua história, as cartas que o compõem e a forma de jogá-lo, além dos algoritmos utilizados. Na seção III é apresentado o *framework* Amê, suas funcionalidades, a arquitetura, seu modelo de classes e projeto de interface gráfica. A Seção IV apresenta a experimentação do *framework* Amê. E por fim, a Seção V aborda a conclusão do projeto e prenuncia os trabalhos futuros que poderão ser realizados.

II Jogos de Competição Baseados em Baralhos

Um jogo é uma atividade composta por um conjunto de ações limitadas a um conjunto de regras, direcionadas para um determinado fim (Schuytema e Manyen 2005). Os jogos de competição baseados em baralhos são jogos nos quais dois ou mais jogadores utilizam um conjunto de cartas (baralho) para competir entre si e determinar um vencedor.

A variedade de baralhos existentes e a possibilidade de usar um mesmo baralho de maneiras distintas gerou uma quantidade imensurável de jogos baseados em baralho, porém este trabalho trata do jogo no qual se utiliza o baralho Hanafuda. Neste capítulo na seção 2.1 descreve-se o jogo, na seção 2.2 são apresentados algoritmos de jogos de inteligência artificial e na seção 2.3 são apresentados os trabalhos relacionados.

2.1 Hanafuda

O Hanafuda é um baralho de origem japonesa usado em jogos voltados para competição. Este baralho assume diferentes nomes de acordo com o país em que se encontra, mas independente de sua localidade o significado é o mesmo: jogo das flores, fazendo referência às imagens de plantas presentes em todas as cartas. Neste projeto, assumiremos o nome japonês – Hanafuda – para um jogo específico no qual descreveremos as regras ao longo deste capítulo.

A história do Hanafuda começa com uma missão de Francisco Xavier, também conhecido como Apóstolo do Oriente, no Japão, a partir do ano de 1549 (Hanafuda 2013, 2014a). Sua missão não garantiu muitas conversões no país, que já tinha grande parte da população budista, o que tornou sua estadia curta. Apesar de breve, a sua passagem acabou por difundir hábitos tipicamente europeus, como o de jogar cartas com o baralho ocidental.

Pouco tempo depois, o Japão entrou no chamado Período Edo (1603 - 1868), período em que o governo fechou toda e qualquer relação com estrangeiros, mantendo contato apenas com a China e a Companhia Holandesa das Índias Orientais para fins estritamente comerciais e em localizações limitadas. Foi nesse mesmo período que apostas relacionadas aos jogos de cartas ocidentais foram proibidas (Hanafuda 2013, 2014a).

Em reação a esta medida, novos baralhos começaram a ser criados, apoiando-se no fato de que o jogo de cartas em si não estava proibido. Diversos baralhos foram criados e banidos desde então, até que foi criado um modelo de jogo que não possui números e tem

longa duração para conclusão, o que dificultava as apostas. Assim surgia o Hanafuda (Hanafuda 2013, 2014a).

A esta altura, os jogos de cartas já tinham sofrido uma queda significativa de popularidade, resultante de tanta repressão. Em 1889, já no período Meiji, Fusajiro Yamauchi fundou a Nintendo Koppai, voltada para confecção de cartas de Hanafuda artesanais (Hanafuda 2014b). O jogo foi incorporado aos salões de jogos de azar e logo ganhou não só popularidade, mas um espaço considerável na cultura do país.

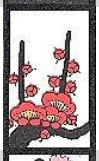
2.1.1 As Cartas

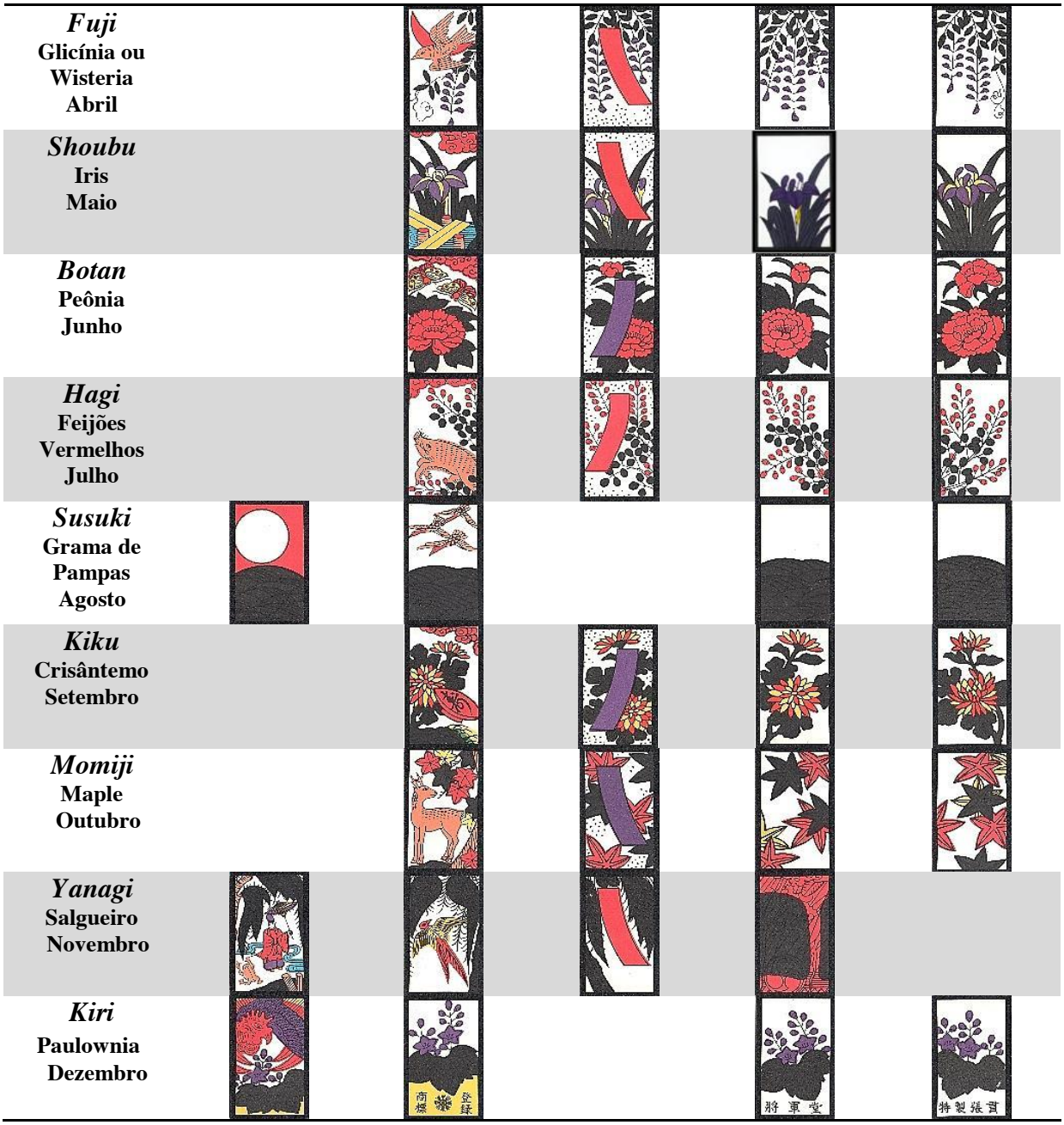
O baralho é composto por quarenta e oito cartas distribuídas em doze famílias, que representam cada mês do ano. Dentro de cada família é possível ainda classificar cada carta pelo seu valor. As classificações são:

- Kasu: são as cartas sem valor;
- Tan: são cartas que possuem uma faixa representada graficamente, e em condições comuns assumem o valor de dez pontos;
- Tane: são cartar com desenhos semelhantes ao Kasu da mesma família, mas com algum elemento gráfico adicional (na maioria dos casos, representação de animais). Estas cartas normalmente valem dez pontos;
- Kô: são as cartas especiais, as de maior valor, contabilizando cinquenta pontos cada uma individualmente.

Assim sendo, as cartas e suas classificações por família e por valor estão representadas abaixo.

Tabela 1 - Famílias das Cartas

Família	Kô	Tane	Tan	Kasu	Kasu
Matsu **Pinho Janeiro**					
Ume **Flor de Ameixa Fevereiro**					
Sakura **Flor de Cerejeira Março**					



Deve-se ressaltar uma família especial, a família *Yanagi* (ou Salgueiro). Nessa família, uma carta apresenta um homem com seu guarda-chuva e um sapo. Chamada de *Amê*, popularmente conhecida como bicho-papão, essa carta possui a propriedade de "comer" uma outra carta de pontos abaixo ou acima dela, ou seja, estando o *Amê* na mão do jogador, ele pode casá-la a qualquer carta de ponto que esteja na mesa. Quando o *Amê* estiver na mesa, o jogador pode pegá-lo apenas com uma carta da mesma família que esteja em sua mão ou que tire do monte.

2.1.2 Como Jogar

O jogo, representado na Figura 1, consiste em cada jogador tentar combinar uma de suas cartas com alguma carta disposta na mesa, sendo de mesma família ou satisfazendo o critério apresentado na sessão anterior, conquistando as cartas combinadas para si. Ao final do jogo, ganha o jogador com maior acúmulo de pontos. Diversas estratégias podem ser traçadas para alcançar o objetivo de ganhar o jogo.

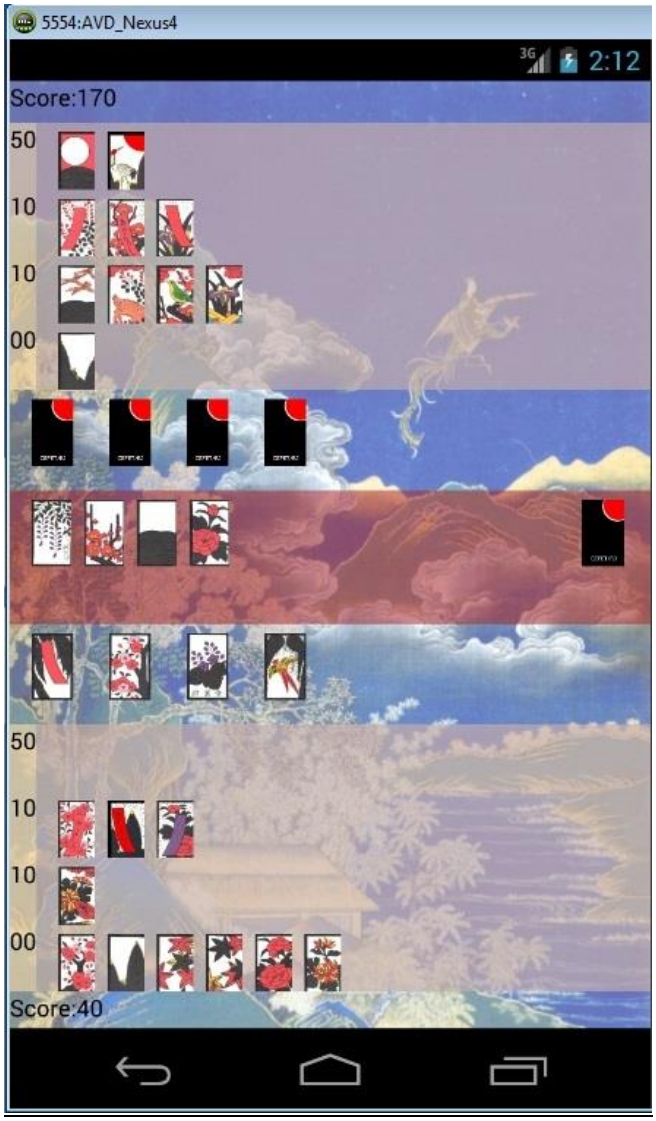


Figura 1 - Jogo Hanafuda

2.1.3 Regras Gerais do Hanafuda

O jogador principal, jogador 1, chamado de *oya*, é o que embaralha e dá as cartas. O monte deve permanecer empilhado e virado para baixo durante todo jogo. Ao final da partida, o vencedor torna-se o novo *oya*. Em caso de empate, o jogador principal não muda.

O jogo Hanafuda é baseado num princípio simples: cada jogador tenta combinar uma de suas cartas com qualquer outra carta presente na mesa, conquistando, assim, duas cartas para si. Essas combinações só podem ocorrer entre famílias, ou seja, Pinheiro combina com Pinheiro, Maple combina com Maple e assim por diante[1].

Cada carta apresenta um valor, por meio deste as cartas serão ordenadas. Primeiramente devem-se agrupar, em pilhas, as cartas de mesmo valor e, posteriormente, colocá-las em ordem decrescente (*kô*, ou luzes, cartas de maior valor; até *kasu*, ou lixo, cartas sem valor).

No final do jogo ocorre a contagem de pontos que, no Hanafuda, apresenta regras específicas. Ao tornar-se tão popular no Japão, diferentes formas de jogar Hanafuda surgiram, sendo alguma delas tradições de família. O que diferencia as versões são os yaku (combinações especiais): suas composições e valores.

2.1.4 O início

Para começar, cada jogador deve tirar uma carta do monte. Aquele que conseguir a carta do mês mais cedo se torna o *oya* da partida. O *oya* deve, primeiramente, embaralhar as cartas com a face para baixo. Em seguida, o jogador 2 deve "cortar" o monte. Depois desses passos, o *oya* distribui quatro cartas para seu oponente, quatro cartas para si e, por fim, quatro para a mesa, repetindo essa etapa duas vezes. Assim, cada jogador inicia o jogo, junto à mesa, com 8 cartas, sobrando ao monte, 24 cartas[2].

2.1.5 A progressão do jogo

O *oya* joga primeiro, tentando combinar cartas a fim de obter o máximo de pontos ou respeitar uma estratégia previamente planejada. Se ele puder combiná-la, simplesmente coloca sua carta sobre a desejada e pega o par para si. Se o jogador não possuir nenhuma carta para combinar, ele apenas descartará uma de suas cartas na mesa.

Em qualquer um dos casos mencionados, o jogador deve tirar e virar para cima a primeira carta do monte, tentando combinar com qualquer uma da mesa. Caso haja uma

[1] Foge a essa regra o *Amê*, em detrimento às regras já apresentas.

[2] Nenhum jogador deve receber três ou quatro cartas pertencentes a uma mesma família. Caso isso ocorra, um ou duas cartas desta família são aleatoriamente trocadas com o monte.

combinação, ela é feita e o jogador adquire o par; caso não, ele a coloca na mesa. Quando os jogadores obtiverem as cartas, devem colocá-las a sua frente, viradas para cima para facilitar a visualização do outro jogador.

Cabe salientar que, segundo as regras mencionadas, durante o seu turno, o jogador pode obter no mínimo nenhuma carta e no máximo, quatro cartas[3]. Se não houver mais cartas na mesa, o próximo jogador pode apenas descartá-las, sem obter nada. O processo descrito nesta sessão constitui uma jogada. As jogadas devem se repetir até o fim do jogo.

Entretanto, se houver três cartas da mesma família na mesa, as mesmas devem ser agrupadas e consideradas como uma só. Se o jogador conseguir uma carta para casar com elas, seja pela mão ou pelo monte, ele fica com as quatro de uma vez, fora aquelas que ele pode conseguir na conclusão desta rodada. Esta é a única exceção à regra das quatro cartas.

2.1.6 O Fim

Cada rodada é constituída de oito jogadas e, ao final de cada rodada, os pontos principais e os *yaku*, pontos provenientes de combos, são contabilizados e o processo descrito até então é repetido. A rodada termina quando o monte é deixado com oito cartas descobertas, independente de quantas ainda estão com a face desvirada na mesa, não obtidas. Portanto, cada jogador pode empatar um total de oito cartas do monte, antes do jogo acabar. Por fim, são contados os *kens* (pontuação bruta) de cada jogador, *i.e*., os pontos principais e os *yaku*, pontos provenientes de combos, detalhados na próxima seção. A pontuação líquida, contabilizada a partir da diferença entre a pontuação bruta dos jogadores, é contabilizada como pontuação da rodada para o jogador vencedor. Por exemplo, se o jogador A tiver feito 520 *kens* e o jogador B, 300 *kens*, então, a pontuação líquida de 220 *kens* é contabilizada para o jogador A, vencedor da rodada. Caso algum jogador tenha totalizado ao longo das rodadas 600 pontos, o *set* é finalizado. Nesse caso, o jogador ganha uma pontuação de *set*.

Ao final do *set*, a pontuação de set pode ser: um ponto de *set*, se o jogador tiver feito 600 pontos líquidos acumulados, dado que oponente possuía pontos líquidos acumulados; dois pontos de *set*, se o jogado tiver feito 600 pontos líquidos acumulados, dado que oponente não havia pontos líquidos acumulados; dois pontos de *set*, se o jogador tiver feito Jogo Forte (detalhado na próxima seção), dado que oponente possuía pontos líquidos acumulados; quatro pontos de *set*, se o jogador tiver feito 600 pontos via Jogo Forte, dado que oponente não possuía pontos líquidos acumulados.

[3] Existe um caso no qual poder-se-ia ter mais de quatro cartas, dependendo da configuração inicial da mesa, mas não será tratado no contexto deste trabalho.

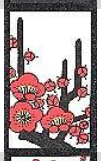
2.1.7 Pontuação das Cartas e Combos

No *Mayumi-no* Hanafuda, a contagem de *ken*, pontos de jogo, recebida por determinado jogador se dá pela fórmula: "*ken* do jogador que mais pontuou menos *ken* do jogador que menos pontuou" e isto é igual ao total de pontos do jogador vencedor.

> OBS: Quando um jogador consegue todas as quatro cartas de uma família, uma das cartas *kasu* marca um bônus de 50 *ken*.

2.1.8 Pontuação de cartas separadas

Tabela 2 - Pontuação das Cartas

Família	50 Ken	10 Ken	10 Ken	Sem pontos	Sem pontos
Matsu **Pinho Janeiro**					
Ume **Flor de Ameixa Fevereiro**					
Sakura **Flor de Cerejeira Março**					
Fuji **Glicínia ou Wisteria Abril**					
Shoubu **Iris Maio**					
Botan **Peônia Junho**					
Hagi **Feijões Vermelhos Julho**					
Susuki **Grama de Pampas Agosto**					
Kiku **Crisântemo Setembro**					

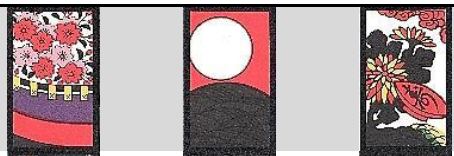
Momiji **Maple Outubro**					
Yanagi **Salgueiro Novembro**					
Kiri **Paulownia Dezembro**					

2.1.9 Yaku (combos especiais)

Durante o jogo, algumas combinações específicas valem mais pontos do que a soma da pontuação individual de cada carta. A estas combinações chamamos combos especiais, as quais podem ser vistas na Tabela 3. Uma estratégia adotada por alguns jogadores é obter essas cartas a fim de compor um combo para ter pontuação maior do que a do adversário.

Tabela 3 - Pontuação das Combos de três cartas

Nome	Pontuação	Cartas		
Matsu-Kiri-Bôzu	150 ken			
Umé + Matsu + Sakura	150 ken			
Aotan (Faixa azul)	100 ken			
Acatã (Faixa vermelha)	100 ken			
Cosan (Faixas escritas)	150 ken			
Inô-Shika-Chô	300 ken			

Tepô	300 ken	
Nizoro	200 ken	
Hanami I-Pao	100 ken	
Tsukimi I-Pai	100 ken	

2.1.10 Jogos Fortes

Algumas combinações de cartas garantem a vitória incondicional da rodada. Assim sendo, considerando que o jogador obteve todas as cartas que compõem alguma destas combinações, mesmo que ele tenha pontuação inferior a dos adversários, ainda assim será o vencedor da rodada. Chamamos estas possíveis combinações de Jogos Fortes. Os jogos fortes estão especificados na Tabela 4.

Tabela 4 - Combinações que formam Jogos Fortes

Nome	**Cartas**
Shikô	
Nanatã	

O *Shikô* é composto por quatro cartas, sendo elas: *Matsu*, *Quiri*, *Boozu e Sakura*, exibidas na tabela nesta mesma ordem. O *Nanatã* é composto pela obtenção de sete cartas com faixas.

2.2 Algoritmos de Inteligência Artificial

Diversos problemas relacionados ao apoio à decisão recaem na exploração de um espaço de busca que tendem a ser não polinomiais. Neste contexto, a exploração deste espaço é computacionalmente não tratável. Para tornar o problema tratável, a busca da solução ótima é amenizada e substituída por uma solução otimizada. Nesta conjuntura, faz-se valer de mecanismos como heurísticas para direcionar a procura de uma solução no espaço de busca. Ademais, quando o espaço de busca pode ser caracterizado em cenário de competição, costuma-se fazer valer de diversos algoritmos.

2.2.1 Algoritmo Aleatório

O algoritmo Aleatório (Russell e Norvig 2009) não possui estratégia previamente definida para solução de problemas, baseando-se em tentativas e erros para alcançar o objetivo. A solução obtida através deste algoritmo é, portanto, ineficiente e ineficaz. Muitas tentativas podem não ser bem sucedidas e mesmo as assertivas são desconsideradas nas jogadas seguintes.

O algoritmo Aleatório tem três componentes: (i) um conjunto de candidatos, a partir do qual a solução é criada; (ii) a função de seleção, que escolhe aleatoriamente o candidato a ser adicionado à solução; (iii) função de solução, que indica quando uma solução é selecionada. Neste algoritmo se uma escolha for feita, mesmo que não represente uma combinação válida, esta não é revisada. Portanto, algumas jogadas podem ser errôneas. Mesmo que existam soluções que proporcionem uma jogada válida, a jogada realizada será a definida a partir da função de seleção.

2.2.2 Algoritmo Guloso (Greedy)

O algoritmo Guloso é um algoritmo que adota a heurística de fazer a melhor escolha aparente no momento, isto é, fazer uma ótima escolha local em cada fase na esperança de encontrar uma solução global ótima (Cormen et al. 2009). Em muitos problemas, uma estratégia gulosa não produz, em geral, uma solução ideal, mas ainda assim uma heurística gulosa pode produzir soluções localmente ótimas que se aproximam de uma solução ótima global em um tempo razoável.

Geralmente é utilizado para resolver problemas de otimização. Constitui-se, um problema de otimização encontrar, a partir de um conjunto S, um subconjunto E de S que possua o menor (ou maior) custo que satisfazem uma certa propriedade. A primeira etapa para a solução de um problema de otimização é caracterizar a estrutura de uma solução ótima. No

algoritmo guloso se trabalha com subestrutura ótima, na qual cada subconjunto de S é satisfeito da melhor forma possível para que o conjunto S possa ser resolvido. Esta subestrutura ótima existente se assemelha a programação dinâmica, tornando-se uma característica comum nos problemas onde se aplica este algoritmo (Cormen et al. 2009).

Em geral, o algoritmo Guloso tem cinco componentes: (i) um conjunto de candidatos, a partir do qual a solução é criada; (ii) a função de seleção, que escolhe o melhor candidato a ser adicionado à solução; (iii) uma função de viabilidade, que é usado para determinar se um candidato pode ser utilizado para contribuir para uma solução; (iv) uma função objetivo, que atribui um valor a uma solução, ou uma solução parcial; (v) e uma função de solução, que indica quando uma solução completa é descoberta.

Neste algoritmo se uma escolha for feita esta nunca mais é revisada, ou seja, não há qualquer tipo de reavaliação, pois confia ter escolhido bem os componentes anteriores. Portanto, nem sempre funciona. Há problemas para os quais, escolher um passo aparentemente menos adequado, resulta eventualmente em uma melhor solução. Logo, a maioria dos problemas em que trabalha, tem duas propriedades: (i)propriedade escolha *Greedy*: uma solução ótima global pode ser obtida através de uma sequência de escolhas ótimas feitas localmente; (ii) subestrutura ótima: uma solução ótima para um problema contém soluções ótimas para subproblemas.

2.2.3 Algoritmo Minmax

Um algoritmo Minmax é um algoritmo recursivo para escolher o próximo passo de um n-jogador do jogo, normalmente de dois jogares (Russell e Norvig 2009). Um valor é associado a cada posição ou estado do jogo. Este valor sendo calculado por uma função de avaliação de posição que indica o quão bom seria o jogador chegar nessa posição. O jogador faz o movimento que maximiza o valor mínimo da posição resultante dos movimentos seguintes possíveis do oponente. Se for a vez de A mover, A calcula um valor para cada um de seus movimentos legais.

O algoritmo Minimax basea-se na heurística de maximizar a ação do jogador e minimizar a ação do adversário. Baseando-se na suposição de que o adversário escolhe sempre o movimento ideal, e nunca comete um erro, gerando toda uma árvore de busca dentro do limite permitido. Sendo ele completo apenas no caso de a árvore ser finita.

Uma versão simples do algoritmo Minimax lida com jogos como o jogo da velha, no qual cada jogador pode ganhar perder ou empatar. Se o jogador A pode vencer com um movimento, ele o executa por ser o melhor que ele pode escolher. Se o jogador B identifica

que pode realizar dois movimentos, um leva a vitória ao adversário e outro ao empate, então a melhor jogada do jogador B é o empate. Após algumas rodadas é fácil definir qual é o melhor movimento. O algoritmo Minimax ajuda a encontrar a melhor jogada ao caminhar pelas opções válidas a partir do fim do jogo. A cada passo assume-se que o jogador A está tentando maximizar as chances de ele ganhar, enquanto na próxima rodada o jogador B está tentando minimizar as chances de isso acontecer (ao maximizar as chances de que ele próprio ganhe).

O algoritmo consiste de cinco passos: (i) gerar toda a árvore do jogo (ou a parte que será analisada), até os estados terminais; (ii) aplicar a função de avaliação aos estados finais, obtendo o valor de cada nodo; (iii) usar o valor dos estados finais para determinar o valor dos nodos um nível acima na árvore de busca; (iv) continuar passando os valores das folhas em direção à raiz, um nível por vez; (v) quando o valor final atingir o topo da árvore, o jogador pode escolher o movimento que lhe traga maior benefício. O algoritmo é repetido nas próximas jogadas até que um dos jogadores vença ou o jogo acabe em empate.

2.3 Trabalhos relacionados

O estudo de diversos projetos foi executado visando entender os diferentes problemas e soluções propostos que, de alguma forma, se relacionam ao trabalho de conclusão de curso em questão. Por meio desses estudos, foi possível entender o que já existe em termos de tecnologia e o que falta, buscando constituir, deste modo, o diferencial deste projeto. Dentre estes trabalhos, podem-se destacar trabalhos baseados no jogo de *poker*, os *frameworks* Athus (Segundo 2011) e Guff (Valente 2005), alguns trabalhos teóricos e alguns protótipos que combinam teoria a prática.

O jogo de *poker* é o jogo de competição utilizado como base para muitos trabalhos. Dos trabalhos os quais o utilizam temos um que faz uso a variação *No-Limit Texas Hold'em* do *poker* para investigar a dinâmica evolutiva do comportamento estratégico neste jogo (Ponsen et al. 2009), estudando como jogadores racionais, em um *site* de *poker online*, alternaram entre diferentes estratégias em circunstâncias diferentes. Outros o empregam na pesquisa de IA, no intuito de desenvolver algoritmos que sejam oponentes precisos e dinâmicos de *poker* (Rubin e Watson 2011), ou criando programas como o *Poki* (Billings et al. 2002) que reproduz razoavelmente o jogo de *poker* na variação *Texas Hold'em*.

O *framework* Athus possibilita o desenvolvimento de jogos para TV Digital utilizando orientação a objeto com codificação feita no IDE Eclipse (Segundo 2011). As aplicações são desenvolvidas com o uso do *middleware* chamado Ginga e seu subsistema lógico, Ginga-

NCL. Apesar do Athus se assemelhar a este projeto, o seu foco está direcionado a interação direta com a TV Digital, sem se preocupar com algum tipo de algoritmo específico. Além do intuito de disponibilizar diversas funcionalidades e suporte as características de portabilidade e interatividade do Ginga.

O *framework* Guff (Valente 2005) é baseado em uma arquitetura reutilizável, que possibilita diferentes módulos computacionais comumente presentes nos jogos para computador a partir do uso de bibliotecas, *toolkits* e *engines*. O Guff fora desenvolvido para aplicação de alguns dos conceitos sobre desenvolvimento de jogos, com duas camadas (uma de aplicação e outra de *toolkit*) e escrito em C++ com auxílio de diversas bibliotecas.

No que diz respeito a trabalhos em IA, existe uma abordagem em relação aos problemas de motivação usando robótica como cenário (Hawes 2011). Sugerindo uma estrutura de gestão de motivação para um sistema inteligente que nos orienta a olhar para os problemas das unidades de codificação (como as necessidades do sistema são representados), a geração de objetivos (como determinados casos de objetivos, ou rotas possíveis, são gerados para satisfazer uma necessidade a partir das unidades com referência ao estado atual), e a seleção de objetivos (como o sistema determina quais dos objetivos gerados podem posteriormente influenciar o comportamento do sistema), focando comportamento alvo-dirigido (característica determinante do comportamento autônomo inteligente).

Dentre as pesquisas desenvolvidas em jogos eletrônicos há aquelas as quais apresentam que o uso de IA nesta área é diferente do de IA comumente adotada no meio acadêmico (Gold 2005, Millington e Funge 2009). A IA abordada em jogos pode requer a criação de algoritmos com comportamentos específicos em determinados contextos. A esta forma de algoritmo é dado o nome Game IA (Millington e Funge 2009, Yannakakis 2012). Outro algoritmo bastante difundido para criação de jogos com IA é baseado no Minmax, no qual são apresentados diversas formas de implementação deste algoritmo no contexto de jogos (Santana 2006).

Alguns trabalhos (Silva e Adamatti 2012) propõem a utilização de técnicas de IA em jogos de estratégia de forma a tornar tanto experiência quanto aprendizado mais interessante e divertido, ao mesmo tempo em que torna a IA adaptativa, o que faz aumentar a dificuldade do jogo. Essa adaptabilidade é um fator diferencial na criação de jogos, tanto para os consumidores das franquias de jogos, quanto na parte de seu desenvolvimento, na qual há o estimulo para o desenvolvimento da arte e da tecnologia de IA. Das técnicas de adaptabilidade presentes em jogos se faz uso de técnicas como: máquinas de estado finito, *scripts*, algoritmos genéticos e redes neurais (Heule e Rothkrantz 2007, Robertson e Howells 2008, Rubin e

Watson 2011, Smith 2007). Comumente, a maiorias destes jogos é desenvolvido na linguagem de programação C++ variando-se a biblioteca gráfica e o ambiente de desenvolvimento. Neste contexto, o jogo presente no artigo de Silva e Adamatti (Silva e Adamatti 2012), por exemplo, é desenvolvido em C++, com biblioteca gráfica ALLEGRO, nos ambientes de desenvolvimento DEV C++ e CodeBlocks.

O trabalho que foi desenvolvido se assemelha aos projetos citados em diferentes aspectos. Sendo um deles a criação de um *framework* para jogos, mas com um foco em jogo de competição utilizando a linguagem de programação Java da plataforma Android, que não só oferece subsídios para o desenvolvimento, mas também para execução do projeto.

O jogo base deste trabalho diferente do jogo de *poker*, não faz uso de apostas. As apostas que ocorrem no *poker*, bem como a análise de comportamento dos oponentes interferem diretamente no desenvolvimento do jogo. Baseando-se nas apostas feitas, é possível supor o valor da mão do oponente. Com a análise do comportamento dos jogadores, é possível formar uma opinião quanto à estratégia do oponente, por exemplo, se este está blefando ou não. Desta forma, a única forma de um jogador pressupor as cartas presentes na mão do oponente é através de uma análise probabilística tendo como base somente as cartas presentes na mesa e na própria mão.

Assim como nos trabalhos mencionados, se realiza uma abordagem no que diz respeito ao conceito da Inteligência Artificial e algoritmos utilizados nos jogos. Dos algoritmos existentes, os algoritmos Aleatório e Guloso são o foco. O jogo Hanafuda possibilita a minimização dos problemas de motivação IA, devido ao uso das regras do jogo na geração da estrutura de gestão de motivação. Entretanto, neste trabalho não se entra no mérito no que diz respeito à história e evolução de IA e jogos.

IIIFramework Amê

O *framework* Amê foi desenvolvido visando criar um ambiente competitivo e possibilitar a implementação de diversos algoritmos, com diferentes heurísticas, para que estes possam ser então, avaliados. Neste capítulo são apresentados os aspectos referentes à criação do *framework* organizados pelos tópicos de Android (seção 3.1), plataforma utilizada, arquitetura (seção 3.2), apresentado a organização da arquitetura do Amê, e projeto de interface gráfica (seção 3.3).

O jogo foi desenvolvido fundamentado em duas funcionalidades, condizendo com a estrutura do Hanafuda. A primeira funcionalidade ocorre quando o jogo se inicia. Neste momento, o *framework* instancia e prepara todas as classes necessárias para seu funcionamento. Em seguida, em turnos, algoritmo e jogador realizariam jogadas condizentes com as regras do Hanafuda até o termino da partida.

3.1 Android

Projetado principalmente para dispositivos móveis *touchscreen*, como *smartsphones* e *tablets*, o sistema operacional Android é um sistema Linux multiusuário, em que cada aplicativo é um usuário diferente. Logo, o sistema atribui a cada aplicativo uma identificação de usuário exclusivo Linux e cada processo tem a sua própria máquina virtual, de modo que o código de um aplicativo é executado isoladamente de outros aplicativos (Android Developers 2013a). A Figura 2 mostra uma visualização das camadas do *software* Android.

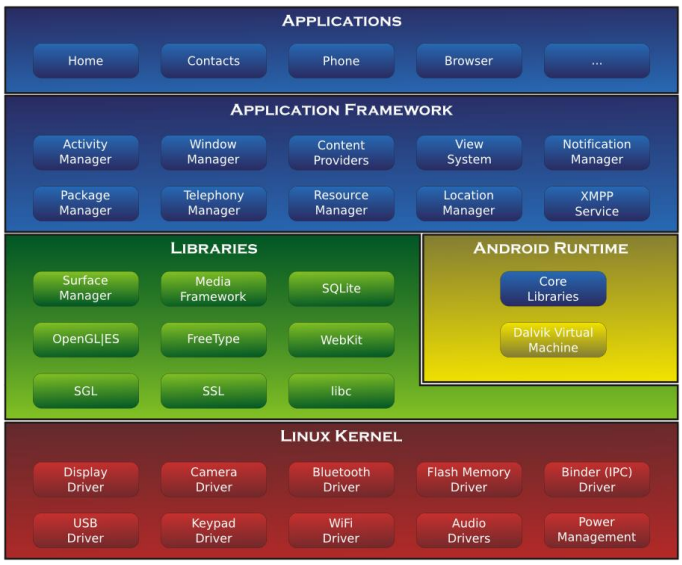


Figura 2 - Camadas do Software Android (Android Developers 2013a)

As aplicações são desenvolvidas na linguagem Java usando o Android SDK (Kit de Desenvolvimento de Software). O SDK inclui um conjunto abrangente de ferramentas de desenvolvimento, incluindo um depurador, bibliotecas de software, um aparelho emulador baseado no Quick EMUlator (QEMU) (QEMU 2013), documentação, código de amostra e tutoriais. O suporte oficial de IDE é o Eclipse (Eclipse 2013)usando o *Android Development Tools* (ADT) *plugin* (Android Developers 2013b).

Toda aplicação desenvolvida em Android possui um arquivo chamado *AndroidManifest.xml*. O *AndroidManifest.xml* contém as informações de configuração necessárias para você instalá-lo corretamente em um dispositivo. Ele inclui os nomes de classes necessárias, os tipos de eventos que o aplicativo está pronto para processar, as permissões necessárias que o aplicativo precisa para execução, declara o nível mínimo exigido pela *Application Programming Interface* (API) do aplicativo, com base em qual das APIs o aplicativo usa, além de declarar os recursos de hardware, software e biblioteca API usados ou exigidos pelo aplicativo.

Os aplicativos Android consistem em quatro tipos de componentes: (i) atividades, (ii) serviços, (iii) provedores de conteúdo e (iv) receptores de transmissão. Uma atividade corresponde à implementação de um aplicativo que possui uma interface visível que se inicia ao selecionar o seu ícone. Um serviço é utilizado para qualquer aplicativo que precise persistir por um longo período de tempo, como um monitor de rede ou um aplicativo de verificação de atualização. Os provedores de conteúdo gerenciam o acesso aos dados que persistem, como um banco de dados SQLite, utilizado em aplicativos complexos ou que disponibilizam dados para várias atividades ou aplicativos. Finalmente, um receptor de transmissão é um aplicativo que pode ser ativado para processar um elemento de dados ou para responder a um evento, como o recebimento de uma mensagem de texto (Android Developers 2013a).

Dos quatro componentes apresentados, optamos por utilizar a atividade, pois possui uma interface para interação com o usuário, criando aplicações com uma ou múltiplas telas, cada uma com sua respectiva atividade. Cada atividade possui um ciclo de vida, Figura 3, e cada uma permite que outras atividades sejam chamadas, criando uma pilha de atividades ou destruindo a atividade atual e criando uma nova a partir de dados recebidos da sua atividade antecessora.

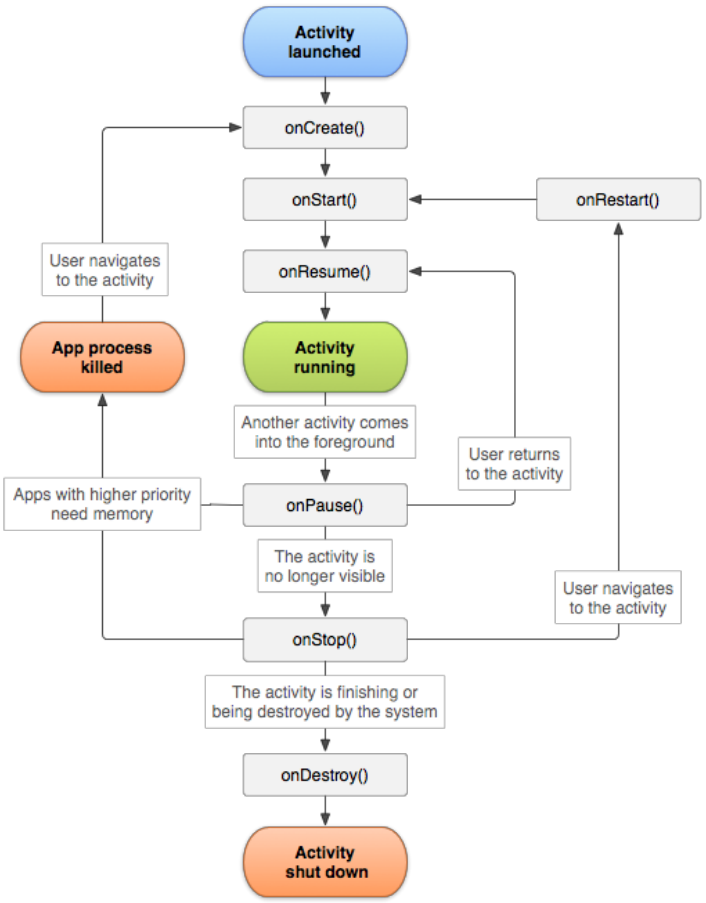


Figura 3 - Ciclo de vida de uma Atividade (Android Developers 2013c)

A *Graphic User Interface* (GUI) para uma atividade é prestada por uma hierarquia de visualizações - objetos derivados das classes *View* e *ViewGroup*. Cada *View* controla um espaço retangular especial dentro da janela da atividade e pode responder à interação do usuário. A classe *View* possui o subgrupo denominado "*widgets*", objetos que fornecem elementos visuais (e interativos) para a tela, como um botão, campo de texto, caixa de seleção, ou apenas uma imagem, enquanto que a classe *ViewGroup* possui os "*layouts*", as subclasses que fornecem um modelo de *layout* exclusivo para a organização dos elementos na tela.

A maneira mais comum para definir um *layout* usando *views* é com um arquivo de *layout* XML salvo em seus recursos do aplicativo. Desta forma, pode-se manter o *design* da interface de usuário separadamente do código-fonte que define o comportamento da atividade.

3.2 Arquitetura

O *framework* Amê implementa uma arquitetura semelhante a utilizada em *softwares* de jogos (Blizzard 2014) (Electronic Arts 2014), pois se baseia no jogo Hanafuda. Logo a arquitetura seria divida em: *Engine* e *Game*. *Engine* representando toda a infraestrutura do jogo (parte gráfica, som, IA, a física do jogo, por exemplo) e Game que implementa o código do "*gameplay*" que realiza tarefas básicas como contar pontos e dizer ao motor qual arquivo carregar (Millington e Funge 2009, Pressman 2006, Unity 2013). Contundo optou-se por adaptar esta arquitetura na visão MVC (*Model – View – Controller*) (Pressman 2006) para facilitar o entendimento, a concepção e a representação da mesma neste trabalho. Assim, o *Enginer* é representado nas camadas *View* e *Model*; e *Game* na camada *Controller*.

A Figura 4, representa o Diagrama de Classes Conceitual do sistema seguindo a forma definida pela linguagem de modelagem unificada (UML) (Fowler 2004) utilizando a ferramenta Astah (Astah 2013) para construí-lo. As classes de cor amarela pertencem à camada *Model*, as de cor verde a *View* e a de cor cinza a *Controller*.

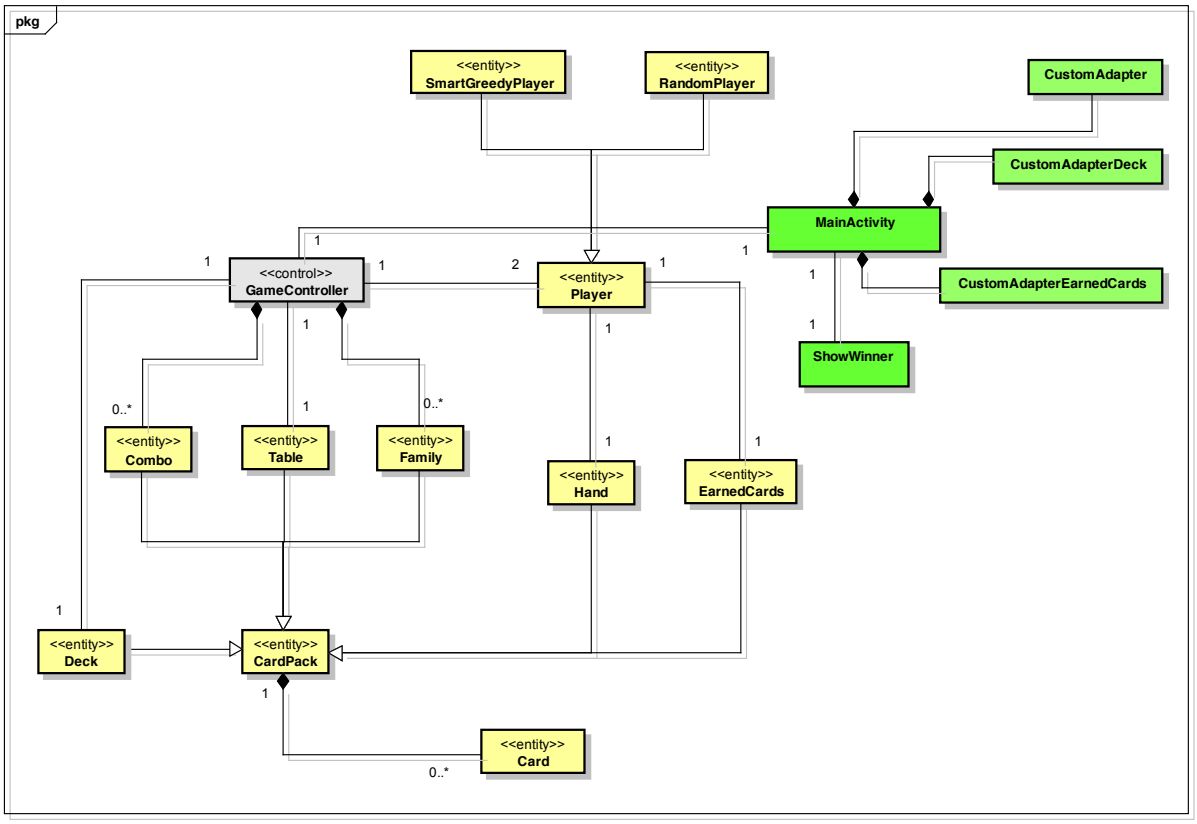


Figura 4 - Diagrama de Classes Conceitual

3.2.1 Estruturação da Camada *Model*

A camada *Model*, Figura 5, também chamada de negócios, contém toda a lógica do sistema. No Amê esta camada é composta por todas as classes relacionadas com o funcionamento do núcleo do jogo. Logo, a camada é composta pelas classes que foram denominadas como *Entity* no diagrama de classe apresentado anteriormente.

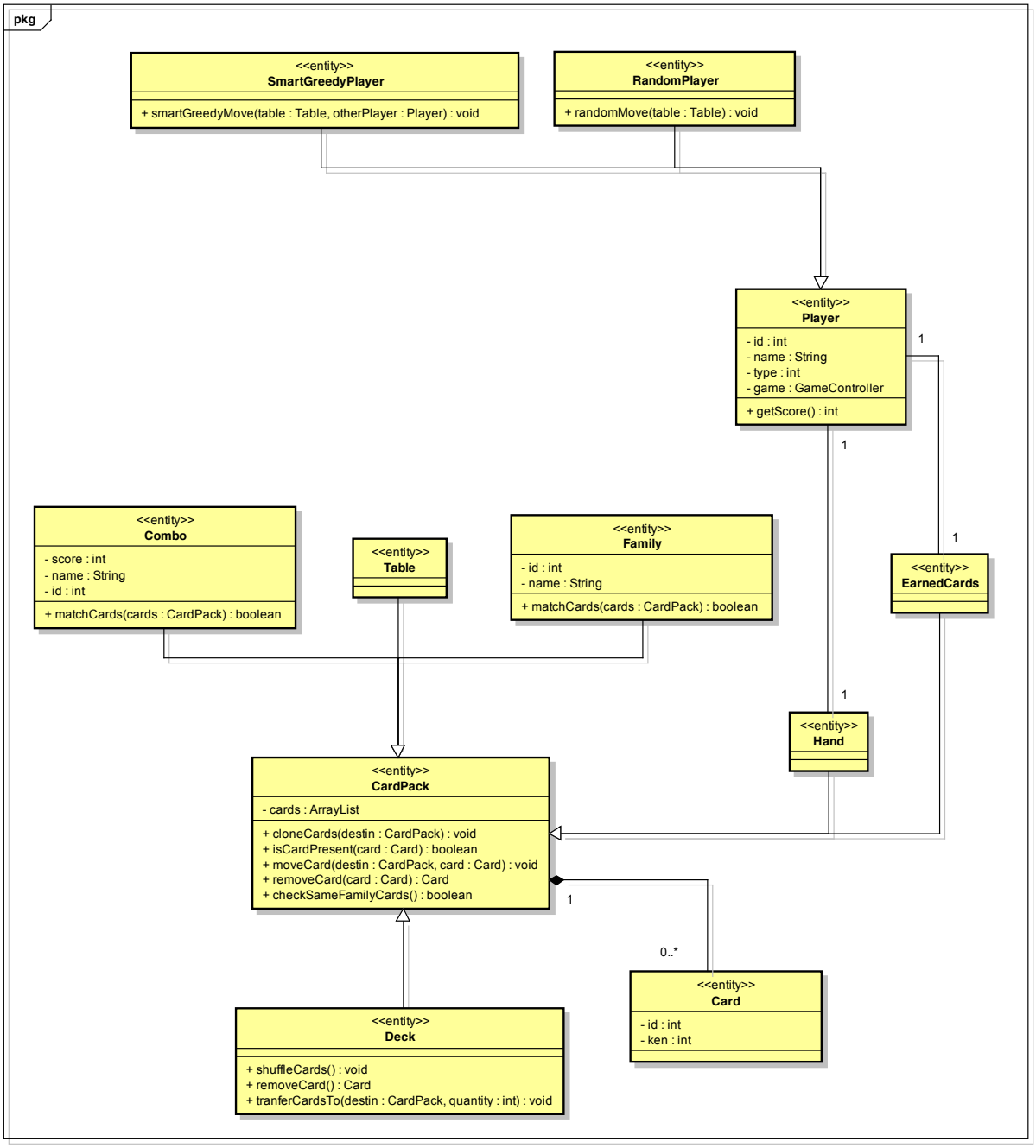


Figura 5 - Camada *Model*

A classe *Card* contém a informação de cada carta do baralho, como a família a que pertence e sua pontuação, sendo utilizada para compor a classe *CardPack*, uma das principais do sistema.

A classe *CardPack* além de ser a fábrica principal de cartas (baralho), também representa, de modo polimórfico, qualquer grupo de cartas reunidas. Por exemplo: cartas da mão, mesa ou monte, são um *CardPack*. A função *cloneCards* copia todo um conjunto de cartas para um grupo de cartas, diferente da *moveCard* que move apenas uma carta para um determinado grupo. A função *removeCard*, como o próprio nome sugere, remove uma carta do grupo, a *isCardPresent*, verifica se uma determinada carta está contida em um grupo de cartas e a *checkSameFamilyCards* verifica o limite de cartas de uma família. Esta funções são usadas no *GameController*.

A classe *Hand* são as cartas pertencentes à mão de um jogador. *EarnedCards* correspondem as cartas adquiridas pelo jogador à cada partida. *Table* correspondem às cartas pertencentes à mesa do jogo, enquanto que *Deck* representa o monte do jogo, que embaralha e distribui as cartas para a mesa e mão dos jogadores. Todas estas classes citadas são um *CardPack* e serão usadas pelo *MainActivity* que será abordado mais a frente.

A classe *Combo* concebe o conjunto de cartas que formam uma combinação específica, possuindo a informação como o valor da sua pontuação e o respectivo nome. A classe *Family* é o conjunto de cartas pertencentes a uma mesma família, além de deter informação sobre o nome da respectiva família. Ambas as classes mencionadas possuem uma função chamada *matchCards* que ajuda a compor a pontuação dos jogadores, as funções se diferenciam no fato de que a da classe *Combo* leva em consideração o número mínimo de cartas necessárias para compor um determinado combo e da classe *Family* o número de cartas que uma família possui.

A classe *Player* é a abstração do jogador, podendo ser tanto uma generalização de um jogador humano quanto da própria máquina, com a função *getScore* para se obter a pontuação do jogador. É a partir desta classe que o *framework* deve ser expandido para a implementação de novos algoritmos. Inicialmente, o *framework* conta com duas implementações iniciais. A máquina controla as classes: *RandomPlayer*, o jogador "aleatório", com uma IA mais rudimentar, em que joga uma carta aleatoriamente selecionada da mão e da mesa; e a *SmartGreedyPlayer*, o jogador "guloso", com uma IA um pouco mais sofisticada que a anterior, em que joga a carta que lhe dará maior pontuação a cada jogada.

3.2.2 Estruturação da Camada *View*

Na camada *View*, Figura 6, também conhecida como camada de apresentação, encontram-se as classes de interfaces. Em se tratando do jogo computacional, as classes que controlam os sons e os textos programados também fazem parte dessa camada. Neste trabalho a *View* é desenvolvida em *Android*, possuindo classes Java que utilizam arquivos XML para gerar as telas do *framework*.

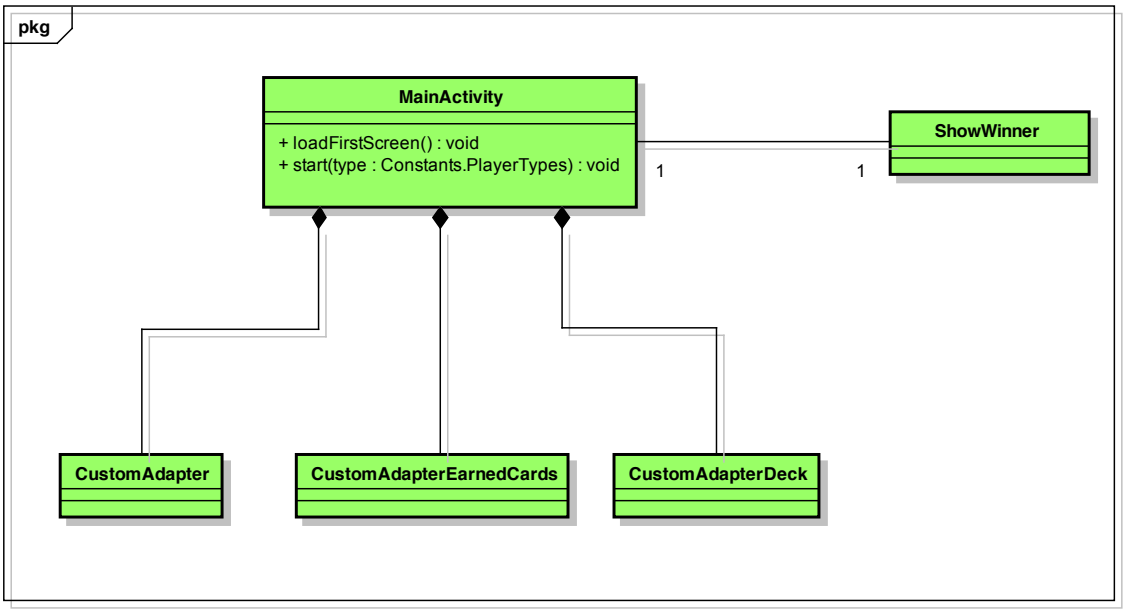


Figura 6 - Camada *View*

A principal classe da camada *View* é a *MainActivity*. Nela se encontra todo gerenciamento do *layout* e interação do jogo, além de ser responsável pelas exibições das telas inicial e jogo do aplicativo Android. Sendo uma classe que estende da classe *Activity* do Java Android, possui uma *View* e um arquivo XML para definir cada *layout* de tela.

A classe *MainActivity* possui uma função *loadFirstScreen* que gera a tela inicial com um botão e a figura do oponente, após o botão ser clicado a função *start* se encarrega de gerar toda a tela de jogo distribuindo todos os elementos de acordo com as instruções que recebe do *GameController*.

As classes *CustomAdapter*, *CustomAdapterDeck*, *CustomAdapterEarnedCards* são classes que estendem a classe *ArrayAdapter<>* do Java Android. Elas geram *Views* personalizadas para compor a classe *MainActivity*. A *CustomAdapter* gera todas as imagens das cartas da mão do jogador, da mesa e da carta comprada do *deck*; a *CustomAdapterDeck* as imagens das cartas do oponente e *deck*; e a *CustomAdapterEarnedCards* é responsável pelas imagens das cartas ganhas de ambos os jogadores.

A classe *ShowWinner*, como a *MainActivity*, descende da classe *Activity* e possui uma *View* e um arquivo XML. Ela é responsável por criar a tela de pontuação e todas as informações geradas na tela são obtidas a partir da *MainActivity*.

3.2.3 Estruturação da Camada *Controller*

A camada *Controller*, Figura 7, ou controladora é uma camada que realiza a comunicação entre a *View* e a *Model*, que propaga as mensagens da *View* para a *Model*. No Amê, esta camada é representada principalmente pela classe *GameController*, que tem a função de controlar todo o mecanismo do jogo.

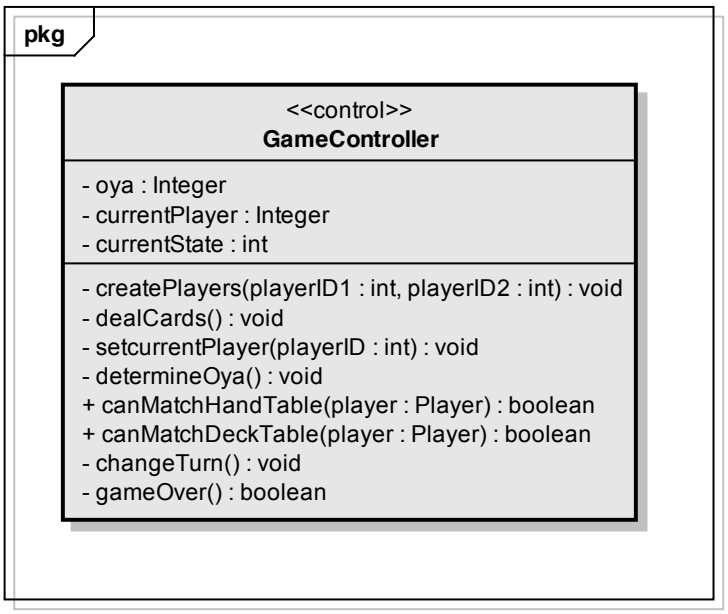


Figura 7 - Camada *Controller*

A classe *GameController* é a principal classe relacionada à lógica do jogo. Realiza a intermediação e interligação com praticamente todas as demais classes. É ela quem prepara o baralho, distribui as cartas do monte para os jogadores, verifica compatibilidade das cartas, controla o término de jogo, dentre outras atribuições.

O *GameController* possui a função *createPlayers* para criar os jogadores e assim poder iniciar o jogo. A distribuição das cartas entre os jogadores é feita pela *dealCards* e a função *gameOver* controla o fim do jogo. A verificação de compatibilidade é realizada pelas funções *canMatchHandTable* e *canMatchDeckTable*, a primeira verifica a compatibilidade entre as cartas da mão e da mesa selecionadas pelo jogador, e a segunda entre a carta comprada do *deck* e a carta da mesa selecionada.

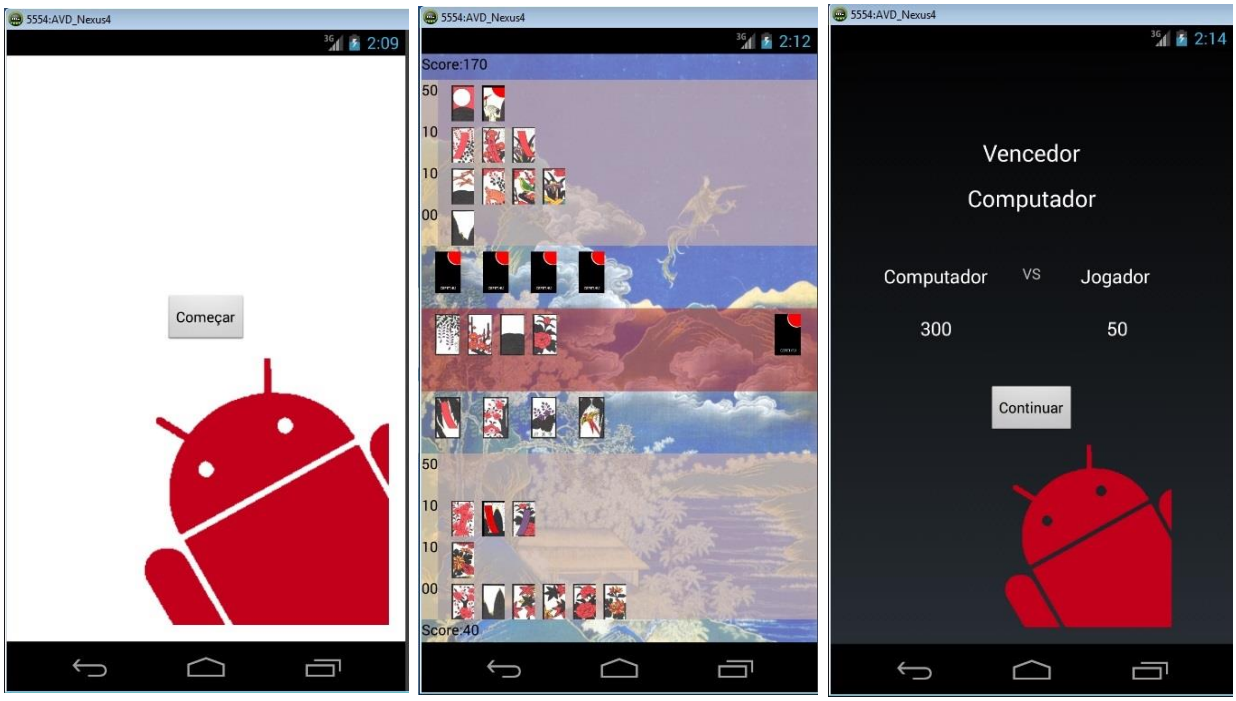
Os atributos da classe *GameController* auxiliam a controlar o fluxo de jogo. O atributo *oya*, determinado pela função *determineOya*, corresponde ao jogador que iniciará a partida. O *currentPlayer* indica a qual jogador o turno atual pertence, cabendo ao *changeTurn* efetuar a

troca de turnos entre os jogadores e ao *setCurrentPlayer* fixar o jogador corrente. Por sua vez, o atributo *currentState* é usado como maneira de determina em que fase do turno o jogador está, podendo ser a fase de compra de carta do *deck*, a de realizar um jogada ou de descartar uma carta.

3.3 Projeto de Interface Gráfica

O projeto de interface gráfica aborda os modelos que representam o *design* em potencial do sistema e seus componentes. A interface gráfica do sistema foi desenvolvida visando facilitar o uso do *framework* Amê e a avaliação das IAs durante a execução dos processos que compõem o jogo. O *framework* Amê, possui três telas que interagem entre si de forma encadeada.

A Figura 8.a apresenta a tela inicial do sistema. É a partir dessa interface que o usuário vai começar a interagir com o sistema. Nesta tela apresenta-se a cor representativa do oponente e o botão que iniciará o jogo.



(a) (b) (c)

Figura 8 – Projeto de Interface – (a) tela inicial, (b) tela do jogo; (c) tela de pontuação

A execução do jogo do Amê acontece todo em uma única tela, Figura 8.b. A parte superior da faixa vermelha representa o computador (IA): suas cartas da mão, cartas ganhas ao longo da partida as quais foram organizadas pelo seu valor individual dentro do jogo, e sua pontuação. Na faixa vermelha encontram-se as cartas da mesa e o *deck*, à direita. E por fim, os

elementos abaixo da faixa vermelha, representam o jogador: cartas da mão, cartas ganhas e pontuação.

Durante a execução do *framework* são feitas interações com o usuário por meio de uma caixa de texto e mensagens de curta duração na tela. As mensagens tem a finalidade de auxiliar no uso do *framework*, como por exemplo, indicar a passagem de turno. Após o término da partida é apresenta a tela de pontuação, Figura 8.c., com a indicação do vencedor, a pontuação de ambos na partida, a cor do oponente enfrentado e o botão de continuar. Este botão ao acionado retorna a tela inicial para começar uma nova partida com um novo oponente.

IV Experimentação

Após a fase de implementação, foram realizados testes que visavam avaliar o *framework* Amê. Em particular o objetivo era identificar se, na percepção do usuário, as diferentes implementações de algoritmos de inteligência artificial eram perceptíveis para os usuários com diferentes níveis de dificuldade.

Neste diapasão foi desenvolvido um roteiro de experimentação e um formulário de avaliação contendo questões para avaliar o aplicativo. Para conceituação do formulário, foi adotado o método GQM (KOCHANSKI 2011). Já o roteiro continha os passos a serem executados durante o processo de avaliação. Este roteiro foi entregue aos usuários piloto logo que o mesmo chegava ao local do experimento. O experimento foi conduzido no Laboratório de Avaliação de *Software* da Escola de Informática & Computação do CEFET/RJ.

Os usuários pilotos eram constituídos por voluntários, alunos da Escola de Informática e Computação. Ao todo, 27 voluntários participaram da experimentação. Suas respostas foram base de estudo para avaliação de hipóteses. Tanto para a instanciação dos formulários, quanto para a avaliação do *software* foi utilizado o FastEval (Santos et al. 2014).

4.1 GQM

O Método GQM (*Goal – Question - Metric*) é uma abordagem para medição de sistemas. Para avaliar um sistema é necessário inicialmente definir objetivos (*Goals*). A experimentação é conduzida de modo a verificar os objetivos definidos. O GQM possui três níveis: conceitual, operacional e quantitativo, conforme apresentado na Figura 9.

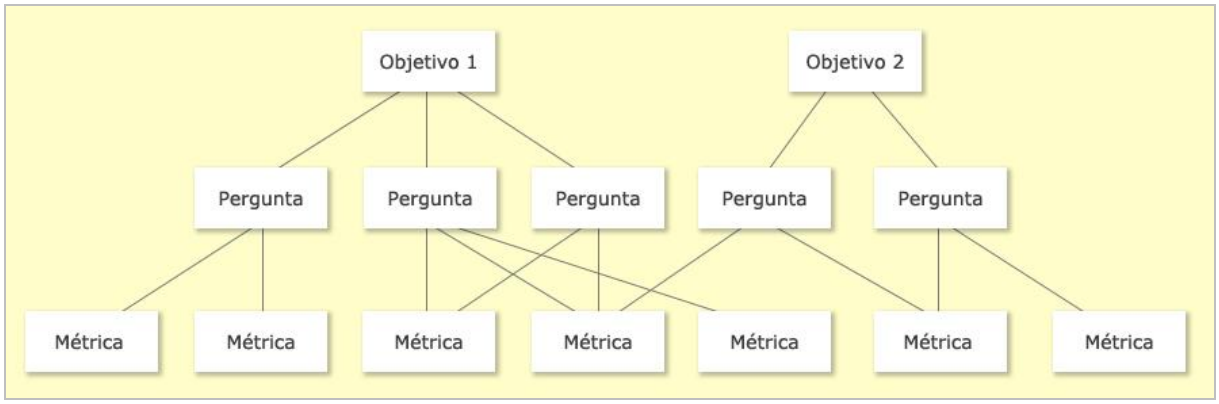


Figura 9 – Estrutura hierárquica do modelo GQM

4.1.1 Nível Conceitual

O Nível Conceitual é o primeiro a ser executado. Neste nível, objetivos mensuráveis devem ser definidos. Considerando a proposta do *framework* Amê, o objetivo do experimento é avaliar se os algoritmos implementados apresentam nível de dificuldade visivelmente diferentes. Em caso positivo o *framework* cumpre seu propósito de avaliar os algoritmos. Para avaliar este objetivo, foram elaboradas quatro hipóteses, sendo uma delas decomposta por três diferentes aspectos, listadas na Tabela 5. Pode-se dizer que, para um determinado aspecto, um algoritmo A é considerado mais difícil que um algoritmo B, quando neste aspecto, o algoritmo A possui uma valoração maior que a do algoritmo B.

Tabela 5 - Hipóteses

		H0	**H1**
C1	Dificuldade do algoritmo		
C1.1	Dificuldade do algoritmo por pontuação	Não há diferença entre os algoritmos	(-) O Guloso é mais difícil
C1.2	Dificuldade do algoritmo por escala	Não há diferença entre os algoritmos	(-) O Guloso é mais difícil
C1.3	Oponente mais desafiador, na visão do usuário	Não há diferença entre os algoritmos	(-) O Algoritmo guloso não é mais difícil
C2	Tempo de jogo antes e depois de leitura	Iguais	(+) Tempo antes maior
C3	Em relação à didática do manual	Igual a cinco	(+) A didática do manual é boa
C4	O jogo é intuitivo	É igual a discordo parcialmente	(+): É intuitivo

4.1.2 Nível Operacional

O Nível Operacional deve ser executado após a conclusão do nível conceitual. Neste momento, perguntas (*Questions*) são elaboradas para mensuração dos objetivos. Desenvolvemos um formulário com as perguntas necessárias para avaliar o *framework* Amê. A composição produziu ao todo 27 perguntas, sendo duas relacionadas ao contato com o voluntário, dez quantitativas, dez qualitativas ordinárias e as demais textuais. A experimentação proposta possui três elementos principais:

- *Player 1*: pessoa que compete com o algoritmo a ser avaliado.
- *Player 2*: o algoritmo que representa o jogador oponente. Pode assumir a estratégia do algoritmo Aleatório ou do algoritmo Guloso.

- *Framework*: responsável por oferecer o ambiente e pelo controle da competição.

Nesta experimentação, o *framework* seleciona de forma aleatória o algoritmo que representa o *Player 2*. O *Player 1* não recebe informação quanto ao nível de dificuldade para derrotar o *Player 2*, tampouco quanto ao algoritmo utilizado por ele. Para que a avaliação seja possível, um ícone de Android é exibido assumindo cores diferentes (veja Tabela 6) de acordo com o algoritmo atribuído ao *Player 2* pelo *framework*.

Tabela 6 - Relação de cor de Android e Algoritmo

Player 2	Cores de Androids
Algoritmo Guloso	
Algoritmo Aleatório	

No primeiro momento, solicitamos que o aluno buscasse jogar o Hanafuda sem ler instruções sobre o jogo. O aluno assume, então, o papel de *Player 1* e espera-se que este não tenha conhecimentos suficientes sobre as regras do jogo, o que torna impossível a elaboração de estratégias para vencer seu adversário. Neste contexto, foi possível avaliar a usabilidade do sistema, bem como o grau de complexidade do Hanafuda e a intuitividade do *framework*, ou seja, se o usuário consegue aprender o jogo em questão através do Amê, aspectos que configuram a hipótese C4.

No segundo momento, foi permitido que o jogador lesse o manual do jogo em um tempo máximo de 15 minutos. Avaliamos então a didática do manual, abordada na hipótese C3. Logo em seguida, a competição foi reiniciada. Neste momento, temos um *Player 1* mais competitivo, por dominar melhor as técnicas do jogo, possibilitando a adoção de estratégias mais elaboradas. Neste cenário, a hipótese C1.1 é testada e avaliada a partir das respostas obtidas por meio do preenchimento do formulário de avaliação (Figura 10).

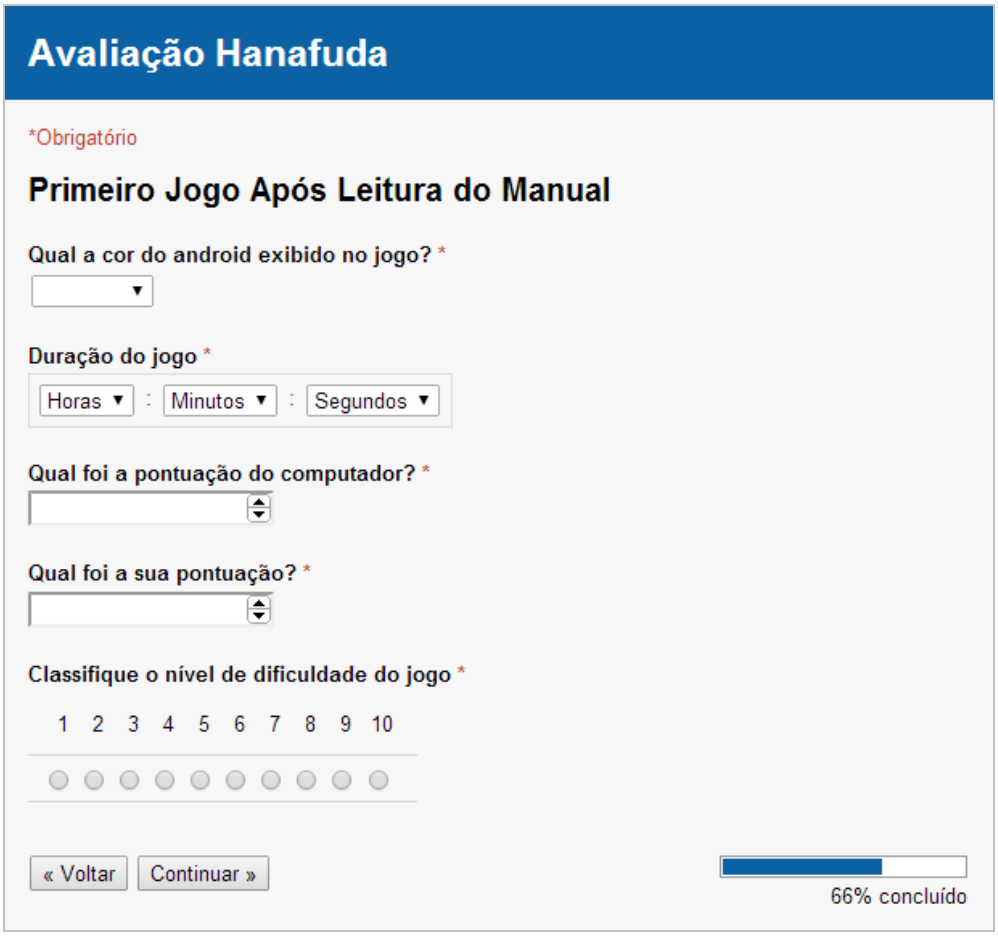


Figura 10 - Tela de formulário com perguntas sobre o jogo

4.1.3 Nível Quantitativo

O Nível Quantitativo é o nível final do método GQM. Após a conclusão do Nível Operacional, os dados coletados devem ser tratados e podem ser usados para mensurar de forma quantitativa o sistema avaliado. Os dados coletados podem ser classificados:

- quantitativos: o dado independe de ponto de vista de quem responde as questões, dependendo apenas do objetivo. Dados desta classificação são predominantemente numéricos.
- qualitativos ordinários: neste caso, os dados coletados são influenciados, além do objetivo, pelo ponto de vista de quem avalia o jogo. Estes dados foram coletados através de perguntas que requisitam a opinião do *Player 1*, como em questões relacionadas à intuitividade do *framework* e quanto à didática do manual do Hanafuda

4.1.3.1 Eficiência dos algoritmos

Para avaliar os algoritmos Aleatório e Guloso, foi considerada a pontuação do *Player 1* e *Player 2* para cada algoritmo adotado, bem como a percepção do usuário quanto a partida que considerou mais difícil. No experimento, cada voluntário jogou três partidas, sendo uma antes da leitura do manual do jogo e duas após a sua leitura. Para termos resultados mais coerentes em relação à percepção do usuário, foram considerados apenas os resultados pós-leitura de manual; em relação à pontuação, consideramos todos os resultados.

A verificação da pontuação foi feita usando duas amostras. As duas amostras contém o saldo de pontos adquiridos através dos jogos. O saldo de pontos consiste na diferença entre pontuação do *Player 1* e pontuação do *Player 2*. Deste modo, saldo positivo indica vitória do *Player 2*, enquanto saldo negativo indica vitória do *Player 1*. A primeira amostra faz referência aos saldos obtidos a partir do algoritmo Aleatório, enquanto o segundo possui dados referentes ao algoritmo Guloso. Tendo como base a análise realizada sobre as amostras, verificamos maior ocorrência de saldos positivos quando o Algoritmo Guloso assume o papel de *Player 2*, como ilustra a Figura 11.

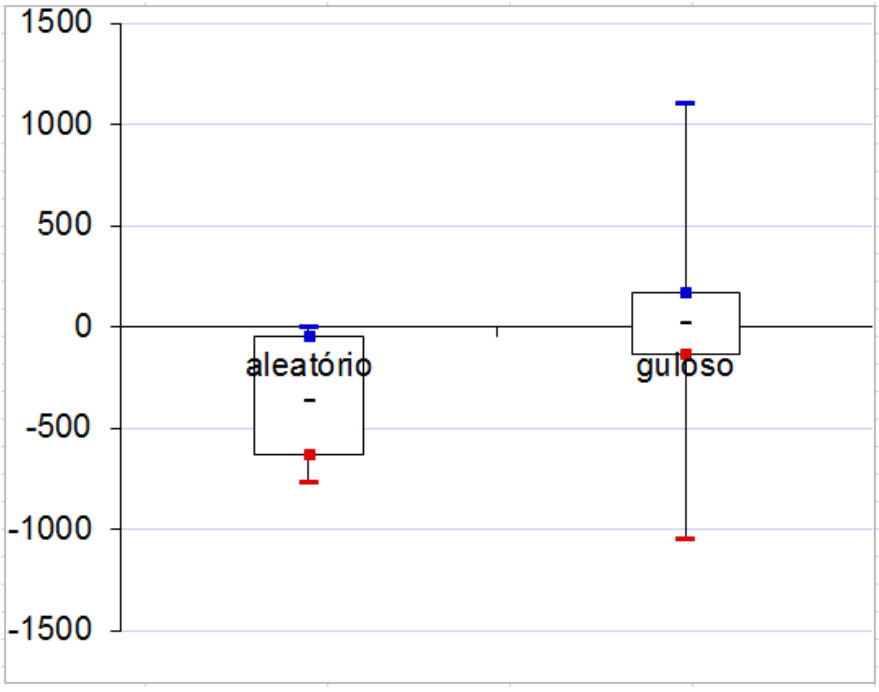


Figura 11 - Gráfico probabilístico de saldos de pontuação

Na verificação da percepção do usuário quanto à partida mais difícil, apenas uma amostra foi necessária. A diferença entre as opiniões dos voluntários foi pequena. A maioria, no entanto, indicou que as partidas que tinham o algoritmo Aleatório foram mais difíceis, ilustrado pela Figura 12. Um dos fatores que podem ter influência nesta avaliação é a mão que do *Player 1* ser relativamente ruim, oferecendo menos possibilidade de combinações.

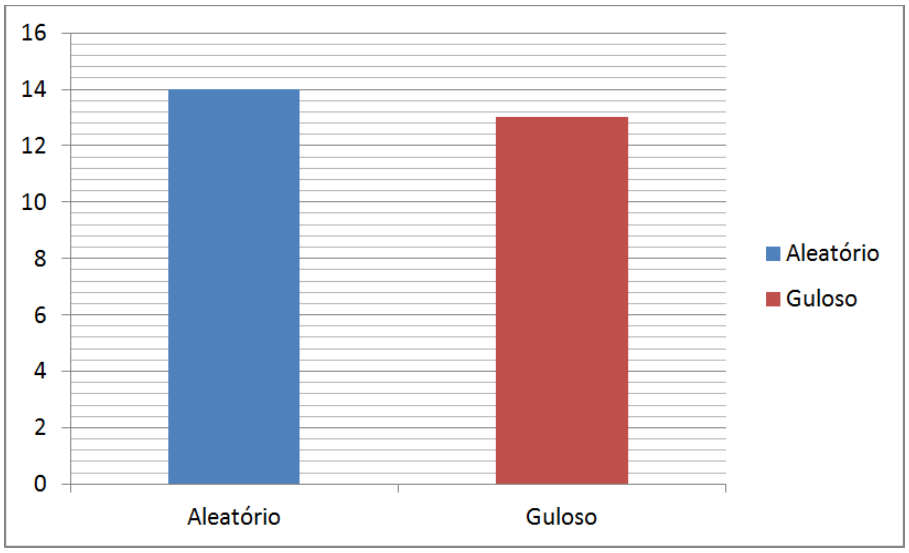


Figura 12 - Percepção do Player 1 quanto ao oponente mais difícil

O resultado desta avaliação é bem interessante e diverge do resultado obtido a partir da análise de pontuação. A análise obtida pela percepção do voluntário faz uso de dados qualitativos ordinários, que estabelecem uma percepção ou sentimento do usuário em relação a dificuldade do jogo, enquanto que a análise por pontuação utiliza dados quantitativos, trazem o score do jogo. A análise por pontuação parece ser, portanto, mais precisa ao indicar que o algoritmo mais desafiador é o Algoritmo Guloso.

De acordo com as respostas coletadas acerca da intuitividade do jogo e das sugestões, foi possível verificar que a maioria dos usuários considerou o Amê não intuitivo. A partir desta avaliação, concluímos que o *framework* deve ser melhorado no aspecto de usabilidade. Apesar de usarmos a plataforma Android, a qual muitos já estão familiarizados, o *framework* se mostro pouco didático. Neste sentido, podemos afirmar que o Amê não contribuiu ativamente para a evolução *Player 1*, principalmente se tratando do entendimento do Hanafuda e suas regras. A disposição da interface gráfica afetou no uso do *framework* Amê. Foram indicados como pontos a ser melhorados: instruções mais claras de quais cartas fazem parte da mão do oponente, mão do usuário, mesa ou monte. Além disso, a falta de uma opção de ajuda com um breve tutorial também foi indicado como falha de usabilidade, uma vez que o jogo é desconhecido. Foi sugerido também, como melhoria de usabilidade, deixar as cartas selecionadas destacadas até a próxima jogada.

Todas essas informações demonstraram que apesar da familiaridade dos usuários com a plataforma Android e com jogos de cartas ocidentais, a implementação inicial do *framework* Amê não tornou o jogo intuitivo. Foi observada a necessidade de potencializar a usabilidade do jogo visando maximizar a experiência do jogador a partir de uma interface gráfica que seja a mais clara possível.

V Conclusão

Este trabalho apresenta o *framework* Amê para avaliação de diferentes algoritmos para jogos computacionais baseados em cartas. O *framework* foi desenvolvido para a plataforma Android por meio da linguagem de programação Java. Para que uma avaliação preliminar fosse realizada, neste projeto, foram implementados dois algoritmos para representar a lógica do computador: Aleatório e Guloso.

Neste contexto foi conduzido um estudo experimental com 27 voluntários, onde foram testadas quatro hipóteses principais visando à avaliação da capacidade do Amê em servir como um *framework* para elaboração de algoritmos para jogos de competição baseados em cartas. Cada voluntário realizava um total de três partidas, sendo a primeira para avaliação da usabilidade do Amê e as seguintes para avaliação de sua capacidade. Com os resultados obtidos com o experimento verificamos que algoritmos mais complexos apresentam maior nível de dificuldade para o usuário. O Algoritmo Aleatório apresentou-se como o mais simples, e foi derrotado com mais frequência, em oposição ao Algoritmo Guloso, que venceu o jogador mais da metade das vezes.

O resultando é interessante, pois, em uma análise preliminar, o *framework* Amê foi eficiente e capaz de apoiar diferentes algoritmos e torná-los perceptíveis para o usuário. Verificamos que o primeiro algoritmo citado pode ser considerado um oponente ideal para o jogador iniciante. Já o segundo algoritmo possui uma estratégia mais rebuscada, se assemelhando a um jogador que já conhece as regras do jogo com capacidade de decisão mais complexa, sendo, portanto um oponente mais difícil de ser derrotado.

Como trabalho futuro, pretendemos continuar as pesquisas até então realizadas. Dentre as medidas para evolução do projeto, incluiremos novos algoritmos de inteligência artificial para comparação e aplicaremos algumas das melhorias que foram sugeridas durante o experimento. A partir das sugestões coletadas, concluímos que é necessário aprimorar a experiência do usuário. Dentre os ajustes necessários para melhoria do *framework* Amê, podemos indicar como ajustes mais necessários: (i) criar área de ajuda com instruções básicas de jogo, como critério para combinação de cartas, onde se encontram as cartas da mão do jogador e do oponente e o monte; (ii) diminuir tempo de mensagens de instrução; (iii) deixar cartas selecionadas destacadas.

REFERÊNCIAS BIBLIOGRÁFICAS

Android Developers, (2013a), *Android - Application Fundamentals*, http://developer.android.com/guide/components/fundamentals.html.

Android Developers, (2013b), *Android - ADT Plugin*, http://developer.android.com/tools/sdk/eclipse-adt.html.

Android Developers, (2013c), *Android - Activities*, http://developer.android.com/guide/components/activities.html.

Astah, (2013), *Astah*, http://astah.net/.

Billings, D., Davidson, A., Schaeffer, J., Szafron, D., (2002), "The challenge of poker", *Artificial Intelligence*, v. 134, n. 1–2, p. 201 – 240.

Blizzard, (2014), *Diablo 3*, http://us.battle.net/d3/pt/.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., (2009), *Introduction to Algorithms*. 3 ed. The MIT Press.

Eclipse, (2013), *Eclipse*, https://www.eclipse.org/.

Electronic Arts, (2014), *Need for Speed*, http://www.needforspeed.com/rivals.

Fowler, M., (2004), *UML distilled: a brief guide to the standard object modeling language*. Boston, Addison-Wesley.

Gold, A., (2005), "Academic AI and Video Games: A Case Study of Incorporating Innovative Academic Research into a Video Game Prototype". In: *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*

Hanafuda, (2013), *Hanafuda - Historical And General Notes*, http://l-pollett.tripod.com/cards9.htm.

Hanafuda, (2014a), *Hanafuda*, http://hanafubuki.org/.

Hanafuda, (2014b), *Hanafuda - Nintendo*, https://club.nintendo.com/rewards-details/a/10505.do.

Hawes, N., (2011), "A survey of motivation frameworks for intelligent systems", *Artificial Intelligence*, v. 175, n. 5–6, p. 1020 – 1036.

Heule, M. J. H., Rothkrantz, L. J. M., (2007), "Solving games: Dependence of applicable solving procedures", *Science of Computer Programming*, v. 67, n. 1, p. 105 – 124.

KOCHANSKI, D., (2011), "Roteiro para apoiar avaliações empíricas de objetos de aprendizagem". In: *17º Congresso Internacional ABED de Educação a Distância*, Manaus, AM, Brasil.

Millington, I., Funge, J., (2009), *Artificial intelligence for games*. Burlington, MA, Morgan Kaufmann/Elsevier.

Ponsen, M., Tuyls, K., Kaisers, M., Ramon, J., (2009), "An evolutionary game-theoretic analysis of poker strategies", *Entertainment Computing*, v. 1, n. 1, p. 39 – 45.

Pressman, R. S., (2006), *Engenharia de software*. McGraw-Hill.

QEMU, (2013), *QEMU*, http://wiki.qemu.org/.

Robertson, J., Howells, C., (2008), "Computer game design: Opportunities for successful learning", *Computers & Education*, v. 50, n. 2, p. 559 – 578.

Rubin, J., Watson, I., (2011), "Computer poker: A review", *Artificial Intelligence*, v. 175, n. 5–6, p. 958 – 987.

Russell, S., Norvig, P., (2009), *Artificial Intelligence: A Modern Approach*. 3 ed. Prentice Hall.

Santana, R. T. de, (2006), *IA em jogos A busca competitiva entre homem e máquina*, Faculdade de Tecnologia de Praia Grande Disponível em: http://www.programadoresdejogos.com/trab_academicos/roberto_tengan.pdf.

Santos, F., Jr, F., Bezerra, E., Quadros, J., Ogasawara, E., (2014), "FastEval: Uma Abordagem Simplificada para Avaliação de Protótipos de Sistemas de Informação Produzidos em Trabalhos de Conclusão de Curso". In: *X Simpósio Brasileiro de Sistemas de Informação*, Londrina, PR, Brasil.

Schuytema, P., Manyen, M., (2005), *Game development with Lua*. Hingham, Charles River Media.

Segundo, R. M. C., (2011), *Athus: um framework para o desenvolvimento de jogos para TV Digital utilizando Ginga*, Universidade Federal da Paraíba Disponível em: http://bdtd.biblioteca.ufpb.br/tde_arquivos/16/TDE-2012-03-19T090200Z-1497/Publico/arquivototal.pdf.

Silva, F. V., Adamatti, D. F., (2012), "Um Jogo de Estratégia Aplicando Técnicas de Inteligência Artificial", *Revista Icceeg Numero 5*, v. 1 (dez.), p. 11 – 19.

Smith, D. K., (2007), "Dynamic programming and board games: A survey", *European Journal of Operational Research*, v. 176, n. 3 (fev.), p. 1299–1318.

Unity, (2013), *Unity*, http://unity3d.com/pt.

Valente, L., (2005), *Guff: um framework para o desenvolvimento de jogos*, UFF

Yannakakis, G. N., (2012), "Game AI Revisited". In: *Proceedings of the 9th Conference on Computing Frontiers*, p. 285–292, New York, NY, USA.

APÊNDICE A – Manual de Instalação do *framework* Amê

O *framework* Amê pode ser instalado em dispositivos móveis que usem o *software* Android e no emulador de Android configurado com o *Android Virtual Device* (AVD). O AVD permite modelar um dispositivo real, definindo opções de *hardware* e *software* a ser emulado pelo emulador Android.

A instalação é realizada através do uso do IDE Eclipse que possua o *plugin* ADT e faz-se necessário possuir o Android SDK instalado. Recomendamos que optem por realizar o *download* do ADT Bundle no *site* do Android Developer[4], pois nele se encontram todos os elementos necessários, incluindo o emulador Android.

1. Instalação em dispositivo móvel real

1. Conecte, via USB, o dispositivo na maquina que contenha o Eclipse com o arquivo do Amê.
2. Habilite o **USB debugging** do dispositivo.
    - Dispositivos com versão Android 3.2 ou menor encontram essa opção em: **Settings > Applications > Development**.
    - Dispositivos com versão Android 4.0 ou maior encontram essa opção em: **Settings > Developer options**. Caso a opção esteja escondida, o que ocorre nas versões 4.2 e acima, vá para **Settings > About phone** e clique no **Build number** sete vezes para habilitar a opção.
3. Abra o projeto Amê no Eclipse e click em **Run** da barra de ferramentas do Eclipse ou botão direito no projeto e selecione: **Run As > Android Application**. Aparecerá a tela de opções de dispositivos Android (**Figure 1**).

[4] http://developer.android.com

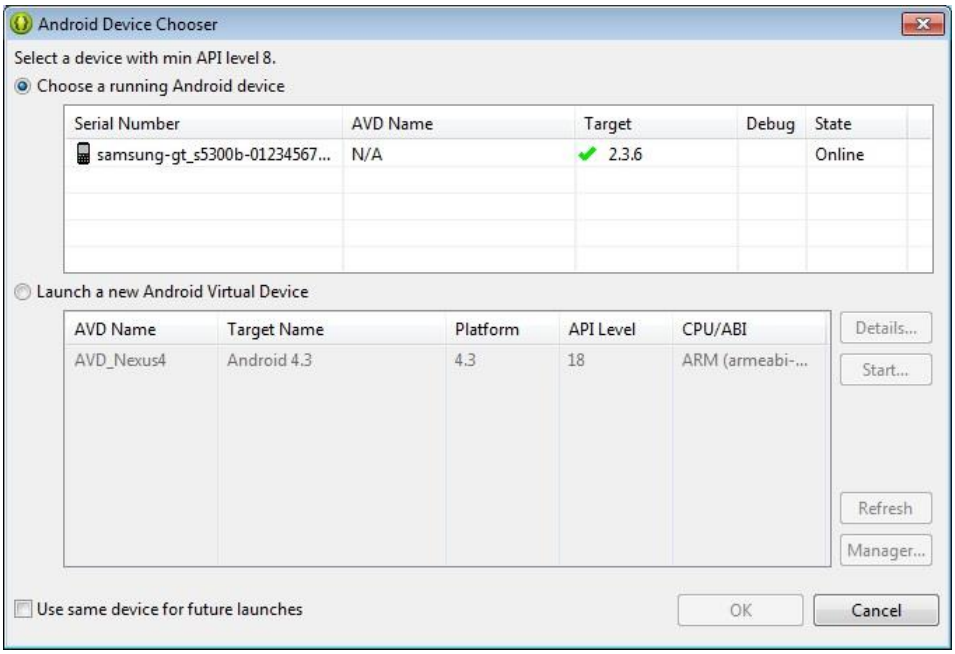


Figure 1 - Android Device Chooser

- Caso a tela não apareça, botão direito no projeto e selecione: **Run As > Run Configurations...**. Na tela (**Figure 2**) clique na aba **Target** e selecione a opção **Always prompt to pick device**.

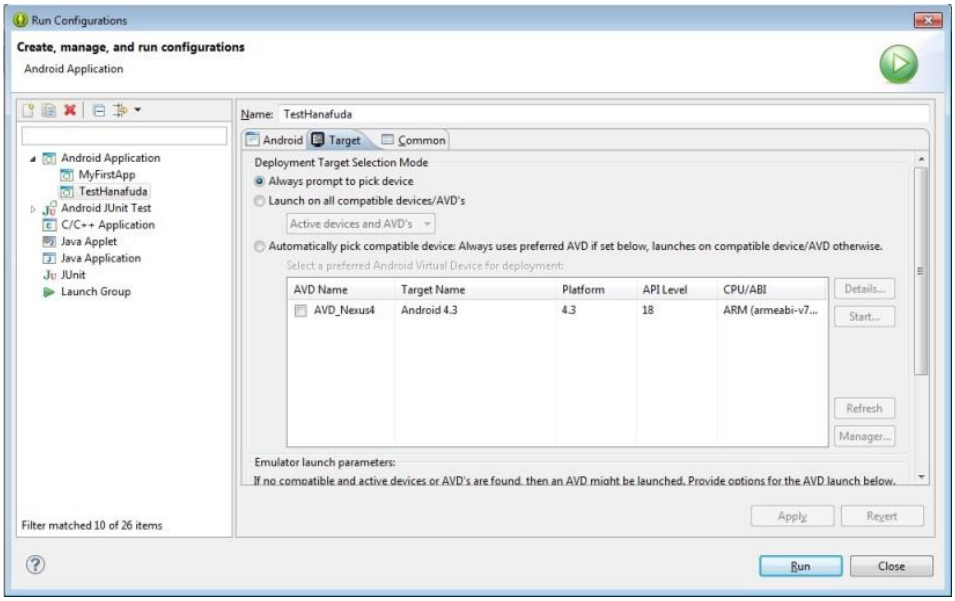


Figure 2 - Run Configurations

4. Selecione o dispositivo e clique em **OK**.
5. O ícone do Amê aparecerá juntamente com os demais aplicativos instalados no dispositivo.

2. Instalação no emulador Android

1. Clique no botão clique em **Android Virtual Device Manager** da barra de ferramentas do Eclipse.
2. Caso já possua o emulador Android criado e configurado pule para o passo 8.
3. Na tela *Android Virtual Device Manager* clique em **New.**
4. Na tela *Create new Android Virtual* Device (AVD), **Figure 3**, coloque o nome do AVD, escolha o dispositivo, selecione a API e adicione o tamanho do cartão de memória.

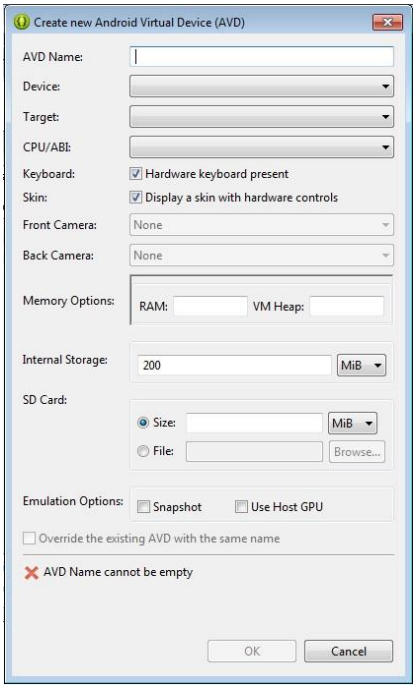


Figure 3 - Create new Android Virtual Device

5. Clique em **OK** para criar o AVD.
6. Selecione o AVD na tela *Android Virtual Device Manager* **Figure 4** e clique em **Start**.

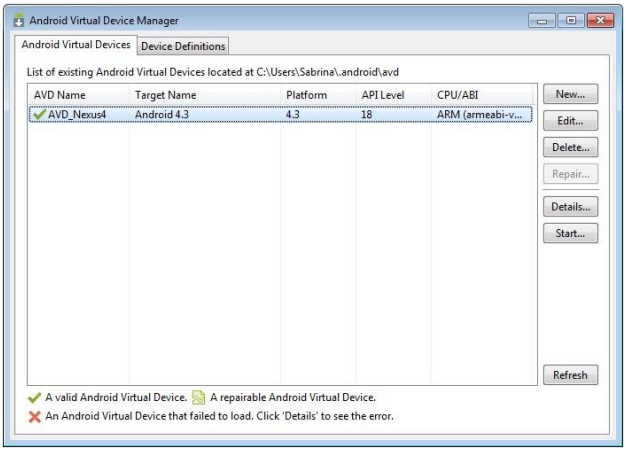


Figure 4 - Android Virtual Device Manager

7. Assim que o emulador terminar de iniciar, destrave a tela do emulador.
8. Abra o projeto Amê no Eclipse e click em **Run** da barra de ferramentas do Eclipse ou botão direito no projeto e selecione: **Run As > Android Application**.
   Aparecerá a tela de opções de dispositivos Android.
   - Caso a tela não apareça, botão direito no projeto e selecione: **Run As > Run Configurations...** Na tela clique na aba **Target** e selecione a opção **Always prompt to pick device**
9. Selecione o AVD e clique em **OK**.
10. O ícone do Amê aparecerá juntamente com os demais aplicativos instalados no emulador.

APÊNDICE B – Experimentação

Para que a realização de uma experimentação fosse viável e bem sucedida, foram elaborados um roteiro e um questionário. Ambos foram desenvolvidos com o objetivo de coletar os dados necessários de forma correta para avaliação do *framework*.

1. Roteiro

Este experimento tem por objetivo a avaliação do projeto Amê e ocorrerá em três etapas. Por favor, execute as três etapas conforme descrito neste roteiro e seja o mais crítico possível.

Etapa 1:

- Antes de iniciar a experimentação, preencha a primeira página do formulário, informando os dados pessoais solicitados.

  As informações obtidas através deste experimento serão utilizadas apenas com finalidade de integrar pesquisas científicas.

Etapa 2:

- Inicie a primeira partida, realizando jogadas até que o final do jogo seja anunciado. Neste momento, responda as questões da segunda página do formulário.

Etapa 3:

- Após a primeira partida, será entregue um manual do jogo, que poderá ser lido em até quinze minutos.
- Após a leitura, responda as questões listadas na terceira página do formulário.
- Realize uma nova partida e, em seguida, responda as questões da página 5 do formulário.
- Após a finalização da partida descrita anteriormente, inicie uma nova partida realizando jogadas até sua finalização. Responda as questões da página 5.
- Ao final, questões sobre o jogo no geral serão levantadas. Responda estas questões, que se encontram na página 6 do formulário.

  O experimento é finalizado com a conclusão do preenchimento da sexta página.

Agradecemos pela sua participação no experimento deste projeto! Sua avaliação foi muito importante para aprimorar este trabalho científico!

2. Questionário

Gostaríamos que você respondesse as questões abaixo para nos ajudar a avaliar este jogo. Um roteiro será entregue para auxiliá-lo durante o preenchimento deste formulário. Os dados aqui obtidos não serão divulgados e serão utilizados somente com finalidade de integrar a pesquisa científica.

1. Data em que foi realizada a pesquisa
2. Nome
3. E-mail
4. Idade
5. Você conhece o Hanafuda e suas regras?
    a. Sim
    b. Não
6. Qual a cor do android exibido no jogo?
    a. Vermelho
    b. Azul
    c. Verde
    d. Laranja
    e. Roxo
    f. Cinza
7. Duração do jogo.
8. Qual foi a pontuação do computador?
9. Qual foi a sua pontuação?

Leitura do Manual

Você terá 15 minutos para ler o Manual do Hanafuda. Continue a responder o questionário somente após o término da leitura.

10. O Jogo é intuitivo? Foi possível compreender as regras do jogo sem auxilio
    a. Discordo plenamente
    b. Discordo parcialmente
    c. Concordo parcialmente
    d. Concordo plenamente
11. Duração da leitura do manual
12. Classifique a didática do manual

a. Escala de 1 a 10

Primeiro Jogo Após Leitura do Manual

13. Qual a cor do android exibido no jogo?
    a. Vermelho
    b. Azul
    c. Verde
    d. Laranja
    e. Roxo
    f. Cinza
14. Duração do Jogo
15. Qual foi a pontuação do computador?
16. Qual foi a sua pontuação?
17. Classifique o nível de dificuldade do jogo.
    a. Escala de 1 a 10

Segundo Jogo Após Leitura do Manual

18. Qual a cor do android exibido no jogo?
    a. Vermelho
    b. Azul
    c. Verde
    d. Laranja
    e. Roxo
    f. Cinza
19. Duração do Jogo
20. Qual foi a pontuação do computador?
21. Qual foi a sua pontuação?
22. Classifique o nível de dificuldade do jogo.
    a. Escala de 1 a 10

Final

23. Qual dos jogos teve o oponente mais desafiador? Indique qual foi o oponente mais difícil.
    a. Primeiro jogo após leitura do manual
    b. Segundo jogo após leitura do manual
    c. Ambos foram igualmente desafiadores
24. A interface induz o usuário a jogar bem?

a. Discordo plenamente
b. Discordo parcialmente
c. Concordo parcialmente
d. Concordo plenamente

25. Quantos bugs o jogo apresentou durante a pesquisa?
26. Sugestões
27. Explique os bugs encontrados. Caso tenham sido detectados bugs, indique os problemas que foram encontrados.