

Universidade Federal do Maranhão - UFMA
Centro de Ciências Exatas e Tecnológicas
Curso de Ciência da Computação

RAFAEL REGO DRUMOND

WANDA: UM FRAMEWORK PARA
DESENVOLVIMENTO DE JOGOS DE CARTAS PARA
O ENSINO DE ALGORITMOS

São Luís - MA
2014

RAFAEL REGO DRUMOND

**WANDA: UM FRAMEWORK PARA
DESENVOLVIMENTO DE JOGOS DE CARTAS PARA
O ENSINO DE ALGORITMOS**

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profº Carlos de Salles Soares Neto

São Luís - MA

2014

Rego Drumond, Rafael

Wanda: Um framework para desenvolvimento de jogos de cartas para o ensino de algoritmos/ Rafael Rego Drumond. – 2014.

66 p.

Impresso por computador (Fotocópia).

Orientador: Carlos de Salles Soares Neto

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2014.

1. Programa de computador 2. Algoritmos - Ensino 3. Jogos de cartas - Programa - Desenvolvimento 4. Wanda - Framework 5. Jogos educacionais I. Wanda: Um framework para desenvolvimento de jogos de cartas para o ensino de algoritmos

CDU 004.421

RAFAEL REGO DRUMOND

**WANDA: UM FRAMEWORK PARA DESENVOLVIMENTO DE
JOGOS DE CARTAS PARA O ENSINO DE ALGORITMOS**

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 17 de Dezembro de 2014.

BANCA EXAMINADORA



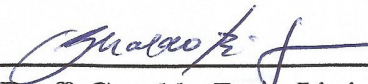
Prof^o Carlos de Salles Soares Neto
(Orientador)

Universidade Federal do Maranhão

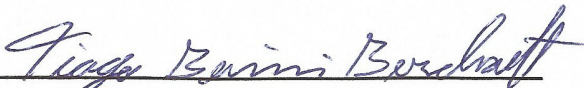


**Prof^o Alexandre César Muniz de
Oliveira**

Universidade Federal do Maranhão



Prof^o Geraldo Braz Júnior
Universidade Federal do Maranhão



Prof^o Tiago Bonini Borchardt
Universidade Federal do Maranhão

*Para meus irmãos:
Lucas, Enio, Thomas e Gentil*

Agradecimentos

Em primeiro lugar a Deus, por tudo que tem me dado durante minha vida.

Aos meus pais Alberto e Celi, por lutarem junto comigo durante meus momentos mais difíceis e por serem os melhores pais que o mundo já viu.

Ao meu irmão Lucas, pelo carinho, amizade e por estar sempre ao meu lado para me dar apoio quando precisei.

Aos meus “irmãos de mentira”, Enio, Gentil e Thomas, por serem irmãos, amigos e companheiros de aventuras e sempre prontos para me dar forças.

À minha família, com quem cresci e me ensinou valores em que acredito até hoje.

Ao meu orientador Carlos Salles e a todos os membros do LAWS, e em especial ao André, Hedvan e Rosendy, pela ajuda apoio e orientação.

Ao professor Alexandre “ACMO” e ao PET-COMP, pelo companheirismo, amizade e pela experiência de grupo incrível, que contribuíram para a formação do meu espírito de Equipe.

Ao meus amigos do Quinteto, André, Lucas, Neoma e Rodrigo, pela amizade mais duradoura que já tive e pelo constante apoio tanto em questões acadêmicas e não acadêmicas.

Aos professores Omar Carmona e Eraldo Cardoso e aos membros da Kendo-Aikido-SLZ e Shotokay, pelo espírito de família e por me ajudarem a moldar meu caráter através dos ensinamentos que me proporcionaram disciplina e perseverança para chegar até aqui.

À AMAGames e à Gorba Games, por todo o apoio, e por me ajudarem a fazer gostar do meu trabalho cada vez mais.

Ao meus queridos amigos de Champaign, tanto os “Brasileiros de Champaign” como os meus amigos da UIUC, Mike Mallinowski e Jason Cho, por serem minha família enquanto estive longe de casa.

A todos os meus outros amigos (que são muitos para listar aqui), e que sempre me deram apoio (e outros que me deram carona).

A todos aqueles que contribuíram para este trabalho, inclusive os calouros de Ciência da Computação de 2013.2 e 2014.1.

*“Muitos que vivem merecem
a morte. E alguns que
morrem merecem viver. Você pode
dar-lhes a vida? Então não seja
tão ávido para julgar e condenar alguém a morte.
Pois mesmo os muitos sábios
não conseguem ver os dois lados.”*
(J.R.R. Tolkien, A Sociedade do Anel)

RESUMO

Ensinar ciência da computação, algoritmos, e outros assuntos sobre programação pode ser uma tarefa estressante e difícil. A motivação de estudantes pode cair devido a dificuldade de se aprender a programar. Para lidar com isto, este trabalho apresenta Wanda, um arcabouço simples que pode ser usado para criar jogos de cartas específicos que podem ser usados como ferramentas educacionais. Com Wanda, professores podem criar jogos eletrônicos de cartas que requerem que o aluno crie um agente virtual para jogá-los. Estes jogos podem ser utilizados por professores e tutores para motivar e ensinar alunos de programação.

Palavras-chaves: Programa de computador. Algoritmos - Ensino. Jogos de cartas - Programa - Desenvolvimento. Wanda - Framework. Jogos educacionais.

ABSTRACT

Teaching computer science, algorithms and other programming subjects can be stressing and difficult. Students motivation might drop due to the difficulty of learning how to program. To deal with this matter, this work presents Wanda, a simple framework created to build specific card-based games that can be used as educational tools. With Wanda, professors can create educational electronic card games that require students to create virtual agents to play them. These games can help teachers and tutors to motivate and teach programming students.

Keywords: Computer program. Algorithms. Card games - Program - Development. Wanda - Framework. Educational games.

Lista de ilustrações

Figura 1 – Telas do jogo <i>Big Brain Academy</i>	19
Figura 2 – Diagrama do fluxo do jogo	28
Figura 3 – Diagrama Conceitual do Arcabouço Wanda	29
Figura 4 – Diagrama de Visão e Controle	31
Figura 5 – Diagrama de Objetos de um jogo criado por Wanda.	33
Figura 6 – Chamadas de funções em um jogo do Wanda.	35
Figura 7 – Exemplo de partida do jogo <i>Jo-Ken-Po</i>	41
Figura 8 – Diagrama de vitória das cartas do jogo <i>Jo-Ken-Po</i>	42
Figura 9 – Campeonato do jogo <i>Jo-Ken-Po</i>	43
Figura 10 – Exemplo de partida do jogo <i>Divide and Conquer</i>	45
Figura 11 – Diagrama de vitória das cartas do jogo <i>Divide and Conquer</i>	46
Figura 12 – Gráfico comparativo de respostas de uma pergunta sobre Auto-Eficiência	50
Figura 13 – Gráfico comparativo de respostas de uma pergunta sobre Valor do aprendizado da matéria	51
Figura 14 – Gráfico comparativo de respostas de uma pergunta sobre Objetivo de Conquista	51
Figura 15 – Comparativo gráfico de respostas de uma pergunta sobre Estimulo por Ambiente de Aprendizado	52
Figura 16 – Gráfico comparativo de respostas de uma pergunta sobre Estimulo por Ambiente de Aprendizado	52

Lista de códigos

Código 4.1 – Exemplo de código para inicialização de cartas	35
Código 4.2 – Exemplo de código para regras de conflito	36
Código 4.3 – Implementação padrão do Wanda da função roundfight()	37
Código 4.4 – Definição padrão do Wanda para definição de estratégias	39
Código 5.1 – Inicialização do baralho do jogo <i>Jo-Ken-Po</i>	42
Código 6.1 – Função de conflito do jogo Divide-and-Conquer	46
Código 6.2 – Inicializando o baralho para o jogo Divide-And-Conquer	47
Código B.1 – Estratégia Jo-Ken-Po	60
Código B.2 – Estratégia Divide-and-Conquer	63

Lista de tabelas

Tabela 1 – Tabela de comparação de trabalhos relacionados com o Wanda	26
Tabela 2 – Funções dos Objetos de Visão e Controle	32

Lista de abreviaturas e siglas

CADES	Card Game Development System
MaI	Mechanics and Interface
MeI	Mecânica e Interface
MVC	Model-view-controller
PNG	Portable Network Graphics
PUCPR	Pontifícia Universidade Católica do Paraná
SDK	Software Development Kit
SDM	Software Development Manager
SPARSE	Software Project semi-Automated Reasoning tool for Software Engineering
UML	Unified Modeling Language

Sumário

1	INTRODUÇÃO	15
1.1	Objetivos	15
1.2	Histórico do Trabalho	16
1.3	Organização do Trabalho	17
2	FUNDAMENTAÇÃO TEÓRICA	18
2.1	Jogos Eletrônicos e Jogos Sérios	18
2.2	Jogos de Cartas	19
3	TRABALHOS RELACIONADOS	21
3.1	Ensino Propulsor	21
3.2	Software Developer Manager	22
3.3	Chien2D	22
3.4	Aprendendo a ensinar programação combinando jogos e <i>Python</i>	23
3.5	SPARSE	24
3.6	CADES	25
3.7	Comparação dos trabalhos	25
4	ARCABOUÇO WANDA	27
4.1	Modelo Conceitual	29
4.1.1	Mecânicas e Interface	29
4.1.2	Regras	29
4.1.3	Estratégias	30
4.2	Arquitetura e Implementação do Arcabouço	30
4.3	Processo de autoria de um jogo usando Wanda	34
4.3.1	Inicializando o Baralho (<i>Deck</i>)	34
4.3.2	Definindo as Regras de Conflito	36
4.3.3	Descrevendo o funcionamento de uma rodada	37
4.3.4	Especificando os argumentos das estratégias (<i>strategy()</i>)	39
5	JO-KEN-PO	41
5.1	Apresentação do Jogo	41
5.2	Instanciação do Wanda	42
5.3	Experimentação	43
6	DIVIDE-AND-CONQUER	45
6.1	Apresentação do Jogo	45

6.2	Instanciação do Wanda	46
6.3	Experimentação	47
7	QUESTIONÁRIO PARA MEDIR MOTIVAÇÃO	48
7.1	Descrição do Questionário	48
7.2	Análise de Resultados do Questionário	49
7.2.1	Auto-Eficiência	49
7.2.2	Valor do aprendizado da matéria	50
7.2.3	Objetivo de Conquista	50
7.2.4	Estímulo por Ambiente de Aprendizado	51
8	CONCLUSÃO	54
	Referências	56
	ANEXO A – QUESTIONÁRIO DE MOTIVAÇÃO	58
	ANEXO B – EXEMPLOS DE ESTRATÉGIAS	60
B.1	Exemplo de estratégia para o jogo Jo-Ken-Po	60
B.2	Exemplo de estratégia para o jogo Divide-and-Conquer	63

1 Introdução

Os cursos de graduação da área de TI no Brasil, tais como Ciência da Computação, Sistemas de Informação e Engenharia da Computação, possuem uma alta taxa de desistência (TAKAHASHI, 2009). Uma razão apontada está relacionada à dificuldade em aprender programação (KINNUNEN; MALMI, 2006), sendo um processo difícil para o aluno devido ao fato de estar lidando com um tipo de tarefa diferente do que ele está acostumado. Isso se torna preocupante quando observamos a falta de profissionais qualificados para suprir a demanda de vagas de TI no mercado de trabalho que vêm aumentando drasticamente ano após ano (CELSO JR., 2009).

Professores de programação muitas vezes esbarram na dificuldade de tentar encontrar formas de atrair e motivar os alunos enquanto estes aprendem e exercitam seus conhecimentos sobre algoritmos. Professores e instrutores devem utilizar estratégias intrínsecas dentro da sala de aula para motivar os alunos (HUITT, 2011). Essas estratégias estão relacionadas com o uso de jogos, simulações e atividades que permitam que os alunos definam seus próprios objetivos. Uma abordagem eficaz para auxiliar no problema é utilizar tarefas para as quais o aluno é familiar ou consegue se relacionar com ela (BAYLISS; STROUT, 2006). Para alcançar este objetivo diversos trabalhos foram feitos utilizando jogos e outras atividades ligadas com a área.

Este trabalho apresenta Wanda (DRUMOND; DAMASCENO; SOARES NETO, 2014), um arcabouço que permite que professores da área de informática e computação possam desenvolver jogos baseados em cartas que requerem que o aluno programe um agente virtual para realizar suas jogadas. As aplicações desenvolvidas com este arcabouço podem ser criadas em um tempo curto e sem precisar escrever muitas linhas de código. Estas aplicações auxiliam no ensino de conceitos básicos de programação e ajudam a superar as dificuldades em aprender programação.

1.1 Objetivos

O objetivo deste trabalho é apresentar um arcabouço para desenvolver jogos de cartas que permitam que professores e instrutores de programação possam usar para aprimorar o processo de ensino-aprendizagem de algoritmos mediante a programação de agentes para jogar tais jogos.

Espera-se que este arcabouço apresente as seguintes características:

- Seguir um esquema de regras de jogos de cartas específico mas que possibilite a

criação de diversos jogos diferentes.

- Ser de fácil utilização permitindo que um professor ou instrutor de uma matéria sobre programação possa criar uma aplicação rapidamente sem precisar de muitas linhas de código.
- Possuir um formato onde, na aplicação final, um agente implementado por aluno irá realizar as jogadas do jogador.
- Possuir a capacidade de mostrar graficamente as jogadas feitas durante a partida.
- Jogos criados com Wanda devem possuir código aberto para permitir que alunos interessados possam investigá-lo.

1.2 Histórico do Trabalho

Inicialmente foi feita uma pesquisa sobre o motor de jogos LOVE 2D (LÖVE-2D, 2014) para descobrir suas funcionalidades básicas enquanto se pesquisou sobre jogos de cartas para descobrir aqueles que podem ser simples o suficiente, ou que podem ser simplificados.

Na etapa seguinte, baseada nos jogos encontrados na etapa anterior, foi estabelecido um sistema de jogos de cartas padrão para o trabalho que possa ser reaproveitado de diversas maneiras para a construção de diversos jogos e que a programação de agentes para que este seja acessível a alunos iniciantes em programação.

A próxima etapa se caracterizou pela criação do arcabouço Wanda, tomando como referência os jogos planejados e o sistema elaborado. Wanda contempla os requisitos de ambos e possibilita a criação de jogos utilizando a implementação de agentes para jogar as aplicações desenvolvidas.

Após essa etapa, já tendo elaborado o arcabouço Wanda, os exemplos de jogos de cartas foram implementados e testados, entregando-os para os alunos. Cada jogo necessita de certos conhecimentos, e devem ser aplicados em momentos diferentes de acordo com o julgamento do professor. Os alunos devem elaborar agentes para jogar os jogos e em seguida competir com outros alunos. Tal competição foi realizada dentro da sala de aula.

Para analisar os resultados, um questionário foi respondido pelos alunos. Esse foi elaborado para medir a motivação dos mesmos. O questionário possui forma de formulário garantindo perguntas diretas e objetivas mas com espaço para os alunos escreverem sobre suas experiências. O questionário foi aplicado antes e depois de cada atividade com os jogos criados com o arcabouço para medir a motivação do aluno em tais momentos de modo a medir os efeitos da atividade através da comparação de resultados. Os resultados

obtidos com o questionário foram analisados e comparados para extrair informações para saber se os jogos tiveram efeitos no aprendizado ou motivação.

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica que sustenta este trabalho. Nele são descritos os conceitos de Jogos Eletrônicos, Jogos de Cartas e Jogos Sérios.
- O Capítulo 3 faz um apanhado de trabalhos similares ou com objetivos parecidos com este.
- O Capítulo 4 apresenta o arcabouço desenvolvido: WANDA, detalhando sua arquitetura, conceito e explicando sua utilização.
- Os capítulos 5 e 6 apresentam dois casos de usos utilizados para testar a utilização do arcabouço proposto.
- O Capítulo 7 apresenta um questionário para medir a motivação dos alunos utilizado para medir os impactos dos casos de teste.
- O Capítulo 8 conclui o trabalho.
- O Anexo A possui perguntas do formulário descrito no Capítulo 7.

2 Fundamentação Teórica

Este trabalho trata de um arcabouço para desenvolvimento de jogos eletrônicos baseados em jogos de cartas no formato de jogos sérios. Este capítulo visa explicar tais conceitos para facilitar o entendimento do trabalho. A Seção 2.1 dá um embasamento de jogos eletrônicos e jogos sérios, enquanto a Seção 2.2 descreve o conceito de jogos de cartas.

2.1 Jogos Eletrônicos e Jogos Sérios

Os jogos digitais podem ser definidos como ambientes de interação atrativos que possibilitam capturar e chamar a atenção do jogador. Estes oferecem desafios que exigem crescentes níveis de destreza e habilidades enquanto fornece divertimento ao jogador (BALASUBRAMANIAN; WILSON, 2006). Tais jogos consistem em uma das principais formas de acesso ao mundo da tecnologia, visto que crianças e jovens tem acesso a este mundo pela primeira vez por meio destes na maioria dos casos (GROS, 2003).

Os jogos eletrônicos ocuparam uma alta posição do mercado Superando o mercado da música e do cinema nas estimativas de 2008 (MARKETING-CHARTS, 2008) e o Brasil foi classificado como o quarto maior mercado de games do mundo em 2012(SEBRAE, 2008).

Jogos Sérios (ou Jogos Educativos) é um conceito atribuído para aqueles jogos eletrônicos que possuem como objetivo ensinar enquanto divertem e distraem o jogador, tendo um cunho pedagógico. A maioria dos jogos educativos possuem o conteúdo a ser abordado, exposto abertamente e são frequentemente utilizados em instituições de ensino (NOVAK, 2007). Contudo diversos Jogos Sérios surgiram no mercado utilizando as proposta de quebra-cabeça (ou *puzzles*) como por exemplo os jogos *Brain Age* e *Big Brain Academy* (Figura 1) da empresa *Nintendo* que tem como objetivo exercitar a mente do jogador com diversos enigmas (NOVAK, 2007).

Um ponto interessante dos jogos eletrônicos que os jogos sérios utilizam é o fato de que muitos jogadores buscam dominar o jogo, ou seja, jogam o jogo para ganhar, de modo que o jogador abraça o ensinamento passado, aprendendo voluntariamente com o jogo Sério (NOVAK, 2007).

É possível citar oito benefícios principais relacionados com o uso de Jogos Sérios na educação (SAVI; ULBRICHT, 2008):

- Motivar o aprendizado;

Figura 1 – Telas do jogo *Big Brain Academy*

Fonte: gamesradar.com

- Facilitar o aprendizado;
- Desenvolver habilidades cognitivas;
- Desenvolver a capacidade de explorar, colaborar e experimentar;
- Experienciar imersão em outros ambientes para aprender sobre eles;
- Socializar com outros jogadores;
- Desenvolver coordenação motora;
- Tornar o jogador *expert* no assunto abordado.

2.2 Jogos de Cartas

Jogos de cartas vêm sendo jogados por muitos anos pela população do mundo em diversos tipos como papéis-cartão pequeno, pedaços finos de madeira, ou até materiais como marfim. Tais jogos podem variar desde jogos que utilizem sorte ou habilidades do jogador (CAVALCANTE, 2006). A popularidade garante que diversas pessoas sejam familiares com jogos de cartas e consigam aprender com facilidade (WOPC, 2014).

Jogos de Cartas consistem basicamente em jogos no qual se utilizam cartas para serem jogados. Em geral tais jogos incluem mais de um jogador onde cada um desses deve utilizar cartas de acordo com as regras do jogo para alcançar a vitória seguindo estratégia e as vezes um pouco de sorte. Existem infinitas possibilidades de jogos que podem ser jogados utilizando um baralho tradicional ou baralhos não convencionais (CAVALCANTE, 2006) que trazem cartas específicas para aquele tipo de jogo, ao contrário do tradicional que pode ser utilizado para jogar diversos jogos com as mesmas cartas. Dentre os diversos exemplos de jogos podemos citar os mais conhecidos como (TROIS, 2012): *Poker*, Pif-Paf, Buraco, *Rummy* e Truco. Todos eles possuem regras e exigem que o jogador elabore estratégias baseada nas cartas em jogo e nas que ele tem na mão.

Os termos mais comumente utilizados para se referir a jogos de cartas são (WOPC, 2014):

- Carta: Representa um cartão que possui um valor próprio podendo ou não estar repetido em outras do baralho.
- Baralho: Coleção de cartas dispostas de forma aleatória. Geralmente é usado para suprir os jogadores com cartas.
- Embaralhar: Misturar as cartas de um baralho para deixá-las em ordem aleatória no mesmo.
- Mão: Consiste nas cartas em posse do jogador. Na maioria dos jogos, a “mão” só pode ser vista pelo jogador.
- Compra: Colocar uma carta do topo de um baralho para a mão.
- Fornecedor de Cartas (*dealer*): Jogador que fornece cartas para outros jogadores.
- Truque: A cada turno uma carta é jogada por cada jogador, vence aquele com a carta mais poderosa.
- Trunfo: Em alguns jogos, um determinado naipe é escolhido como o trunfo. Os outros são naipes simples. Qualquer carta de um naipe trunfo é mais poderosa que qualquer outra carta de um naipe simples.
- Carta Líder: É a primeira carta jogada para um truque. Na maioria dos casos, ela determina o naipe que será trunfo pela rodada corrente.
- Turno: É o momento reservado para cada jogador executar uma ação.
- Rodada: Representa um conjunto de turnos. Depois que cada jogador tiver terminado seu turno, a rodada é dita terminada, podendo ou não ser reiniciada.

3 Trabalhos Relacionados

Existe uma vasta literatura sobre desenvolvimento de jogos para fins educacionais. Para validar o trabalho proposto, foi feito um apanhado de trabalhos similares com objetivos e áreas semelhantes. Este capítulo aborda brevemente os trabalhos analisados e compará-los com o arcabouço desenvolvido neste trabalho.

A organização deste capítulo consiste em descrever um trabalho por seção, com o trabalho “Ensino Propulsor” descrito na Seção 3.1, “SDM” na 3.2, “Chien2D” na 3.3, “Ensino de programação combinando jogos e *Python*” na 3.4, “SPARSE” na 3.5 e “CADES” na 3.6. A Seção 3.7 faz um comparativo destes trabalhos com o arcabouço Wanda.

3.1 Ensino Propulsor

O projeto “Ensino Propulsor” (KESSLER, 2010) consiste em construir jogos digitais para alavancar o processo de ensino e aprendizado de diferentes áreas acadêmicas de português, matemática, química, física e estatística, além de ensinar os conceitos básicos de cada uma dessas áreas. O ensino propulsor tem como objetivo auxiliar na redução dos índices de repetência e evasão da universidade.

Os jogos são criados utilizando a plataforma *Adobe Flash* devido a possibilidade de desenvolver programas interativos utilizando recursos que agilizam a criação de jogos didáticos. A concepção de novos aplicativos reúne coletar imagens, textos e animações e definir como será a interação entre o jogador e os elementos de modo a motivá-lo e transmitir conteúdo.

Os jogos desenvolvidos como estudo de caso se basearam em concepções de “revisão de conceitos” e “construção de conhecimento”. Como os alunos que participaram dos testes eram alunos universitários, os testes se basearam em resgatar conhecimentos de níveis anteriores de modo a fortalecer a base de conhecimento dos alunos para superar obstáculos de futuras construções de conhecimento.

Um estudo qualitativo, feito a partir de questões abertas, buscou analisar a contribuição dos jogos na aprendizagem dos acadêmicos e com ele foi possível validar o projeto em questão.

O Ensino Propulsor, contudo, não é focado para o ensino de algoritmos.

3.2 Software Developer Manager

Software Developer Manager (SDM) (KOHWALTER; CLUA; MURTA, 2011) é um jogo de computador que simula uma empresa de desenvolvimento de software, onde o jogador deve administrá-la de modo a atender os requisitos de clientes que solicitam o desenvolvimento de produtos. O trabalho tem como objetivo treinar o jogador e ensinar a ele conceitos e práticas de engenharia de software aproveitando do fator da diversão de um jogo eletrônico. O jogo SDM apresenta a importância da administração de uma equipe de desenvolvimento de software, destacando a atenção que se deve dar para as capacidades individuais de cada membro.

O jogo permite que o jogador administre desde os funcionários até os projetos. Os funcionários possuem diversas características descritas como atributos e especializações. Funcionários podem receber treinamentos, serem afetados por condições de trabalho e outros fatores. Os projetos podem ser administrados desde a negociação com o cliente, porém seu sucesso está ligado diretamente às relações humanas presentes na equipe.

O jogo utiliza o motor gráfico *Unity3D* (UNITY-TECHOLOGIES, 2014) e foi entregue a alguns grupos de alunos (boa parte composta de alunos de graduação) o qual testaram o jogo respondendo um questionário sobre o mesmo. A maioria destes alunos respondeu o questionário subsequente com um *feedback* positivo, indicando interesse, aprendizado e divertimento da parte deles.

Apesar de interessante e de rápida aplicação, o foco de tal trabalho é ensinar conceitos de engenharia de software e não conceitos mais básicos de programação.

3.3 Chien2D

A biblioteca Chien2D (RADTKE; BINDER, 2010) foi desenvolvida com o propósito de ensinar alunos a utilizar a linguagem de programação C utilizando métodos de aprendizado baseados em problemas. Para utilizar a biblioteca, foi criada uma metodologia chamada de *Games Tutorial*, a qual esquematiza um roteiro de aulas sobre a biblioteca tendo jogos criados pelos alunos como produto final além do aprendizado da linguagem C.

Desenvolver um jogo com Chien2D é feita de maneira linear sem a necessidade de implementar objetos. Dessa forma, Chien2D não requer que o aluno saiba orientação a objeto (como exigem a maioria das outras bibliotecas e motores de jogos), mas exige que este saiba conceitos básicos de algoritmos. Outra característica da biblioteca são as funções traduzidas para o português (já que a biblioteca destina-se a alunos brasileiros) além de possuírem nomes que representam de maneira literal a sua funcionalidade. Um código de um jogo feito com Chien2D consiste apenas de uma função linear com um laço infinito dentro do qual são implementadas as animações, interações e outros elementos do jogo.

A metodologia *Games Tutorial* consiste em quatro passos:

- Ensinar os alunos sobre a arquitetura e operações básicas com a biblioteca.
- Desenvolver um jogo simples.
- Elaborar uma ideia e um protótipo de um jogo, criando uma interface e menus de jogo.
- Desenvolvimento de um jogo completo, levando em conta o protótipo criado além de elaborar um *website* e um manual para o jogo.

Um estudo de caso foi realizado com um grupo de alunos da universidade PUCPR, onde estes tiveram aulas baseadas no *Games Tutorial*. Após realizar uma pesquisa, constatou-se que a atividade foi relevante para 92% dos alunos e ajudou 50% a aprenderem conceitos de programação, enquanto 20% se sentiram auxiliados a dominar a linguagem C. Apesar de não haver comprovações quanto a relação com a aplicação da metodologia, a taxa de evasão do curso diminuiu se comparada com as anteriores ao trabalho com a Chien2D.

O trabalho se mostra muito útil para o ensino de linguagens de programação e pode ser adaptado para o ensino de conceitos mais básicos, porém consumirá muito tempo e esforço dos instrutores para aplicá-lo, especialmente em uma turma de iniciantes em programação.

3.4 Aprendendo a ensinar programação combinando jogos e *Python*

Este trabalho trata-se de uma investigação de métodos motivantes para o ensino de programação através da linguagem *Python* e programação de jogos (REBOUÇAS, 2010). A ideia principal consiste em investigar formas de aumentar o interesse de alunos pela área de informática enquanto os torna capazes de criar jogos simples utilizando a biblioteca *pyGame* (SHINERS, 2014). Os jogos criados neste projeto são de cunho educacional e visam auxiliar o aprendizado de certos assuntos de matérias escolares.

Após estudar e aprender sobre a linguagem *Python* e sobre a biblioteca *PyGame*, alunos de Licenciatura em Ciência da Computação ministraram aulas de programação de jogos para uma turma de estudantes de ensino médio e conduziram seus alunos para a criação de aplicações educativas. Os três jogos criados pelos alunos que mais se destacaram foram os PyQuímica, o PyLavra e o PyGonometria que buscam ensinar conceitos sobre química, ortografia e trigonometria respectivamente.

O trabalho mostrou que o uso de desenvolvimento de jogos em cursos de computação podem aumentar a motivação dos alunos e contribuir para sua formação. Observou-se

também que os jogos criados por alunos também seriam úteis para reforçar o aprendizado de conteúdos com os quais estes tinham dificuldades.

O projeto é capaz de ensinar conceitos básicos de programação, contudo o tempo gasto com as tarefas e atividades pode ser considerado como extenso.

3.5 SPARSE

O software SPARSE (*Software Project semi-Automated Reasoning tool for Software Engineering*) (SOUZA M.M., 2010) é um jogo que tem o objetivo de simular um escritório de engenharia de software. O jogador assume o papel de um gerente de projeto necessitando interagir com os desenvolvedores conhecendo suas habilidades além de participar das etapas de criação do projeto.

O simulador apresenta alguns elementos essenciais para o desenvolvimento de um software como:

- Modelo de processo: conjunto de atividades pré-definidas e parcialmente ordenadas para definir a ordem do projeto;
- Projeto de software: modelo de processo instanciado com desenvolvedores e restrições de prazo e custo, tendo o objetivo final de entregar o software pronto para o cliente;
- Gerente de projetos: o jogador, que deve administrar todo o processo de desenvolvimento do projeto;
- Desenvolvedores: funcionários responsáveis por desenvolver o software do projeto. Possuem vários atributos a serem controlados pelo gerente;
- Ferramentas: softwares ou técnicas de apoio aos desenvolvedores no desenvolvimento do projeto.

SPARSE foi entregue para alguns alunos de graduação em ciência da computação. Os participantes da atividade foram escolhidos de diferentes etapas do curso, aplicando em seguida um questionário para definir o perfil dos alunos. Após os alunos terem jogado o jogo, um segundo questionário foi aplicado para medir os resultados e verificar a experiência individual de cada um. Os jogadores se mostraram satisfeitos com a atividade e o questionário mostrou um aumento do conhecimento sobre engenharia de software, até nos mais experientes da área.

O trabalho contudo não é aplicável para o ensino de conceitos básicos de programação, sendo somente para ensinar conceitos de engenharia de software.

3.6 CADES

O arcabouço CADES (*Card-Game Development System*) (CAVALCANTE, 2006) tem como objetivo facilitar a criação de jogos de cartas, reunindo as características que a maioria dos jogos têm em comum, elaborando assim um esqueleto para qualquer jogo eletrônico. Embora não tenha um suporte para a parte gráfica, CADES possibilita a programação de um jogo de carta de qualquer tipo e até mesmo jogos de tabuleiros. O tempo e a dificuldade para se programar um jogo com CADES variam de acordo com sua complexidade e convencionalidade.

Programado em C++ e utilizando conceitos de orientação a objeto, o CADES possui classes de objetos genéricas dos elementos utilizados no jogo. Os relacionamentos destes objetos seguem o relacionamento usual presente em tais jogos, o que facilita o autor da aplicação a criar um jogo. Os objetos principais já implementados no arcabouço são:

- Rules (Regras)
- Player (Jogador)
- Board (Mesa)
- Card (Carta)
- Deck (Baralho)
- Card Collection (Coleção de Cartas)
- Hand (Mão)
- Round (Rodada)
- Turn (Turno)
- Action (Ação)

Como caso de uso foram implementados alguns jogos mais simples, como Divide-and-Conquer (CELKO; MCLEOD, 2014) e alguns outros mais complexos sem consumir muito tempo.

O CADES não possui suporte para criação de interface gráfica do jogo e utilizam uma tela de console para serem jogados.

3.7 Comparação dos trabalhos

Os trabalhos apresentados neste capítulo serão comparados entre si. As características de cada trabalho foram dispostas na Tabela 1 para uma melhor visualização. As características levadas em consideração foram:

- Algoritmos: O trabalho (ou aplicações geradas) é destinado ser usada para ensinar algoritmos para iniciantes.
- Tempo: O trabalho pode ser executado em um tempo curto ou ele pode gerar aplicações com pouco tempo.
- Instrução: O trabalho requer pouca instrução para ser utilizado ou aplicado.
- Esforço: O trabalho requer pouco esforço para ser utilizado ou aplicado.
- *Feedback* Visual: O trabalho proporciona uma interface gráfica para o usuário final.
- Código Aberto: ó trabalho ou a aplicação gerada por ele é mantido em código aberto.

Tabela 1 – Tabela de comparação de trabalhos relacionados com o Wanda

	Algoritmos	Tempo	Instrução	Esforço	<i>Feedback</i> Visual	Código Aberto
Ensino Propulsor		X	X	X	X	
SDM				X	X	
Chien2D	X				X	X
Jogos com Python	X			X	X	
SPARSE				X	X	
CADES		X	X	X		X

Esta comparação dos trabalhos foi feita baseada em características de uma ferramenta para ensinar algoritmos para alunos iniciantes em programação, sem que o instrutor precise gastar muito tempo e esforço para construir uma aplicação ou tarefa para isso. Tal ferramenta deve passar um *feedback* visual da interação com o aluno além de deixar o código aberto para despertar a curiosidade dos mesmos. Através desta comparação, podemos concluir que nem todas as ferramentas conseguem suprir tais necessidades. O arcabouço Wanda, apresentado neste trabalho, busca atender todas as necessidades citadas.

4 Arcabouço Wanda

O arcabouço Wanda oferece para professores e instrutores considerável reuso de código para instanciar novos jogos baseados em cartas. O objetivo é que tutores de matérias de programação possam desenvolver jogos para sua classe e os alunos devem jogá-los através da criação de agentes que possam especificar ações dado um estado do jogo. Enquanto o aluno implementa a sua estratégia em seu agente, este exercita seus conhecimentos sobre lógica de programação e algoritmos.

Wanda é capaz de auxiliar a criação de jogos que seguem as regras especificadas pela Figura 2 como padrão. Essas regras são:

1. No início da partida um conjunto de cartas específico é embaralhado.
2. O conjunto embaralhado é dividido entre dois jogadores. As cartas que cada um recebe nesta etapa representam a “mão” do jogador.
3. O jogo é dividido em rodadas.
4. No início de cada rodada, os jogadores devem escolher e descartar uma carta de suas respectivas mãos ao mesmo tempo.
5. As cartas são comparadas de acordo com regras específicas do jogo que definem qual carta subjuga a outra.
6. O jogador que tiver jogado a carta que subjugou a outra, vence a rodada e contabiliza um ponto para si.
7. Novas rodadas são jogadas até que não restem mais cartas nas mãos de cada jogador.
8. Após o fim de todas as rodadas, o jogador que obtiver mais pontos é considerado vencedor da partida.

É importante lembrar que tais regras são apenas uma base, podendo ser alteradas em diversos pontos para adaptar jogos com regras de estruturas semelhantes.

O trabalho do professor é alimentar o arcabouço com as regras do jogo definindo e criando o seguinte:

- i. O conjunto de cartas a serem utilizadas na aplicação
- ii. As regras de conflito ou, em outras palavras, as regras que definem qual carta subjugará a outra.

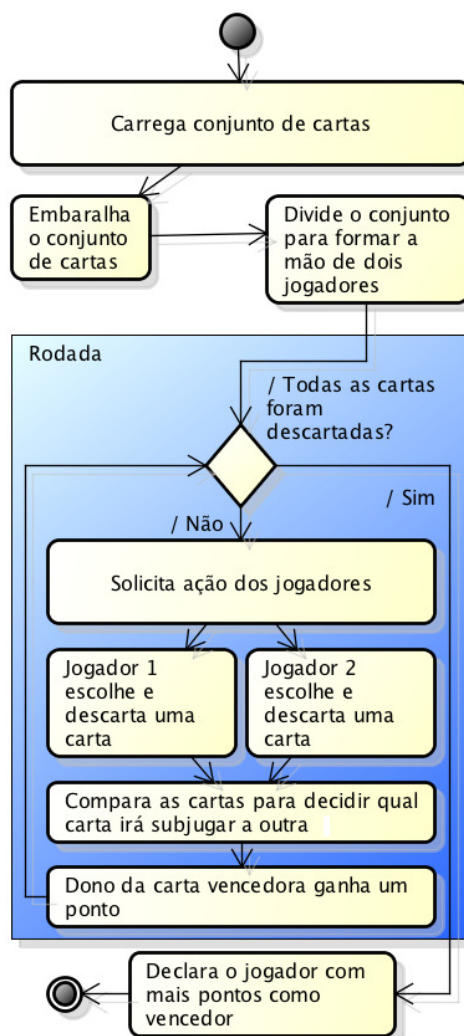


Figura 2 – Diagrama do fluxo do jogo

- iii. Quais informações estarão disponíveis ao aluno e como elas estarão dispostas. Por exemplo: O jogador poderá ver quais cartas estão na sua mão e na mão do adversário dispostos em vetores. Em outro jogo, o jogador poderá ver somente duas cartas de sua mão escolhidas aleatoriamente em forma de variáveis.
- iv. Imagens para ilustrar o jogo.

O instrutor em seguida deve entregar ao aluno a aplicação resultante para que se possa dar início à atividade. O aluno deve finalizar a implementação da aplicação escrevendo a sua “estratégia” diretamente no código da aplicação do jogo em uma função pré-construída. Os argumentos da função são definidos pelo professor que irão representar as informações as quais o jogador tem acesso naquela rodada. Utilizando tais argumentos, o aluno deve retornar a ação do jogador (por padrão: a carta que ele decidir usar naquela rodada).

Como o jogo deve ser jogado por dois jogadores, dois alunos devem submeter uma

implementação de estratégia (cada) para competir entre si.

A Seção 4.1 descreve o modelo conceitual adotado pelo arcabouço. A Seção 4.2 apresenta como o Wanda foi projetado e explica a implementação do arcabouço. A Seção 4.3 explica o processo de criação de um jogo usando Wanda.

4.1 Modelo Conceitual

Wanda pode ser dividido conceitualmente em três partes diferentes: *Mecânica e Interface (MeI)* (ou *Mechanics and Interface, MaI*); *Regras* (ou *Rules*); e *Estratégias* (ou *Strategies*). Cada uma dessas partes tem um papel diferente na aplicação final. A relação entre as três partes é representada pela Figura 3.

4.1.1 Mecânicas e Interface

MeI representa o núcleo do arcabouço. Essa parte não é necessária saber como funciona totalmente ou realizar quaisquer alterações para utilizar o Wanda. *MeI* controla as funções básicas do jogo, como: Embaralhar, distribuir e remover cartas do jogo; gerenciar a pontuação, animação e outras informações visuais da tela; e ligar com as regras especificadas pela parte *Regras*.

4.1.2 Regras

Regras agrupa funções que definem as regras do jogo. O instrutor ou professor é responsável pela implementação desta parte. Essa parte é chamada pela *MeI* para definir o conjunto de cartas a ser usado no início do jogo, assim como definir as texturas usadas pelas cartas. Para todas as partidas inicializadas, a *MeI* faz uma chamada para a *Regras*

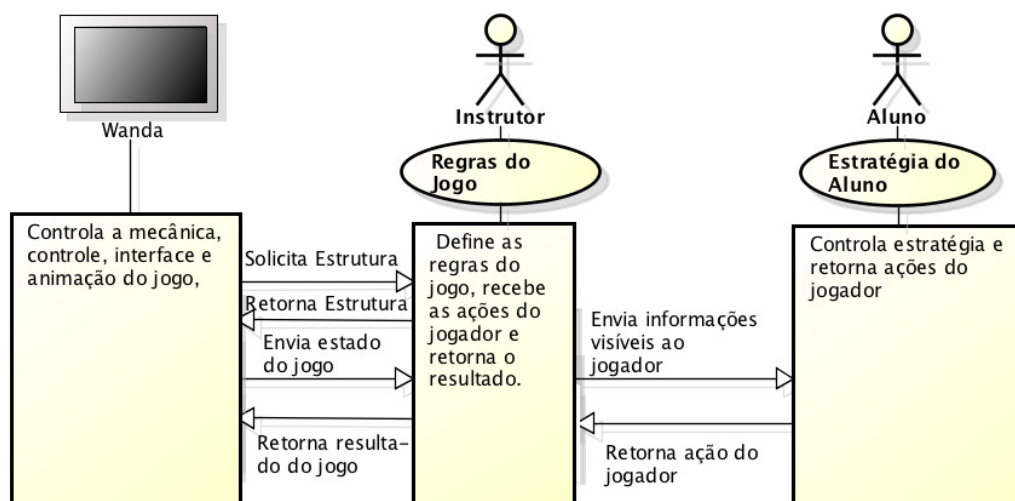


Figura 3 – Diagrama Conceitual do Arcabouço Wanda

a cada rodada, esta por sua vez irá receber o estado do jogo, formatá-lo e enviar para o código dos alunos através do *Estratégias* e em seguida recebe as ações de ambos jogadores computando os efeitos causados por elas, como a vitória da rodada e indicando quais cartas serão descartadas. Estes resultados devem chegar a *MeI* que irá exibir a animação das ações tomadas.

4.1.3 Estratégias

A parte *Estratégias* é o foco principal do arcabouço, pois é aqui onde o aluno irá exercitar suas práticas de programação. *Regras* deve enviar para essa parte a situação do jogo e, em troca, irá receber a ação tomada pelos jogadores na atual rodada. A tarefa do estudante é implementar uma função deixada em branco propositalmente pelo instrutor (o instrutor irá escrever os argumentos previamente para receber os dados do jogo através dos argumentos).

A função programada pelo aluno deve analisar o estado do jogo e retornar qual a ação deve ser tomada baseada nessa análise. O professor deve focar em como a informação é enviada: como *arrays*, tabelas, listas e outros.

Como o propósito deste trabalho é desenvolver um jogo para motivar e/ou alunos de programação, logo o jogo deve exigir certos conhecimentos de algoritmos baseados nos que a maioria da classe (com a qual está sendo trabalhada) possui.

4.2 Arquitetura e Implementação do Arcabouço

Wanda foi implementado em Lua (IERUSALIMSCHY; WALDEMAR; FIGUEIREDO, 2014) utilizando o motor gráfico para jogos: Löve2D (LÖVE-2D, 2014). O motor foi escolhido pelo fato de utilizar a linguagem Lua que é mais simples e fácil de usar que linguagens mais famosas como C, C++, C#, Python ou Java, linguagens mais utilizadas por motores populares como PyGame (SHINERS, 2014) ou Unity (UNITY-TECHOLOGIES, 2014), pois deve-se levar em consideração que os alunos em foco são iniciantes em programação). A possibilidade de deixar o código aberto para o aluno e de não necessitar instalar softwares muito grandes (mais de 100 megabytes de dados), foi outro fator favorável para a escolha da *engine*. O código deixado aberto facilita a implementação da estratégia do aluno dentro do próprio arcabouço além de instigar a curiosidade dele sobre a implementação do jogo.

Duas funções utilizadas pelo Wanda são chamadas de *Draw* e *Update*. A função *draw* é uma função da biblioteca da Löve a qual deve ser escrita pelo usuário do motor. Dentro da *Draw* devem ser chamadas todas as funções gráficas, ou seja aquelas que exibem imagens na tela. *Update* deve ser preenchida com as funções que atualizam os dados do jogo em tempo real.

A arquitetura do Wanda foi criada segundo o modelo MVC (*Model-View-Controller*) (BURBECK, 1992) a qual é descrita pelo diagrama de Visão e Controle da Figura 4. Para dividir o arcabouço apropriadamente de acordo com o modelo MVC, o objeto *View* (Visão) é definido por todas as funções que são chamadas dentro da função *draw()* (da engine *Löve*) para desenhar na tela os elementos visuais do jogo (como figuras, pontuações e outras informações do jogo). O objeto *Controller* (Controle) é definido pelas funções chamadas dentro de *Update()*, essas controlam o fluxo do jogo e garantem o funcionamento, animação e chama as funções criadas pelo usuário do arcabouço. Os objetos Visão e Controle podem ser entendidos como os componentes da *MeI* (Seção 4.1) e suas funções são descritas pela Tabela 2.

A parte *Model* (Modelo) do planejamento MVC é constituída por objetos representados pelo diagrama de objetos da Figura 5. Os objetos marcados em azul representam os pontos de flexibilidade do arcabouço, ou seja, aqueles que irão necessariamente ser construídos pelo usuário do Wanda. Cada objeto representa um elemento do jogo:

- *Deck*: Representa o baralho do jogo que é distribuído entre os jogadores no início de cada partida. É inicializado pelo objeto *Rules* ao criar um array de objetos *Card*.
- *Card*: Representa cada carta do jogo, pode ser representado por qualquer tipo de

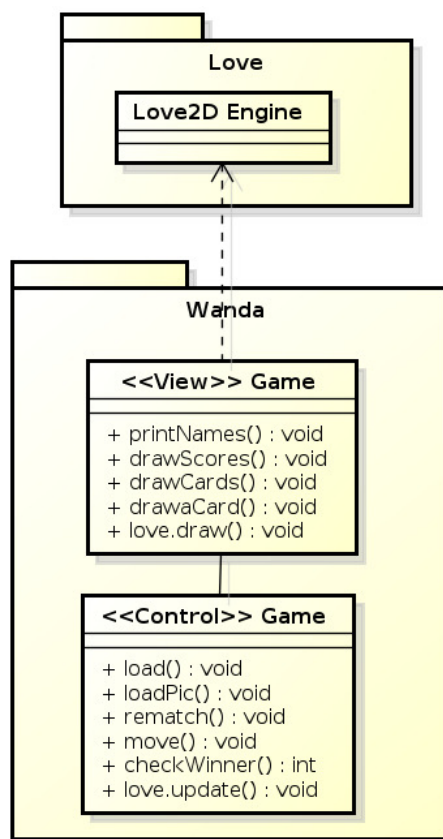


Figura 4 – Diagrama de Visão e Controle

Tabela 2 – Funções dos Objetos de Visão e Controle

Objeto Visão	
Nome	Descrição
<code>printNames()</code>	Imprime os nomes e legendas na tela de jogo.
<code>drawScores()</code>	Imprime as pontuações, tanto da rodada quanto partidas vencidas de cada jogador na tela de jogo.
<code>drawCards()</code>	Desenha as cartas em jogo na tela de jogo em suas respectivas posições.
<code>drawaCard()</code>	Utilizado pela função <code>drawCards()</code> , imprime individualmente uma carta em jogo.
<code>love.draw()</code>	Função do motor gráfico Löve que deve ser preenchido por seu usuário. Nela todas as funções do Visão são chamadas assim como funções do motor para desenhar a imagem de fundo e os personagens dos jogadores.
Objeto Controle	
Nome	Descrição
<code>load()</code>	Função que inicia o jogo. Seu papel é inicializar todas as variáveis necessárias para o jogo funcionar. Aqui também são carregadas as funções criadas pelo usuário do arcabouço.
<code>loadPic()</code>	Função para auxiliar o usuário a definir imagens específicas para representar um certo tipo de carta.
<code>rematch()</code>	Reinicia a partida. Quando chamada, o baralho é reorganizado, as cartas redistribuídas e as pontuações da partida são zeradas.
<code>move()</code>	Realiza a atualização da posição das cartas jogadas para realizar a animação.
<code>checkWinner()</code>	Verifica se a partida chegou ao final e se sim, retorna o vencedor.
<code>love.Update()</code>	Função do motor gráfico Löve que deve ser preenchido por seu usuário. Nela todas as funções do Controle são chamadas bem como as funções do usuário do arcabouço, externas ao objeto.

informação.

- *CardImage*: Guarda uma imagem que representa uma carta graficamente.
- *Player*: É o objeto que representa um dos jogadores. Cada jogador deve implementar sua função *strategy()* para realizar as jogadas.
- *Hand*: É o objeto que agrupa as cartas presentes nas mãos de um jogador.
- *HandCopy*: Um objeto que pode ser inicializado por *copyHand()*, este representa uma cópia da mão de um jogador e poderá ser usado pelo autor do jogo para auxiliar na criação do jogo.
- *Rules*: É o objeto que deve ser implementado pelo criador do jogo. Aqui ele pode descrever a inicialização do jogo através da função *loadSetup()*. As regras de disputa entre as cartas podem ser implementadas em *conflict()*. O fluxo do jogo já vem

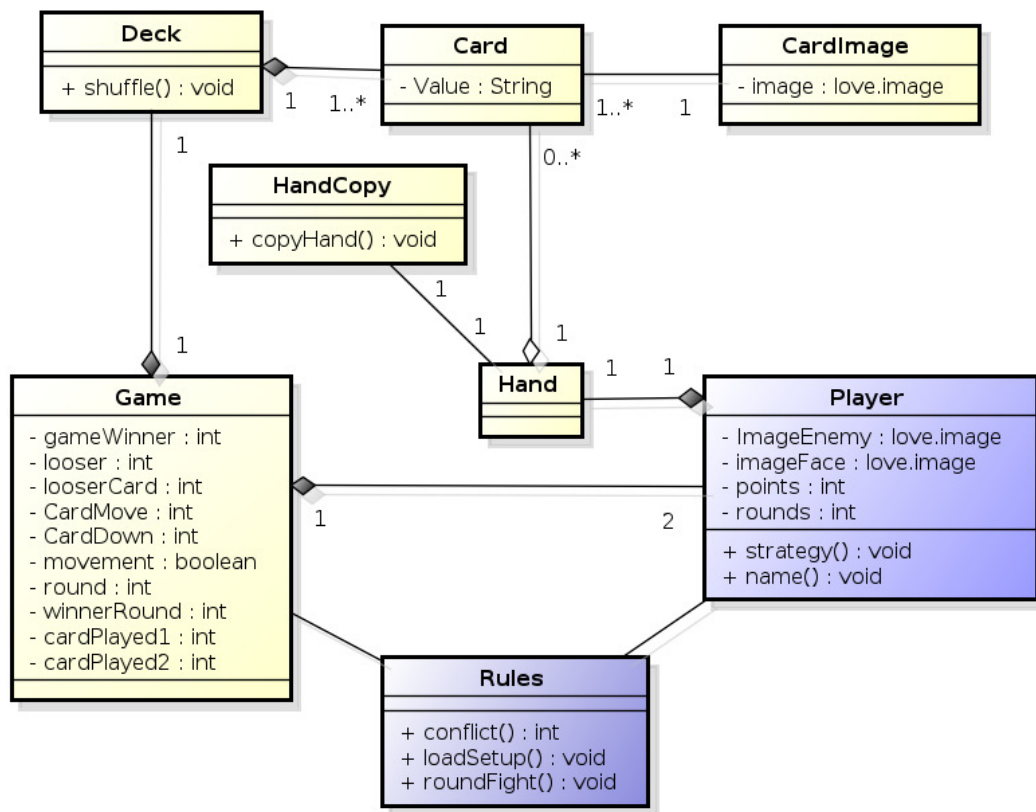


Figura 5 – Diagrama de Objetos de um jogo criado por Wanda.

pré-montado na função *roundFight()*, mas pode ser alterada com rapidez para atender as regras específicas de cada aplicação.

- *Game*: É o objeto que guarda todas as informações do jogo. Se trata do agrupamento de variáveis que vão definir o estado do jogo, tanto informações sobre a mecânica do jogo quanto sobre o funcionamento do arcabouço.

Um jogo criado com o arcabouço Wanda possui o seguinte funcionamento interno:

- 1 A função *load()* é chamada para inicializar as variáveis e chamar a função *loadSetup()* do objeto *Rules*, para carregar o objeto *Deck* (baralho) definido pelo autor.
- 2 A função *load()* carrega os objetos *cards* do *Deck* em um vetor e cria os objetos *hands* com *Cards* aleatoriamente.
- 3 Enquanto todas as cartas não tiverem sido usadas:
 - 3.1 A *love.update()* chama a função *roundFight()* do *Rules* para obter o resultado da rodada.
 - 3.2 A *roundFight()* chama a função *strategy()* do *Player 1* e do *Player 2*.

- 3.3 A *roundFight()* compara a jogada retornada das funções *strategy()* com a função *conflict()*.
- 3.4 A *roundFight()* atualiza o número de rodadas vencidas do *Player* que realizou a melhor jogada.
- 4 Caso todas as partidas não tenham sido jogadas, a função *rematch()* é chamada para reiniciar a partida, e aumenta-se os pontos do *player* que venceu mais rodadas (zerando a contagem de rodadas vencidas e voltando para o terceiro passo).
- 5 Caso todas as partidas tenham sido jogadas, a função *drawScores()* passará a imprimir o vencedor na tela.

A Figura 6 mostra como as funções são chamadas em paralelo com o fluxo do jogo explicado na Figura 2.

4.3 Processo de autoria de um jogo usando Wanda

Para criar um jogo usando Wanda é preciso implementar quatro funções diferentes do objeto *Rules*: *loadSetup()* e *loadImages()* para inicializar o baralho do jogo e as imagens das cartas; *conflict()* para definir as regras de conflito entre as cartas; *roundFight()* para definir o funcionamento das rodadas. Também deve ser implementada a função do objeto *Player*, *strategy()* para especificar os argumentos a serem enviados para os agentes dos jogadores. As subseções a seguir irão explicar como proceder em cada parte da implementação do jogo. Exemplos maiores podem ser encontrados nos Capítulos 5 e 6.

4.3.1 Inicializando o Baralho (*Deck*)

Para definir quais cartas serão usadas no jogo é necessário definir um vetor (chamado “cards”) dentro da função *loadsetup()* com o valor das cartas (números inteiros, cadeias de caracteres ou mesmo tabelas em lua). É necessário também definir uma quantidade de ciclos de partidas atribuindo um valor para a variável *amount_of_matches*.

Por padrão, as imagens que representarão cada carta serão carregadas de arquivos no formato *.png* com o mesmo nome delas na pasta *textures*. Contudo, alguns valores de cartas não podem ser convertidos em nomes de arquivos, como tabelas ou vetores. Para evitar problemas como esse, é possível carregar uma imagem usando a função *loadpic()* onde o usuário passa o nome de uma imagem e ele carrega uma imagem do diretório *textures* o qual deve ser atribuído ao vetor *cardsimage* na posição equivalente ao valor da carta representado pela imagem.

No código a seguir é apresentado um exemplo de configuração de um jogo. Nesse exemplo, temos um jogo composto por quatro cartas (enumeradas de 1 a 4), com cem

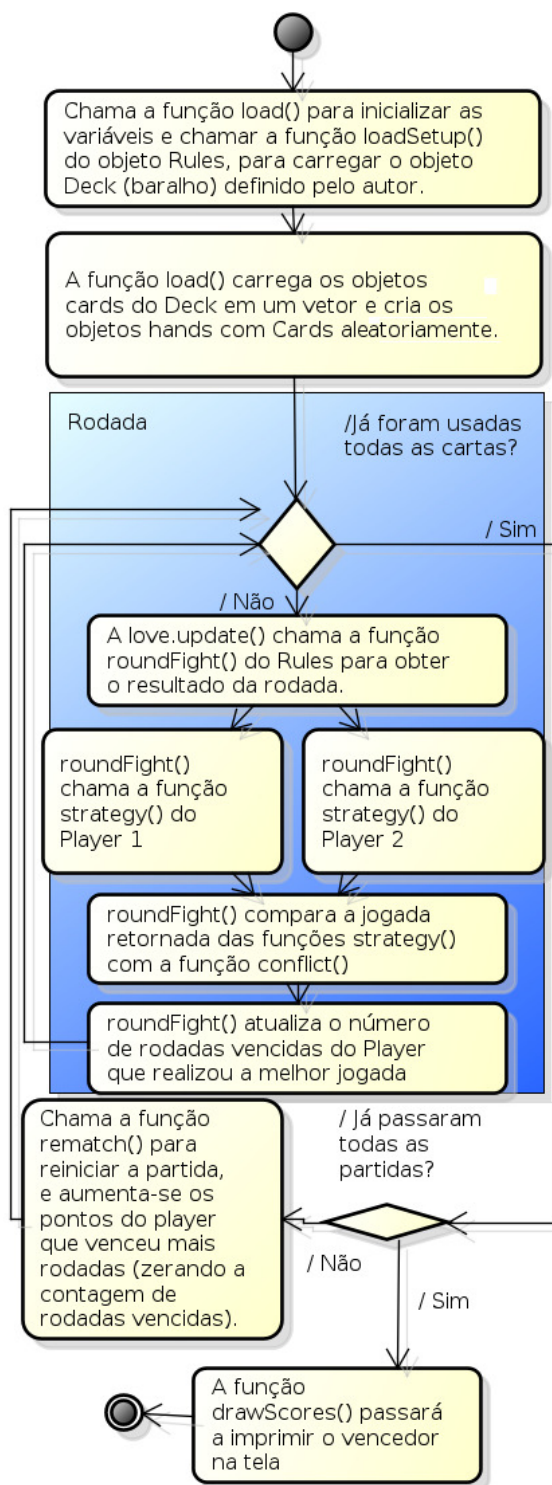


Figura 6 – Chamadas de funções em um jogo do Wanda.

partidas e com a imagem da carta 1 sendo definida por um arquivo chamado “cartinha.png” ao invés do nome de acordo com o padrão “1.png”. Caso não tivesse sido carregada uma imagem específica para a carta “1”, a aplicação iria tentar carregar uma imagem com o nome “1.png”.

Código 4.1 – Exemplo de código para inicialização de cartas

```
function loadsetup ()
    cards = {1,2,3,4}
    amount_of_matches = 100
end

function loadimages ()
    cardsimage [cards [1]] = loadpic ("cartinha.png")
end
```

4.3.2 Definindo as Regras de Conflito

Durante cada rodada, dois jogadores deverão jogar uma carta, deve haver uma comparação entre as duas que poderá resultar nos seguintes resultados: “carta do jogador 1 vence”, “carta do jogador 2 vence” ou “nenhuma carta vence”. Tais resultados são necessários para conferir pontos de cada rodada para cada jogador em cada partida. O autor da aplicação deve escrever o código da função *conflict()*.

A função *conflict()* recebe dois argumentos chamados *p1* e *p2*, cada um representa uma das cartas jogadas em uma rodada pelo Jogador 1 e Jogador 2, respectivamente. A função deve retornar o número 1 caso a carta do Jogador 1 vença a rodada, ou 2 se o Jogador 2 vencer a rodada. Caso um empate entre as cartas ocorra, a função deve retornar 0.

No exemplo a seguir, as regras de conflito das cartas se baseiam no valor inteiro delas. Ou seja, vence a carta que tiver o maior número. Um empate ocorrerá se ambas tiverem o mesmo valor.

Código 4.2 – Exemplo de código para regras de conflito

```
function conflict (p1, p2)
    if p1==p2 then
        return 0 —rodada com empate
    end
    if p1>p2 then
        return 1 —jogador 1 vence a rodada
    else
        return 2 —jogador 2 vence a rodada
    end
end
```

4.3.3 Descrevendo o funcionamento de uma rodada

Essa etapa é uma das etapas mais importantes, pois ela define o funcionamento do jogo. Para definir o funcionamento de uma rodada de um jogo criado com Wanda, é necessário implementar a função *roundfight()*. Wanda já possui uma implementação padrão que funciona da seguinte forma:

1. São geradas cópias dos vetores das mãos que guardam as cartas dos jogadores.
2. Envia os vetores das cópias das mãos dos jogadores para a função *strategy()* de cada jogador bem como o número de cartas na mão destes.
3. Recebe os índices das cartas a serem usadas pelos jogadores através do retorno das funções *strategy()* salvando nas variáveis *play1* e *play2*.
4. Verifica se os valores retornados são válidos, ou se o índice existe no vetor. Em caso de valor irregular, a carta selecionada pelo jogador que teve um mal retorno de função irá ser forçado a usar a primeira carta da mão.
5. Compara as cartas com a função *conflict* e salva o vencedor da rodada na variável *winnerround* (para poder ocorrer a animação), o perdedor na variável *looser* e a carta usada por este último na variável *loosercard*. Após isso, os pontos são contabilizados para o vencedor (este passo é um procedimento padronizado pelo arcabouço que necessita ser seguido para que ocorra a animação do jogo corretamente).

Os passos 1 até 4 podem ser modificados sem muitos problemas para adaptar a outros estilos de jogos. As únicas exigências são que:

- As variáveis *play1* e *play2* devem conter o índice (referente ao vetor das mãos dos jogadores) das cartas usadas pelos jogadores 1 e 2 respectivamente.
- A variável *winnerround* e *looser* devem conter o jogador que venceu e o que perdeu, respectivamente.
- A variável *loosercard* deve conter o índice da carta jogada pelo jogador que perdeu a rodada.
- A pontuação dos jogadores devem ser atualizadas.

O código abaixo representa a implementação padrão da função *roundfight()*.

Código 4.3 – Implementação padrão do Wanda da função *roundfight()*

```
function roundfight ()
```

```

copyhands()
if round<cardsindeck/2 then
    play1=jog1.strategy(hand1copy,hand2copy,
                        cardsindeck/2+1-round)
    copyhands()
    play2=jog2.strategy(hand2copy,hand1copy,
                        cardsindeck/2+1-round)
else
    play1=1
    play2=1
end

if type(play1)~="number" or play1<1 or
    play1>totalcards/2-round or play1%1>0 then
    play1=1
end
if type(play2)~="number"
    or play2==nil
    or play2<1 or play2>totalcards/2-round
    or play2%1>0 then

    play2=1
end
— Procedimento padronizado
— Nao recomenda-se modificar a partir
— daqui
winnerround=conflict(jogador1.mao[play1],
                    jogador2.mao[play2])
looser=0
if winnerround==1 then
    jogador1.partida=jogador1.partida+1
    looser=2
    loosercard=play2
end
if winnerround==2 then
    jogador2.partida=jogador2.partida+1
    looser=1
    loosercard=play1
end

```

```

        movement=true
end

```

4.3.4 Especificando os argumentos das estratégias (*strategy()*)

Esta etapa é a que requer menos linhas de código, mas é crucial, pois ela define a comunicação do aluno com o jogo criado através da função *strategy()*. Nessa etapa é necessário especificar uma tabela em Lua com duas funções. Uma função para estratégia e outra para retornar o nome do jogador (que deve ser chamada de “name”). Por fim deve-se retornar a tabela com as funções. Vale lembrar que tais funções fazem parte do objeto *Player*.

Por padrão, o arquivo com a função de estratégia a ser implementada pelo aluno vem na forma de uma tabela instanciada com o nome de *jog*. Nessa tabela são definidas duas funções: uma chamada de *strategy()* e outra chamada *name*. A função *name()* deve conter apenas um retorno de uma string com o nome do jogador. A *strategy()* recebe três argumentos:

- *cards*: um vetor que guarda as cartas da mão do jogador.
- *cardsoponent*: um vetor que guarda as cartas do oponente
- *numberofcards*: um número inteiro que guarda o número de cartas na mão do jogador.

O objetivo padrão da função *strategy()* é retornar o índice no vetor da carta da mão do jogador a ser usada pelo jogador.

Tendo em vista que a função que chama a *strategy()* (*roundfight()*) é também implementada pelo autor do jogo, torna-se necessário lembrar que os argumentos citados podem ser alterados os nomes, acrescentados novas variáveis ou mesmo utilizar estruturas de dados diferentes do padrão explicado. Tudo isso de acordo com a vontade e necessidade do autor da aplicação.

Código 4.4 – Definição padrão do Wanda para definição de estratégias

```

jog={}

function jog.strategy(cards, cardsoponent, numberofcards)
    --Estrategia, escreva abaixo
end

```



```
function jog.name()  
    return "Name"  
    — troque o –Name– pelo nome  
    — do seu avatar (maximo 10 caracteres)  
end  
  
return jog
```

5 *Jo-Ken-Po*

Este capítulo fala do primeiro jogo criado utilizando o arcabouço Wanda: *Jo-Ken-Po* (Figura 7). Na Seção 5.1, são abordadas as características gerais do jogo, explicando as regras e o funcionamento do mesmo. A Seção 5.2 explica como o jogo foi implementado utilizando o arcabouço. A experimentação do jogo é apresentada na Seção 5.3, explicando como o jogo foi utilizado.

5.1 Apresentação do Jogo

O jogo *Jo-Ken-Po* foi baseado no jogo Divide and Conquer (apresentado na Seção 6.1). A cada partida, um baralho contendo seis cartas é embaralhado e dividido aleatoriamente entre dois jogadores. O baralho consiste em duas cartas do tipo “papel”, duas do tipo “tesoura” e duas do tipo “pedra”. A duração de uma partida é de três rodadas.

Em cada rodada cada jogador deverá escolher jogar uma das cartas da sua mão ao mesmo tempo que o adversário. As cartas são comparadas seguindo a regra (como apresentado na Figura 8): tesoura ganha de papel, papel ganha de pedra, pedra ganha de tesoura e cartas iguais empatam. Após comparadas, o jogador que jogar uma carta a qual vence aquela jogada pelo oponente, marca um ponto. As cartas jogadas são removidas de jogo, e a partida termina quando não houverem mais cartas para serem jogadas pelos oponentes.

O jogador que conseguir mais vitórias em rodadas vence a partida. Caso ambos



Figura 7 – Exemplo de partida do jogo *Jo-Ken-Po*

jogadores marquem a mesma pontuação, o resultado é declarado como empate.

5.2 Instanciação do Wanda

Instanciar o jogo *Jo-Ken-Po* com o arcabouço Wanda é uma tarefa simples mesmo necessitando que o autor escreva poucas linhas de código.

O baralho de cartas do jogo é definido na variável “cards” da função *loadsetup()* por strings que representam os três tipos de cartas do jogo “pedra”, “papel” e “tesoura”, como mostra o trecho de código abaixo. Não é preciso implementar nada na função *loadimages()*, pois as imagens das cartas possuem os mesmos nome das que estas ilustram.

Código 5.1 – Inicialização do baralho do jogo *Jo-Ken-Po*

```
cards = { ‘pedra ’ , ‘pedra ’ , ‘papel ’ ,  
         ‘papel ’ , ‘tesoura ’ , ‘tesoura ’ }
```

A função *conflict()* consiste em apenas comparar duas strings que representam as cartas usadas por cada jogador. Se as cartas forem iguais a função retorna empate (valor “0”). Se forem diferentes as strings são comparadas de acordo com as regras do jogo (representada na Figura 8) e retorna “1” caso a string que represente a carta vencedora seja aquela que representa a carta usada pelo primeiro jogador, ou “2” caso contrário.

Cada carta em jogo é enviada como argumento da função *strategy()* de modo que o aluno possa verificar as cartas em jogo checando cada variável. Em outras palavras, existem três argumentos chamados de “carta1”, “carta2”, e “carta3” que representam as cartas na mão do jogador, e “carta1oponente”, “carta2oponente” e “carta3oponente” para representar as cartas na mão do oponente, cada uma dessas variáveis possui uma string “pedra”, “papel” ou “tesoura” para representar uma carta cada. Esta função deve

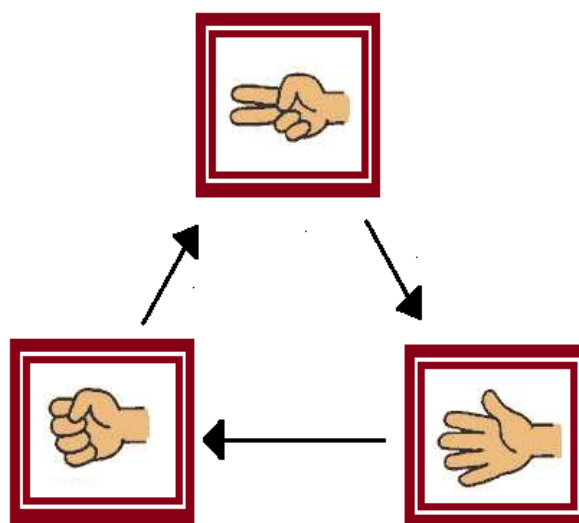


Figura 8 – Diagrama de vitória das cartas do jogo *Jo-Ken-Po*

retornar uma das strings “pedra”, “papel” ou “tesoura” para indicar qual carta foi usada pelo jogador.

A função *roundfight()* segue a mesma implementação padrão, com apenas algumas diferenças. A primeira diferença se refere ao fato de que como a função *strategy()* retorna uma string, a função *roundfight()* irá guardar nas variáveis “play1” e “play2” os endereços no vetor da mão do jogador onde existem aquelas cartas. Caso a função *strategy()* retorne algo que não seja uma string ou uma carta que o jogador não possui, a carta a ser jogada será a primeira do vetor da mão daquele jogador.

Implementar o jogo *Jo-Ken-Po* necessitou de menos de 50 linhas de código.

5.3 Experimentação

Uma turma de alunos do primeiro período de Ciência da Computação da Universidade Federal do Maranhão recebeu o jogo *Jo-Ken-Po*. Os alunos apenas receberam aulas de programação sobre execução de comandos de atribuição e de condição: *se (if) e se-não(else)*. Eles também receberam instruções de como fazer estas operações em linguagem de programação Lua.

Os alunos receberam instruções sobre como o jogo funcionava e o que estes deveriam fazer para criar o agente para jogá-lo. Os alunos formaram duplas e cada dupla foi instruída para criar um agente. Uma competição (retratada na Figura 9) foi organizada em seguida com as duplas, onde cada uma submeteu o agente criado por elas.

Esta atividade tem como objetivo exercitar a capacidade de programação dos alunos com foco nas operações de condição e atribuição. Outro objetivo é o de dar noções da utilização de programação e seu funcionamento de maneira visual para o aluno.



Figura 9 – Campeonato do jogo *Jo-Ken-Po*

Antes da competição foi aplicado um questionário para medir a motivação (descrito no Capítulo 7) para saber como estava a motivação dos alunos. O mesmo questionário foi repetido para a turma logo após a competição e os resultados foram contrastados. A maioria dos alunos conseguiram implementar o agente.

Os resultados e análise dos resultados do questionário serão discutidos no Capítulo 7. Uma implementação de uma estratégia para o jogo *Jo-Ken-Po*, elaborada por um dos alunos deste experimento, encontra-se no Anexo B deste trabalho.

6 Divide-and-Conquer

Este capítulo fala do segundo jogo criado utilizando o arcabouço Wanda: *Divide-and-Conquer* (apresentado na Figura 10). Na Seção 6.1, são abordadas as características gerais do jogo, explicando as regras e o funcionamento do mesmo. A Seção 6.2 explica como o jogo foi implementado utilizando o framework. A experimentação do jogo é apresentada na Seção 6.3, explicando como o jogo foi utilizado.

6.1 Apresentação do Jogo

O jogo *Divide-and-Conquer* (CELKO; MCLEOD, 2014) possui um baralho de dez cartas. Nove delas são enumeradas de 2 até 10 e uma carta com o número 12. No início da partida as cartas são embaralhadas e distribuídas aleatoriamente entre dois jogadores. Cada partida dura 5 rodadas.

Em cada rodada ambos jogadores devem jogar uma carta da mão ao mesmo tempo. As cartas são comparadas seguindo a seguinte regra (apresentada na Figura 11): A carta com maior valor ganha somente se a outra não representar um número divisor ou antecessor do maior número jogado, caso contrário, a com menor valor ganha. Após comparadas o jogador que jogar uma carta a qual vence aquela jogada pelo oponente, marca um ponto. As cartas jogadas são removidas de jogo, e a partida termina quando não houverem mais cartas para serem jogadas pelos oponentes.

O jogador que conseguir mais vitórias em rodadas vence a partida. Não existe

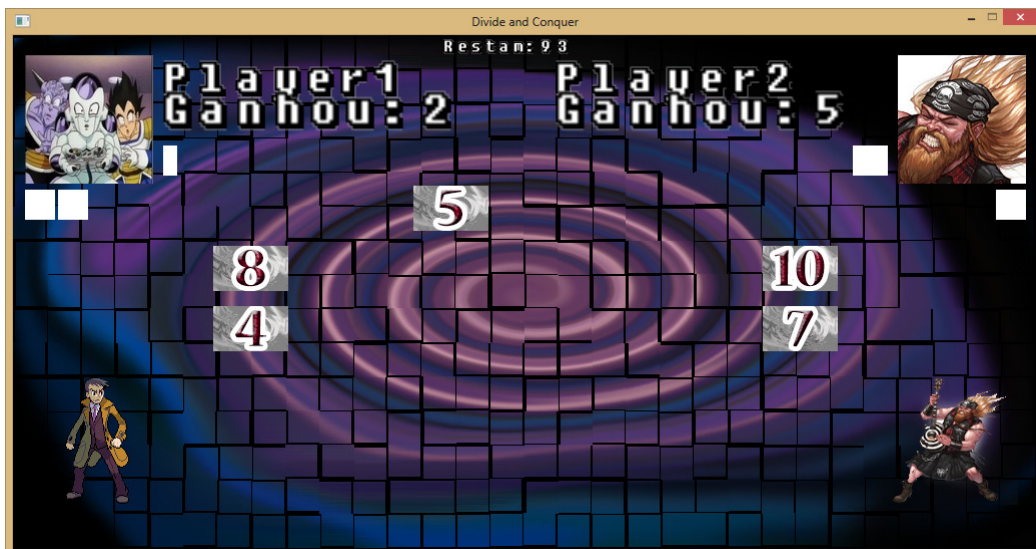


Figura 10 – Exemplo de partida do jogo *Divide and Conquer*

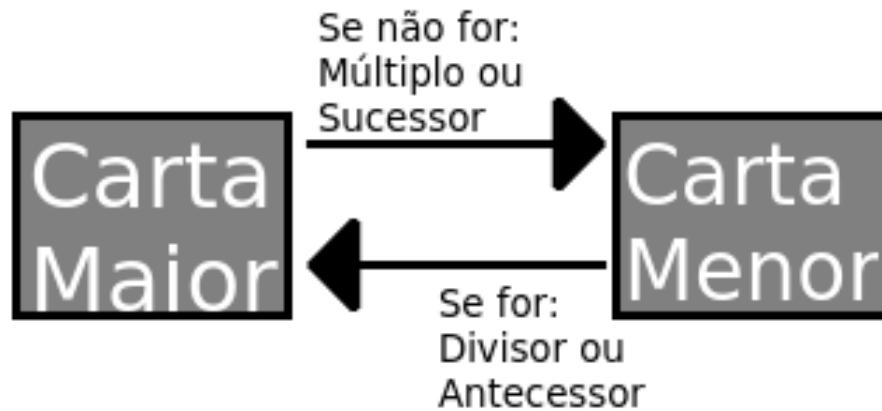


Figura 11 – Diagrama de vitória das cartas do jogo *Divide and Conquer*

empate no resultado da partida ou em uma rodada.

6.2 Instanciação do Wanda

O jogo *Divide-and-Conquer* segue fielmente ao modelo padrão do Wanda e não requer muitas alterações.

A função *conflict()* se resume em comparar dois números (que equivalem a duas cartas jogadas por cada jogador). Se o maior dos números não for múltiplo ou sucessor do menor, esta carta será considerada vitoriosa, caso contrário a outra é definida vencedora. A função então retorna “1” se a carta do primeiro jogador for a vencedora, ou “2” caso contrário. Não há possibilidade de empate.

Código 6.1 – Função de conflito do jogo *Divide-and-Conquer*

```
function divisivel(x,y)
  if x % y == 0 then return true
  else return false end
end
function conflict(p1,p2)
  if p1>p2 then
    if divisivel(p1,p2) or (p2+1==p1) then
      return 2
    else
      return 1
    end
  else
    if divisivel(p2,p1) or (p1+1==p2) then
      return 1
    else
```

```
                return 2
            end
        end
    end
    return 0
end
```

As cartas são carregadas na função *loadsetup()* através de uma atribuição de um vetor de números inteiros de 2 a 10 e mais o número 12.

Código 6.2 – Inicializando o baralho para o jogo Divide-And-Conquer

```
cards = {2,3,4,5,6,7,8,9,10,12}
```

As funções *roundfight()* e *strategy()* não sofreram alterações seguindo assim o modelo padrão que já vem pré-programado no arcabouço. Dessa forma a função que executa a estratégia programada pelo jogador irá receber as cartas em jogo em forma de dois vetores (um para representar a mão do jogador e de seu oponente) necessitando que seja retornado a posição da carta que o jogador deseja usar.

6.3 Experimentação

A mesma turma de alunos que recebeu o jogo Jo-Ken-Po (descrito no Capítulo 5), recebeu o jogo Divide-and-Conquer. Neste momento os alunos além de saberem comandos de atribuição e de condição, aprenderam também como realizar operações de laço (*para (for)* bem como funcionamento de vetores ou *arrays*. Eles também receberam instruções de como fazer estas operações em linguagem de programação Lua.

Os alunos receberam instruções sobre como o jogo funcionava e o que eles deveriam fazer para criar o agente para jogá-lo. Os alunos formaram duplas e cada dupla foi instruída para criar um agente. Uma competição foi organizada em seguida com as duplas, onde cada uma submeteu o agente criado por elas.

Esta atividade busca exercitar a capacidade de programação dos alunos com foco nas operações com laços e vetores. Outro objetivo é mostrar para os alunos o funcionamento do código de maneira visual e a utilidade do conteúdo de programação aprendido, assim como na atividade *Jo-Ken-Po*. A competição ocorreu após ser aplicado o segundo questionário sobre motivação (Capítulo 7), após a atividade *Jo-Ken-Po* (Capítulo 5).

O mesmo questionário foi repetido para a turma logo após a competição do jogo Divide-And-Conquer e os resultados foram contrastados com os anteriores. Diversos alunos conseguiram implementar o agente. Os resultados e análise dos resultados do questionário serão discutidos no Capítulo 7. Uma implementação de uma estratégia para o jogo *Divide-and-Conquer*, elaborada por um dos alunos deste experimento, encontra-se no Anexo B deste trabalho.

7 Questionário para Medir Motivação

Nos Capítulos 5 e 6, foram apresentados dois estudos de caso que consistem na utilização de dois jogos criados com o arcabouço apresentado neste trabalho. Para medir o impacto na motivação dos alunos causado pela utilização dos jogos foi aplicado um questionário para medir a motivação dos jogadores em diferentes momentos.

Antes da atividade com o jogo Jo-Ken-Po (descrita no Capítulo 5) foi aplicado o questionário. Após o término da atividade, outro questionário foi aplicado para poder contrastar os resultados. Após isso, a atividade aplicada com o jogo Divide-and-Conquer (descrita no Capítulo 6) foi executada, seguido de da terceira e última aplicação do questionário. Foram entrevistados 35 alunos, os quais participaram de todas as atividades.

Este capítulo descreve e explica o questionário utilizado na Seção 7.1 e analisa os resultados de sua aplicação na Seção 7.2.

7.1 Descrição do Questionário

O questionário tratado neste capítulo foi baseado em um existente para medir a motivação de alunos de ciências (TUAN, 2005). Este questionário possui diversas perguntas que foram adaptadas quanto a temática para o aprendizado da matéria de introdução a computação.

Tuan, analisa em seu trabalho seis fatores diferentes, que são medidos através das perguntas criadas por ele, são eles:

1. Auto-Eficiência: O aluno acredita em sua capacidade em ir bem na matéria.
2. Estratégias ativas de aprendizado: O aluno busca diferentes métodos para desenvolver novos conhecimentos.
3. Valor do aprendizado da matéria: O valor do aprendizado é permitir que os alunos adquiram competência para resolver problemas, questionar, estimular o raciocínio e relacionar os estudos com problemas do dia-a-dia.
4. Objetivo de Desempenho: O objetivo dos alunos é competir entre si pela atenção do professor.
5. Objetivo de Conquista: Os alunos sentem satisfação enquanto eles ficam mais competentes.

6. Estimulo por Ambiente de Aprendizado: Os elementos presentes no ambiente de aprendizado, como interação do aluno com o instrutor, o ensino do professor e outros elementos influenciam a motivação dos alunos.

Destes itens os pontos mais relevantes para este trabalho são os os itens 1, 3, 5 e 6, pois estão relacionados diretamente com o problema e portanto serão os pontos aos quais será dado atenção.

Cada fator possui perguntas relacionadas a eles e todas elas constituem afirmações sobre o aluno, o qual deve responder em uma escala de 1 a 5, onde 1 representa discordância total sobre o assunto e 5 concordância total.

Além das perguntas adaptadas da proposta de Tuan, este trabalho utiliza perguntas diretas ao aluno sobre as atividades em que ele participou (utilizando a mesma escala como resposta) e uma questão dissertativa onde o aluno deve escrever sobre o que ele achou das atividades.

O questionário de motivação está disponível no Apêndice A deste trabalho.

7.2 Análise de Resultados do Questionário

Os resultados dos três questionários aplicados foram comparados para ver o progresso geral da turma. É importante lembrar que apenas alguns poucos alunos (cerca de 10%) possuíam conhecimentos prévios de programação ou experiência com a área. As subseções a seguir analisam os fatores mais relevantes do questionário em relação a este trabalho e comparam entre si.

Cada subseção apresentará um gráfico sobre uma pergunta que representa o fator em questão. Neste gráfico são exibidos os percentuais de alunos que escolheram cada opção de resposta em cada um dos três questionário. As respostas podem ser um número na escala de 1 a 5, onde 1 representa que o aluno discorda completamente do enunciado da pergunta, e 5 ele concorda plenamente.

7.2.1 Auto-Eficiência

Analisando as respostas dadas, pode-se perceber que houve uma leve queda na confiança dos alunos entre a primeira e segunda aplicações do formulário. Contudo houve uma leve recuperação na terceira etapa. Um dos motivos prováveis nesta queda de confiança está ligado a fatores externos. O desempenho da turma foi abaixo da média na prova da disciplina de Algoritmos aplicada anteriormente ao segundo questionário e à atividade *Jo-Ken-Po*, o que pode ter sido uma das causas para tal mudança na confiança. O crescimento da confiança no terceiro questionário pode ter sido resultado da atividade com o jogo

Divide-and-Conquer visto que não houve quaisquer atividades ou avaliações entre os dois questionários, senão o próprio jogo. Outro fator importante a ser considerado é que o primeiro formulário foi aplicado no início da disciplina e os alunos ainda não tinham tido muitas aulas e atividades. Um exemplo que representa essa mudança está no comparativo de respostas da Figura 12.

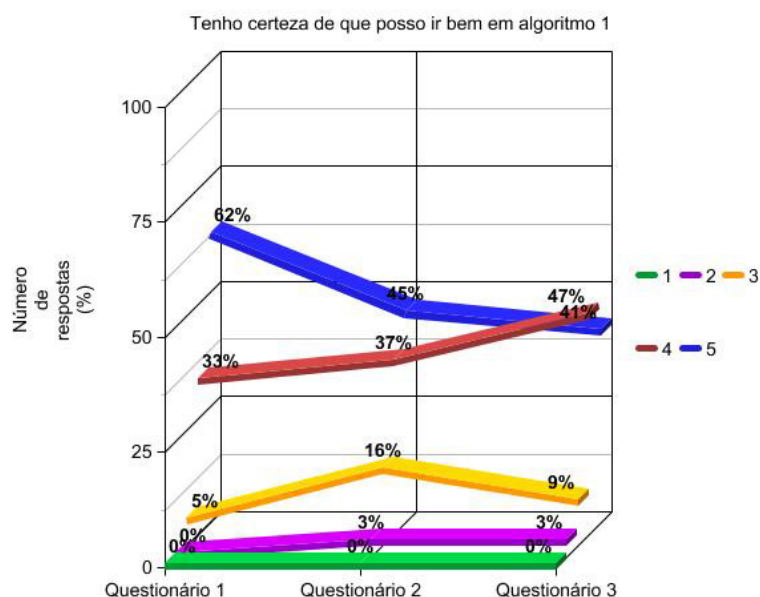


Figura 12 – Gráfico comparativo de respostas de uma pergunta sobre Auto-Eficiência

7.2.2 Valor do aprendizado da matéria

Nas perguntas que se referem ao entendimento da importância da matéria (ou se o aluno enxerga o quão importante é aquela matéria para o curso), os alunos mostraram uma variação em relação ao primeiro resultado. Alguns alunos começaram a enxergar o assunto de algoritmos como menos importante. O terceiro resultado contudo, mostrou um aumento nesse aspecto. O que sugere que talvez a ideia de expor o funcionamento de um algoritmo de maneira gráfica pode ajudar a fazer o aluno a entender a importância do assunto. Um exemplo disso está representado na Figura 13

7.2.3 Objetivo de Conquista

A satisfação em aprender novos conteúdos iniciou bem alta durante o início do curso. Notou-se uma baixa discreta nesse aspecto após a aplicação do jogo *Jo-Ken-Po* e um crescimento da mesma após a atividade com o jogo *Divide-and-Conquer*. A Figura 14 exemplifica essas mudanças. O professor da turma associou a baixa com o desempenho de uma atividade avaliativa ocorrida após o jogo *Jo-Ken-Po*. A mudança contudo não foi grande o suficiente para se concluir que a turma tornou-se desinteressada em aprender novos conhecimentos.

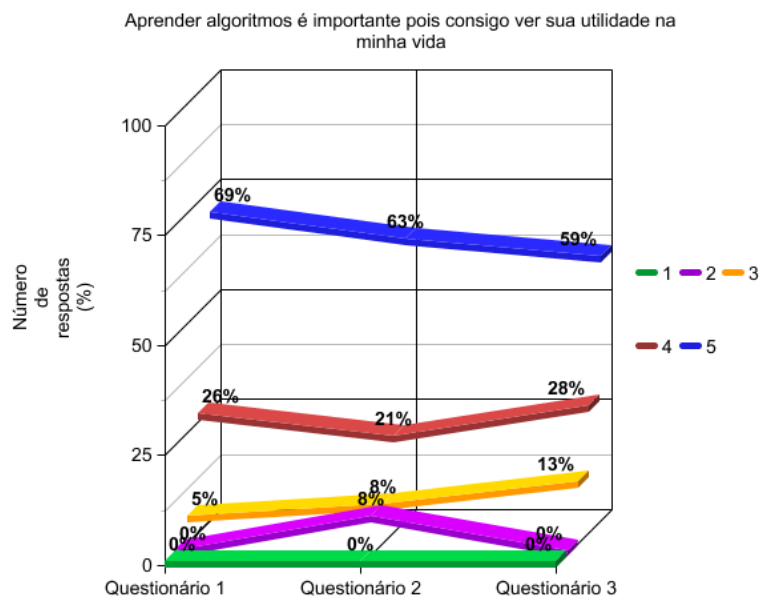


Figura 13 – Gráfico comparativo de respostas de uma pergunta sobre Valor do aprendizado da matéria

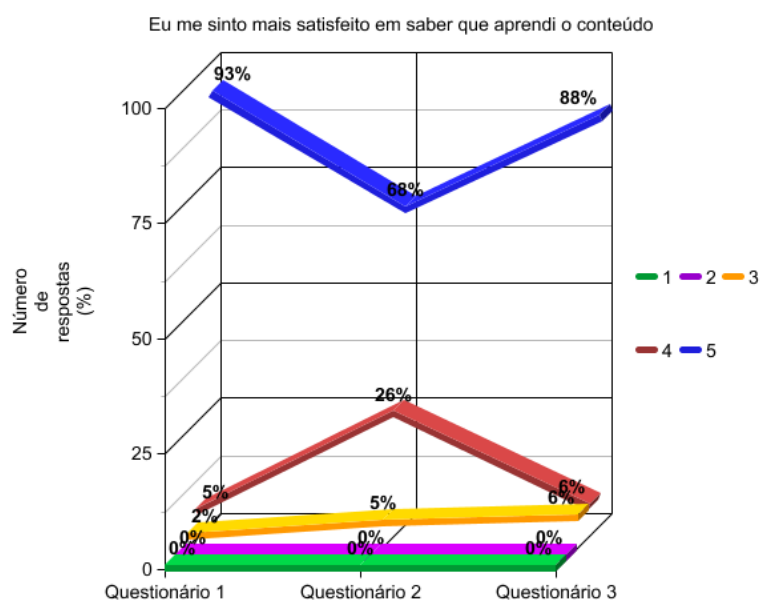


Figura 14 – Gráfico comparativo de respostas de uma pergunta sobre Objetivo de Conquista

7.2.4 Estímulo por Ambiente de Aprendizado

A motivação do aluno para estudar a disciplina de Algoritmos mostrou crescimento da resposta “5” no segundo questionário e um aumento menos acentuado da resposta “2”. Estes resultados melhoraram no terceiro questionário onde houve uma redução nas respostas negativas (“1” e “2”) e um aumento nas positivas (“4” e “5”) acusando uma motivação final bem melhorada.

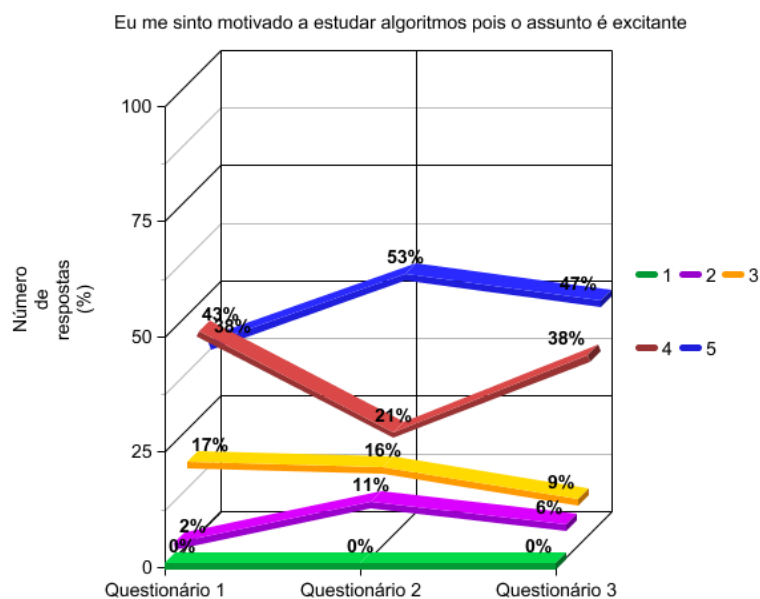


Figura 15 – Comparativo gráfico de respostas de uma pergunta sobre Estimulo por Ambiente de Aprendizado

Como descrito na sub-seção anterior, um motivo plausível para a primeira queda pode ter sido os resultados da primeira avaliação aliado aos primeiros contatos com a disciplina. Após o Divide-and-Conquer, o lado negativo diminuiu e o positivo aumentou mostrando um bom resultado nesse aspecto. Como exibido na Figura 15.

Como mostrado na pergunta da Figura 16. Os alunos se mostraram indiferentes quanto a utilização de métodos diferentes de ensino. Houve uma melhora desde o primeiro

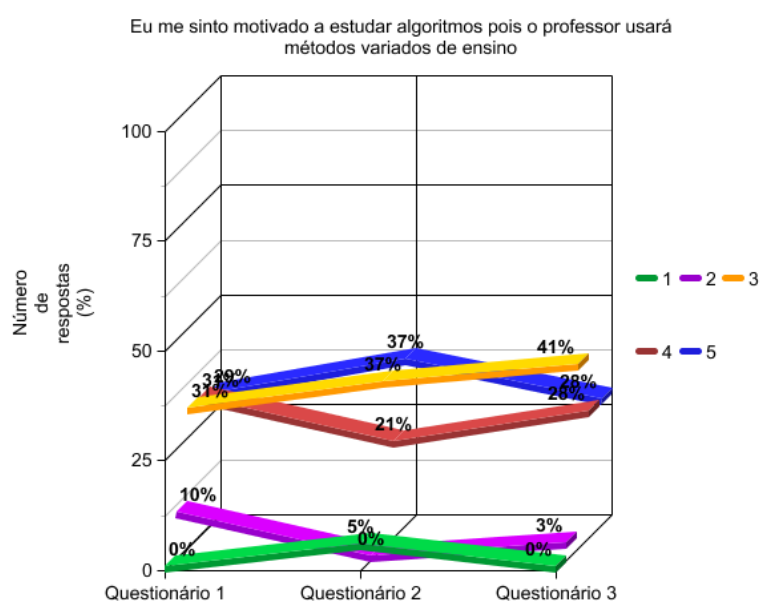


Figura 16 – Gráfico comparativo de respostas de uma pergunta sobre Estimulo por Ambiente de Aprendizado

questionário. Contudo, este resultado apenas indica que os alunos não estão conscientes se há mais motivação ou não por serem usados outros métodos de avaliação dentro da sala de aula.

Este quesito foi visível entre os alunos através do comportamento e entusiasmo que era claro durante o momento da competição. O momento também foi propício a questionamentos e maior interação entre alunos, monitores e professor.

8 Conclusão

Com este trabalho é possível concluir que Wanda é uma ferramenta útil para a criação de jogos de cartas eletrônicos voltados para a educação, processo que leva pouco tempo e esforço com a ferramenta. Esta facilidade é graças ao fato de que o arcabouço já possui implementado boa parte das funções da mecânica do jogo. Onde assim, o autor da aplicação deve entrar apenas com um pedaço de código muito pequeno e formatar arquivos para receber as estratégias de alunos. As estratégias são escritas em formato de código e irão ser responsáveis por realizar as jogadas pelo jogador.

Wanda alcança todos os objetivos estabelecidos, pois: os jogos seguem um esquema específico de jogos de cartas; as aplicações necessitam de poucas linhas de código para serem criadas, facilitando a criação e economizando tempo; os alunos podem submeter suas estratégias em forma de código (e os autores podem customizar o formato do arquivo); possui recursos gráficos para exibir o desenrolar do jogo; os jogos criados com Wanda ficam com código aberto. Dessa forma, Wanda atende a todos os requisitos e supre as necessidades descritas na comparação de trabalhos do Capítulo 3 (Seção 3.7).

Para medir a eficácia do arcabouço, dois jogos foram criados: *Jo-Ken-Po* e *Divide-and-Conquer*. Ambos seguem regras semelhantes, mas possibilitam exercitar conhecimentos diferentes sobre algoritmos. Alunos egressos no curso de Ciência da Computação receberam os jogos em diferentes momentos de acordo com os conhecimentos necessários para jogá-los. Antes de cada jogo ser aplicado, um questionário para medir motivação foi aplicado aos alunos, o mesmo foi aplicado após as atividades com os jogos. Os resultados dos questionários foram analisados e comparados para medir os efeitos dos jogos.

Notou-se, pela comparação dos resultados, que houve uma melhora na motivação dos alunos bem como na confiança destes para com o seu desempenho no curso. A opinião emitida pelos alunos mostrou que a maioria estava satisfeita com a atividade. Isso possibilita concluir que Wanda e seus aplicativos foram avaliados positivamente.

Outra contribuição deste trabalho é mostrar como a utilização de jogos pode ser útil para o ensino de conceitos de programação. Jogos podem ajudar também a motivar os alunos a estudarem e possivelmente diminuir a taxa elevada de evasão dos cursos da área de programação (TAKAHASHI, 2009), já que a dificuldade de programar é um das mais apontadas para o problema (KINNUNEN; MALMI, 2006) além ser uma estratégia intrínseca, ideal para motivar alunos (HUITT, 2011).

Outro fruto deste trabalho foi uma publicação de artigo completo (DRUMOND; DAMASCENO; SOARES NETO, 2014) na trilha de computação do XIII Simpósio Brasileiro de Jogos e Entretenimento Digital.

Wanda, contudo, necessita de mais funcionalidades para que esta possa ser mais abrangente e genérica de modo que com esta ferramenta seja possível instanciar aplicações com uma gama maior de variedade.

Como trabalhos futuros, devem ser implementadas novas instanciações de aplicações mais complexas e mais distantes do padrão do arcabouço. Também deve-se enriquecer a documentação sobre Wanda, criando um tutorial sobre como instanciar uma aplicação usando a ferramenta. Dessa forma podem ser estudadas mudanças na ferramenta para que se possa atender a novas funcionalidades.

Referências

- BALASUBRAMANIAN, N.; WILSON, B. G. Games and simulations. In: *Society for Information Technology and teacher Education International Conference*. [S.l.: s.n.], 2006. Citado na página 18.
- BAYLISS, J. D.; STROUT, S. Games as a "flavor" of cs1. *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, v. 37, p. 500–504, 2006. Citado na página 15.
- BURBECK, S. Applications programming in smalltalk-80 (tm): How to use model-view-controller (mvc). *Smalltalk-80 v2*, v. 5, 1992. Citado na página 31.
- CAVALCANTE, D. C. Cades: Um arcabouço para a implementação de jogos de cartas. *Trabalho de conclusão de curso. Universidade Federal do Maranhão. São Luís, MA*, 2006. Citado 2 vezes nas páginas 19 e 25.
- CELKO, J.; MCLEOD, J. *Divide and Conquer*. 2014. <<http://www.pagat.com/tile/wdom/divide.html>>. Acesso em 20 de junho de 2014. Citado 2 vezes nas páginas 25 e 45.
- CELSO JR., L. *Setor de TI aponta falta de mão-de-obra qualificada*. 2009. <<http://bit.ly/bQnsOg>>. Acesso em 20 de junho de 2014. Citado na página 15.
- DRUMOND, R. R.; DAMASCENO, A. L. B.; SOARES NETO, C. S. Wanda: a framework to develop card based games to help motivate programming students. *Anais Simpósio Brasileiro de Jogos e Entretenimento Digital*, 2014. Citado 2 vezes nas páginas 15 e 54.
- GROS, B. The impact of digital games in education. *First Monday*, v. 8, n. 7, p. 6–26, 2003. Citado na página 18.
- HUITT, W. *Motivation to learn: An overview*. 2011. <<http://www.edpsycinteractive.org/topics/motivation/motivate.html>>. Acesso em 20 de junho de 2014. Citado 2 vezes nas páginas 15 e 54.
- IERUSALIMSCHY, R.; WALDEMAR, C.; FIGUEIREDO, L. *Lua: the Programming Language*. 2014. <<http://www.lua.org/>>. Acesso em 20 de junho de 2014. Citado na página 30.
- KESSLER, M. E. A. Impulsionando a aprendizagem na universidade por meio de jogos educativos digitais. *Anais do Simpósio Brasileiro de Informática na Educação*, 2010. Citado na página 21.
- KINNUNEN, P.; MALMI, L. Why students drop out cs1 course? *Proceedings of the second international workshop on Computing education research*, p. 97 – 108, 2006. Citado 2 vezes nas páginas 15 e 54.
- KOHWALTER, T.; CLUA, E.; MURTA, L. Sdm – an educational game for software engineering. *Anais Simpósio Brasileiro de Jogos e Entretenimento Digital*, 2011. Citado na página 22.

- LÖVE-2D. 2014. <<https://love2d.org>>. Acesso em 20 de junho de 2014. Citado 2 vezes nas páginas 16 e 30.
- MARKETING-CHARTS. *VGs to Surpass Music in Revenue This Year*. 2008. <<http://www.marketingcharts.com/interactive/pwc-videogames-tosurpass-music-in-revenue-this-year-750/>>. Acesso em 20 de junho de 2014. Citado na página 18.
- NOVAK, J. *Desenvolvimento de Games—Tradução da 2ª Edição norte-americana*. [S.l.]: CENGAGE Learning, 2007. Citado na página 18.
- RADTKE, P.; BINDER, F. Chien 2d: A multiplatform library to teach the c language through games programming. *Anais Simpósio Brasileiro de Jogos e Entretenimento Digital*, 2010. Citado na página 22.
- REBOUÇAS, A. E. A. Aprendendo a ensinar programação combinando jogos e python. *Anais Simpósio Brasileiro de Jogos e Entretenimento Digital*, 2010. Citado na página 23.
- SAVI, R.; ULBRICHT, V. R. Jogos digitais educacionais: benefícios e desafios. *RENOTE*, v. 6, n. 1, 2008. Citado na página 18.
- SEBRAE. *Brasil tem o maior mercado de games no mundo em 2012*. 2008. <<http://www.sebrae2014.com.br/Sebrae2014/Alertas/Brasil-tem-o-maior-mercado-de-games-no-mundo-em-2012>>. Acesso em 20 de junho de 2014. Citado na página 18.
- SHINERS, P. *PyGame Wiki*. 2014. <<http://www.pygame.org/wiki/>>. Acesso em 20 de junho de 2014. Citado 2 vezes nas páginas 23 e 30.
- SOUZA M.M., E. A. Sparse: Um ambiente de ensino e aprendizado de engenharia de software baseado em jogos e simulação. *Anais do Simpósio Brasileiro de Informática na Educação*, 2010. Citado na página 24.
- TAKAHASHI, F. *Matemática e ciências da computação têm alta taxa de abandono*. 2009. <<http://bit.ly/daPZUm>>. Acesso em 20 de junho de 2014. Citado 2 vezes nas páginas 15 e 54.
- TROIS, J. *Guia prático dos jogos de cartas*. [S.l.]: Clube de Autores, 2012. Citado na página 19.
- TUAN, H. L. The development of a questionnaire to measure students motivations towards science learning. *International Journal of Science*, p. 639–654, 2005. Citado na página 48.
- UNITY-TECHOLOGIES. *Unity Game Engine*. 2014. <<http://unity3d.com>>. Acesso em 20 de junho de 2014. Citado 2 vezes nas páginas 22 e 30.
- WOPC. *The World of Playing Cards*. 2014. <www.wopc.co.uk/>. Acesso em 20 de junho de 2014. Citado 2 vezes nas páginas 19 e 20.

ANEXO A – Questionário de Motivação

- Não importa se o conteúdo sobre algoritmos é difícil, tenho certeza que consigo entender.
- Eu não me sinto capaz de entender conceitos importantes sobre algoritmos.
- Tenho certeza de que posso ir bem em algoritmo 1
- Não importa quanto esforço eu coloque, eu não consigo entender algoritmos.
- Quando uma tarefa é muito difícil eu tento fazer, eu desisto e faço apenas as partes fáceis.
- Durante as atividades eu prefiro perguntar para colegas de turma ao invés de pensar sozinho.
- Quando eu acho o conteúdo difícil eu não tento aprendê-lo
- Quando estou aprendendo novos conceitos eu tento entendê-los.
- Quando estou aprendendo algo novo eu tento conectá-lo com conhecimentos anteriores
- Quando não entendo um conceito novo, eu procuro sobre ele em fontes relevantes que possam me ajudar
- Durante meu aprendizado eu tento conectar novos conceitos com algum que eu já conheço.
- Quando não entendo um conceito, eu debato sobre este com meu professor e colegas.
- Quando cometo um erro, eu tento entender o porque
- Ainda que eu não entenda um conceito novo, eu tento aprendê-lo mesmo assim
- Eu sempre tento entender conceitos novos que conflitam com conceitos aprendidos anteriormente.
- Eu acho que aprender algoritmos é importante pois consigo ver sua utilidade na minha vida
- Eu acho que aprender algoritmos é importante pois estimula meu pensamento
- Em algoritmos, é muito importante resolver problemas
- Eu me sinto mais satisfeito em saber que aprendi o conteúdo

- Eu me sinto mais satisfeito quando consigo resolver um problema difícil.
- Eu me sinto motivado a estudar algoritmos pois o assunto é excitante.
- Eu me sinto motivado a estudar algoritmos pois o professor usará métodos variados de ensino.
- Eu me sinto motivado a estudar algoritmos pois acho que não haverá pressão sobre mim.
- Eu me sinto motivado a estudar algoritmos pois o assunto é desafiador.
- Eu me sinto motivado a participar das aulas pois os outros alunos estarão envolvidos em discussões.

O segundo questionário aplicado (aplicado após a atividade Jo-Ken-Po), possui estas perguntas:

- Eu gostei da atividade Jo-Ken-Po
- A atividade Jo-Ken-Po me motivou a estudar um pouco mais
- Escreva um pouco sobre o que você achou da atividade (dissertativa)

O terceiro questionário aplicado (aplicado após a atividade Divide-and-Conquer), possui estas perguntas:

- Eu gostei da atividade Divide-and-Conquer
- A atividade Divide-and-Conquer me motivou a estudar um pouco mais
- Escreva um pouco sobre o que você achou da atividade (dissertativa)

ANEXO B – Exemplos de Estratégias

B.1 Exemplo de estratégia para o jogo Jo-Ken-Po

Código B.1 – Estratégia Jo-Ken-Po

```

jo1={}
function jo1.player1round1(cartas1 ,cartas2 ,cartas3)
  --Estrategia 1 lo round escreva abaixo
  local tesouras = 0
    if cartas1 == "tesoura" then
      tesouras=tesouras + 1
    end

    if cartas2 == "tesoura" then
      tesouras=tesouras + 1
    end

    if cartas3 == "tesoura" then
      tesouras=tesouras + 1
    end

  local pedras = 0
    if cartas1 == "pedra" then
      pedras=pedras + 1
    end

    if cartas2 == "pedra" then
      pedras=pedras + 1
    end

    if cartas3 == "pedra" then
      pedras=pedras + 1
    end

  local papeis = 0
    if cartas1 == "papel" then
      papeis=papeis + 1
    end

```

```
end

if carta2 == "papel" then
    papeis=papeis + 1
end

if carta3 == "papel" then
    papeis=papeis + 1
end

if papeis > 0 then
    return "papel"
else
    return carta1
end

if tesouras > 1 then
    return "tesoura"
else
    return carta2
end

if pedras > 0 then
    return "papel"
else
    return carta3
end
-- fim
end

function jo1.player1round2(carta1, carta2, oponente1, oponente2)
    --Estrategia 1 2o round escreva abaixo
    local tesouras = 0
    if carta1 == "tesoura" then
        tesouras=tesouras + 1
    end

    if carta2 == "tesoura" then
```

```
        tesouras=tesouras + 1
    end

    if carta3 == "tesoura" then
        tesouras=tesouras + 1
    end

local pedras = 0
    if carta1 == "pedra" then
        pedras=pedras + 1
    end

    if carta2 == "pedra" then
        pedras=pedras + 1
    end

    if carta3 == "pedra" then
        pedras=pedras + 1
    end

local papeis = 0
    if carta1 == "papel" then
        papeis=papeis + 1
    end

    if carta2 == "papel" then
        papeis=papeis + 1
    end

    if carta3 == "papel" then
        papeis=papeis + 1
    end

    if papeis > 1 then
        return "tesoura"
    else
        return carta1
    end
end
```

```
    if papeis > 0 then
        return "papel"
    else
        return carta2
    end

    if tesouras > 0 then
        return "papel"
    else
        return carta3
    end
    -- fim
end

function jog1.player1name()
    return "Aiolia" -- troque o Player 1 pelo
                    -- nome do seu lutador
                    -- (maximo 10 caracteres)
end

return jog1
```

B.2 Exemplo de estratégia para o jogo Divide-and-Conquer

Código B.2 – Estratégia Divide-and-Conquer

```
jog={}
```

```
function jog.estrategia(cartas, cartasoponente, numerodecartas)
    --Estrategia 1 1o round escreva abaixo

    for c2 = 1,5 do
        if cartas[c2] == 2 then
            break
        end
    end

    end

    for c3 = 1,5 do
        if cartas[c3] == 3 then
            break
        end
    end
```



```
end
for c4 = 1,5 do
    if cartas[c4] == 4 then
        break
    end
end
for c5 = 1,5 do
    if cartas[c5] == 5 then
        break
    end
end
for c6 = 1,5 do
    if cartas[c6] == 6 then
        break
    end
end
for c7 = 1,5 do
    if cartas[c7] == 7 then
        break
    end
end
for c8 = 1,5 do
    if cartas[c8] == 8 then
        break
    end
end
for c9 = 1,5 do
    if cartas[c9] == 9 then
        break
    end
end
for c10 = 1,5 do
    if cartas[c10] == 10 then
        break
    end
end
for c12 = 1,5 do
    if cartas[c12] == 12 then
        break
    end
end
```

```
        end
    end

    if c2 ≈= nil then
        return c2--2
    end

    if c3 ≈= nil and c2 ≈= nil then
        return c3--3
    end

    if c4 ≈= nil then
        return c4
    end

    if c5 ≈= nil and c4 ≈= nil then
        return c5--5
    end

    if c6 ≈= nil and c5 == nil and c12 ≈= nil then
        return c6
    end

    if c7 ≈= nil and c6 ≈= nil and c8 ≈= nil and
        c10 ≈= nil and c12 ≈= nil then

        return c7--7
    end

    if c8 ≈= nil and c9 == nil and c4 ≈= nil and
        c7 ≈= nil then

        return c8--8
    end

    if c9 ≈= nil and c3 ≈= nil and c10 ≈= nil and
        c12 ≈= nil then
        return c9--9
    end

end
```

```
        if c10 ~= nil and c5 ~= nil and c9 ~= nil and
            c12 ~= nil then

                return c10--10
            end
        end

end

function jog.name()
    return "Obelix" -- troque o Player 1 pelo
                    -- nome do seu lutador
                    -- (maximo 10 caracteres)
end

return jog
```