UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM AMBIENTE 2D NA ÁREA DE ENTRETENIMENTO, UTILIZANDO RECURSOS MULTIMÍDIA

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

FRANCISCO REGO BELTRÃO

BLUMENAU, MARÇO/2000

2000/1-24

PROTÓTIPO DE UM AMBIENTE 2D NA ÁREA DE ENTRETENIMENTO, UTILIZANDO RECURSOS MULTIMÍDIA

FRANCISCO REGO BELTRÃO

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Dalton Solano dos Reis — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Dalton Solano dos Reis

_____

Prof. Roberto Heinzle

_____

Prof. Maurício Capobianco Lopes

AGRADECIMENTOS

Ao meu orientador, prof. Dalton Solano dos Reis, pela atenção dispensada a este trabalho.

A minha namorada e aos meus colegas pela amizade, apoio e colaboração. As pessoas que tiveram força de vontade para testar meu protótipo.

As bandas Iron Maiden, Kiss, Millencolin, Pennywise, No use for a name, entre outras, que não me deixaram dormir durante a madrugada.

A minha família que mesmo longe, sempre me deu apoio e amor, e são as pessoas mais importantes em minha vida.

As outras pessoas que posso ter esquecido, vocês sabem quem são.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho consiste num estudo sobre o desenvolvimento de software na área de entretenimento e modelagem 2D. Seu objetivo é a especificação e implementação de um protótipo de um combate espacial, utilizando também técnicas multimídia. Ele poderá ser jogado por vários jogadores, distribuídos em uma rede, simultaneamente. A plataforma de desenvolvimento e execução é o Windows 98, utilizando como ferramenta de desenvolvimento o Microsoft Visual C++ 6.0 e a biblioteca *DirectX*.

ABSTRACT

This work is a study of entertainment software development and 2D modeling. It's goal is the specification and implementation of a space combat game, using multimedia techniques. The player will be able to play a match against someone that is miles away through the Internet. It was built to run in Windows 98 operation system, and was developed using the Microsoft Visual C++ 6.0 development environment and the DirectX library.

1 INTRODUÇÃO

Não são raros os casos em que o primeiro contato entre o ser humano e o mundo da informática começou por intermédio de jogos. Os jogos são um mercado que movimentam muito dinheiro. Segundo [WAT1996], nos dias de hoje o mercado de jogos de computadores é maior que o mercado para filmes.

Pode-se dividir jogos segundo algumas de suas características. Pode ser classificado quanto às dimensões de seus gráficos (2D ou 3D), pode também ser dividido segundo o tema (estratégia, simulação, ação, educativo,...). Pode ser classificado, ainda, quanto ao modo em que é jogado (individualmente e/ou grupo). Para ilustrar, pode-se classificar o jogo Quake2, (jogo desenvolvida pela Id Software [IDS2000]) segundo suas características, sendo este um jogo de ação em 3D que pode ser jogado individualmente ou em grupo. Seu sucessor, Quake3, foi desenvolvido para utilizar a Internet como ambiente.

O protótipo proposto neste trabalho é classificado como um jogo em duas dimensões (2D), e seu assunto é ação. Trata-se de um jogo para ser jogado em grupo, onde o jogador controla uma nave e tem uma equipe. Esta equipe pode ter naves controladas pelo computador, tendo sempre como o principal objetivo destruir o maior número de naves inimigas.

Mas, apenas os conhecimentos de computação gráfica não foram suficientes para a conclusão do trabalho. Segundo [WAT1996] “para escrever jogos de computador e simuladores de realidade virtual são necessárias várias disciplinas de programação. Se virmos um jogo simplesmente como um programa de computador, nós iremos nos preocupar com gráficos em tempo real, efeitos sonoros, estrutura de dados, modelos de simulação para objetos que interagem entre si e a interação entre o modelo de simulação e as entradas do usuário.”

Para que as naves controladas pelo computador não façam movimentos randômicos, um estudo na área de Inteligência Artificial foi realizado. Desse modo os *bots* (nave controlada pelo computador, abreviatura de *robot*) têm reações próximas das dos

humanos, apreendendo e até mesmo esperando o momento certo de atacar.

A especificação do protótipo foi feita segundo a *Unified Modeling Language* (UML), e o protótipo foi implementado na linguagem C++.

1.1 MOTIVAÇÃO

Grande parte dos usuários de micro computadores iniciou o uso do mesmo motivado por algum jogo. Grande parte dessas pessoas não imagina o que é necessário para construir um, onde exige o conhecimento não focado em apenas uma área.

É necessário o embasamento em Física para poder modelar as colisões e os choques entre os objetos do jogo. Habilidades em Computação Gráfica são necessárias para a construção gráfica dos objetos do jogo, assim como o projeto da interface com o usuário. Não menos importante é a Inteligência Artificial para que a interação entre o jogador e os personagens controlados pelo computador seja racional e humana. O conhecimento em programação e estrutura de dados é igualmente necessário, pois é nesse momento que todo o embasamento nas outras áreas é integrado, sempre objetivando a melhor performance.

Sendo crescente, no aspecto monetário, esse ramo da informática e possibilitar a integração de várias áreas do conhecimento é importante um estudo nesse tipo de software.

1.2 OBJETIVOS

O objetivo deste trabalho é desenvolver um protótipo de um jogo espacial em duas dimensões, para ser jogado em um ambiente de rede utilizando a biblioteca gráfica *DirectX* e recursos multimídia, onde se tenha uma interação entre os objetos (naves controladas pelo computador).

1.3 RELEVÂNCIA

Integrar Inteligência Artificial em um ambiente de entretenimento, tendo também a utilização de recursos multimídia. Além disso, o estudo do sincronismo necessário para

implementar um jogo que possibilita que vários jogadores disputem uma partida, estando separados geograficamente, tendo a Internet como meio de transmissão.

1.4 ORGANIZAÇÃO DO TEXTO

O texto organiza-se nos capítulos abaixo:

a) Capítulo 1 – Introdução: o presente capítulo descreve o trabalho de forma geral.
b) Capítulo 2 – Entretenimento: apresenta o conceito de entretenimento, principalmente relacionado a jogos de computador.
c) Capítulo 3 – Inteligência Artificial: demonstra a utilização dessa área da computação em jogos.
d) Capítulo 4 – Análise e Projetos de Jogos: nesse capítulo são apresentados os passos na construção de jogos.
e) Capitulo 5 – Desenvolvimento: descreve-se, nessa parte, as ações realizadas para desenvolver o protótipo.
f) Capítulo 6 – Conclusão: demonstra-se as conclusões obtidas durante o trabalho, assim como extensões e dificuldades encontradas.

2 ENTRETENIMENTO

Segundo o dicionário Aurélio [FER1985] entretenimento significa distração, um momento de diversão e alegria. Pode ser obtido a partir de uma conversa animada com amigos, um bom livro, um programa de televisão engraçado. Mas com o advento da tecnologia que invade os lares de forma impressionante, ele passou a ser cibernético também. Pessoas passam horas navegando pela Internet, conversando com pessoas de outros países, comprando livros e cd's, entre outros. Além da diversão que a Internet pode proporcionar, o computador também oferece o entretenimento através de jogos.

Jogos que vão do mais simples como um jogo de cartas, aos mais complexos em que se deve invadir uma base inimiga sem ser percebido. Sem sombra de dúvida os jogos se tornam mais interessantes quando são disputados com amigos, ou pessoas totalmente desconhecidas que podem morar até mesmo há milhares de quilômetros. Atualmente isto é possível tendo em vista que a Internet permite ter jogadores remotos participando do mesmo jogo.

Essa é uma indústria em que fazer testes em produtos significa disputar um jogo até que se chegue ao seu final. Trata-se de uma área que até o trabalho, de uma certa forma, é divertido.

2.1 JOGOS

Sabendo que a história da computação teve seu início nos anos 60, é interessante notar que de acordo com Lamothe [LAM1999] já na década de 70 existiam jogos rodando em mainframes, em forma de texto ou com gráficos de baixa qualidade. Finalmente em 1976-1978 foram inseridos no mercado os primeiros computadores que um consumidor poderia comprar (TRS-80, Atari, Apple), e juntos alguns jogos também de baixa qualidade. Na década de 80 os microcomputadores de 16 bits surgiram, assim como os primeiros jogos em 3D.

Em 1993, quando o mercado era dominado pelos micros compatíveis com o padrão IBM-PC, a Id Software [IDS2000] lança o jogo chamado Doom. O lançamento do Doom provou que a plataforma IBM-PC poderia suportar jogos com mais qualidade.

De uma forma geral, percebe-se que muita coisa foi feita até a presente data, tais como bibliotecas gráficas com ótima performance, placas aceleradoras de vídeo, computadores mais velozes e ferramentas de produção sonora e gráfica incríveis. Muitas vezes construir um jogo é tarefa que deve ser planejada e demorada. Mas uma boa idéia e um projeto bem executado pode levar qualquer um ao mundo dos milionários. Nos dias de hoje é uma industria tão forte como a de filmes [WAT1996].

Lamothe [LAM1999] divide os jogos em 10 grupos:

a) Em primeira pessoa: são em 3D, e a visão é da perspectiva do personagem. Teoricamente são os mais difíceis de serem implementados;

b) Esporte: podem ser 2D ou 3D, os gráficos não são tão importantes como em jogos em primeira pessoa. Comparando com os outros grupos, esse é o que utiliza a Inteligência Artificial da forma mais avançada;

c) Luta marcial: geralmente 2D ou câmeras 3D que flutuam pelo cenário para mostrar os melhores ângulos;

d) Tiro/Plataforma: são jogos que já vem da época dos primeiros jogos. São 2D, mas estão sendo reformulados para 3D;

e) Simulação mecânica: são jogos 3D que simulam tanques, aviões, submarinos, carros, entre outros;

f) Ecossistema: são jogos onde existe uma simulação de desenvolvimento de alguma organização ou grupo. O exemplo clássico é SimCity que é um jogo onde o objetivo é executar da melhor forma o papel de prefeito de uma cidade;

g) Estratégia ou guerra: jogos onde o tempo é dividido em turnos, tendo cada participante sua vez de jogar em cada rodada;

h) Estórias interativas: são feitos utilizando filmagens reais, e não se tem a liberdade de realizar qualquer ação. É como jogar um livro;

i) Jogos antigos: são jogos que fizeram sucesso no passado e são escritos novamente para tirar proveito das novas tecnologias;

j) Enigmas: podem ser 2D ou 3D e são como um quebra-cabeça. O exemplo clássico é Tetris.

Além disso, um jogo pode ser implementado para desfrutar o poder de diversão de sua utilização em uma rede de computadores, permitindo assim vários jogadores na mesma

partida. Conhecidos como jogos para multi usuários (*Multiplayer*).

O protótipo objetivo desse trabalho é classificado como Tiro/Plataforma, sendo em duas dimensões. Cada participante da partida defende uma equipe (azul ou vermelha). Cada equipe possui um cristal que deve ser protegido. O objetivo do jogo é fazer a maior quantidade de pontos por equipe. Os pontos são obtidos destruindo as naves inimigas, assim como roubando o cristal da equipe adversária, entre outras maneiras. A ação que vale mais pontos é denominada “capturar o cristal” e é realizada no momento em que uma nave, que encontra-se com o cristal adversário encosta no cristal de sua equipe, estando este na posição inicial.

Quando uma nave rouba o cristal da outra equipe, os membros desta devem tentar destruir a nave que possui o cristal. Se essa nave for destruída o cristal fica na posição onde a nave explodiu, retornando a posição inicial no momento em que uma nave da mesma equipe do cristal encostar nele.

2.2 MULTIPLAYER

Michael [MIC1999] afirma que na época em que os primeiro jogos foram escritos, o suporte a múltiplos jogadores já existia, não sendo assim uma característica nova. É incomum nos tempos de hoje um jogo ser lançado sem essa capacidade. Existem até serviços na Internet (IGZ, TEN, entre outros) para esse tipo de jogo. A empresa Eletronic Arts lançou em meados de 1996 o jogo Ultima Online [UOL2000], onde o usuário só consegue jogar se estiver conectado. Ele interpreta um personagem dentro da história que é construída por todos os jogadores que tem esse jogo. Uma verdadeira comunidade virtual é simulada por todos os jogadores, das mais diversas partes do mundo. Isso demonstra que os jogos de computador estão se tornando um meio de interação social. Os jogos passaram a possibilitar que amigos passem um tempo se divertindo e interagindo eletronicamente, mesmo que estejam distantes entre si.

O meio de comunicação é o modo com que as informações dos jogadores transitam por uma rede de computadores. Esse fluxo de informações pode ocorrer através de uma rede telefônica, pela Internet (*TCP/IP*) ou qualquer outro protocolo de comunicação. Outro fator

importante é o modelo de comunicação, que pode ser ponto a ponto ou cliente-servidor.

No modelo ponto a ponto cada jogador que participa da partida deve ter um conhecimento apurado acerca dos outros jogadores. Utilizando-se como meio de comunicação a Internet, isso implicaria o conhecimento prévio do endereço *IP* de cada jogador.

Já no modelo cliente-servidor as informações trafegam de forma centralizada. Nesse modelo tem-se uma máquina conhecida como servidor ou *host*, que recebe as informações de todos os jogadores e tem a obrigação de distribuir essa informações entre os outros jogadores. Novamente tendo como meio de comunicação a Internet, a máquina servidora é, geralmente, disponibilizada pelo fabricante do jogo e tem seu endereço IP amplamente divulgado.

Para um jogo ter suporte a esse tipo de partida é necessário apresentar algumas funções. A primeira delas é a capacidade dos jogadores trocarem dicas, fazerem amizade ou até mesmo combinarem partidas. O modo de comunicação a ser utilizado é o *Chat* (modo eletrônico de ter uma conversa com outra pessoa). Já quando se está no meio de uma partida o *Chat* passa a ser utilizado para informar onde estão determinados objetos ou inimigos, ou no caso do jogo suportar times, combinar um ataque em grupo.

A comunicação entre jogadores, assim como a transmissão das informações dos jogadores, é realizada através do componente *Direct Play*, que faz parte do *DirectX*. É uma biblioteca desenvolvida pela Microsoft [MIC2000] para facilitar o desenvolvimento de jogos e programas multimídia, eliminando a diferença entre hardwares, tendo como base à metodologia de componentes criada e adotada pela Microsoft, chamada *COM* [ROG1997].

O *Direct Play* pode ser utilizado nos mais diversos meios de comunicação como a comunicação via Modem, via protocolo *IPX* ou até mesmo a conexão serial. No protótipo é utilizado como meio de comunicação o protocolo *TCP/IP* por ser o mesmo da Internet. O modelo de comunicação é cliente-servidor.

3 INTELIGÊNCIA ARTIFICIAL

[RAB1996] define a Inteligência Artificial como o resultado da aplicação de técnicas e recursos, especialmente de natureza não numérica, viabilizando a solução de problemas que exigiriam do humano certo grau de raciocínio e de perícia.

A Inteligência Artificial, ou IA, tenta entender as entidades inteligentes para poder definir a sua construção [RUS1995]. Ela tenta fazer com que softwares ou hardwares pensem ou raciocinem do modo que os humanos fazem, automatizando assim o processo de tomada de decisão e resolução de problemas.

O raciocínio do computador pode ser obtido através de várias formas. Uma delas são as redes neurais que é uma aproximação do cérebro humano [LAM1999]. Outra maneira é a utilização de sistemas genéticos que é um conjunto de técnicas para que o sistema evolua de um modo biológico.

Sua utilização é encontrada nas mais diversas áreas do conhecimento. Um dos softwares mais conhecidos, MyCin, é capaz de dar um diagnóstico de uma possível doença através dos sintomas de um paciente. Outros softwares podem auxiliar a tomada de decisões, com base na análise de dados. Para permitir que um jogador tenha um desafio ao enfrentar o computador em uma partida de xadrez, se faz necessário também a presença de IA.

A IA aplicada a jogos vem avançando significativamente. Dos antigos jogos de xadrez eletrônico ao recente desafio entre Kasparov e o super computador *Deep Blue*, evento que trouxe a IA uma popularidade no mundo inteiro, ocorreram grandes avanços [BAR2000]. De acordo com Lamothe [LAM1999] o uso da IA para esse tipo de aplicação demonstrou avanços e farão a diferença entre os jogos.

Em jogos, a IA é utilizada para permitir que os adversários ou personagens simulados pelo computador e que interagem com o jogador humano tomem decisões e ajam como um ser racional, desafiando seus oponentes. De acordo com [HOW2000] a IA de um jogo não é representada apenas por redes neurais, aprendizado e complexas estruturas matemáticas, ainda que possa ser, mas primeiramente o objetivo é de criar um ambiente em que os objetos aparentam estar pensando.

O personagem controlado por intermédio da IA pode ter personalidade, podendo agir com medo ou extrema coragem. Além disso, algoritmos dessa área podem ser utilizados para perseguir adversários por labirintos, fugir de inimigos, procurar ouro em uma mina entre outros.

Existem inúmeros métodos para diferentes tarefas. Um algoritmo utilizado em um jogo de xadrez não é semelhante ao método utilizado em um jogo de futebol. Os métodos utilizados dependem completamente da natureza do jogo [BAR2000].

Os jogos mais antigos tinham como IA um conjunto de condições (*IF's*) para definir seus movimentos e ações. Os jogos de xadrez por sua vez, em sua maioria, utilizam métodos de busca em árvore. A cada instante em que um movimento precisa ser realizado são criados caminhos por onde o jogo pode seguir, dependendo do movimento simulado. O caminho que apresenta a melhor chance de vitória é seguido. O melhor caminho é selecionado através de uma função de pontuação para cada estado do mundo.

Mas muitas vezes sabe-se como chegar a resposta de forma mais rápida. Aplicando um conhecimento prévio do problema pode-se reduzir o tempo de busca a resposta. Essa abordagem é chamada de heurística.

Ás vezes o jogo é complicado para um algoritmo de força bruta como o de busca em árvore. Em situações como essa não se pensa em métodos rudimentares e exaustivos. Para esses tipos de problemas são usadas técnicas avançadas como o planejamento.

Planejamento é o processo de dividir uma tarefa em tarefas menores, ordenando de acordo com suas naturezas e dependências e completando-as na ordem [BAR2000]. Esse é o método utilizado pelas pessoas na resolução de seus problemas.

Utilizando essas técnicas pode-se permitir que o personagem controlado aprenda com seus adversários e ainda lembre-se dos lugares e fatos que vivenciou, fazendo com que cada ação não seja mais atômica, ou seja, tomada na hora. Utilizando técnicas de memória e aprendizado, o mecanismo de decisão utiliza toda a história do objeto em questão para auxiliar em sua resposta ao estado do mundo [LAM1999].

No protótipo foram utilizadas as técnicas de planejamento e memória, permitindo que cada nave controlada virtualmente pense o que é preciso para alcançar seu objetivo momentâneo, com o auxílio da análise de suas ações do passado.

3.1 PLANEJAMENTO

Imaginando que se está em casa e surge uma vontade de tomar leite. O problema é que não há nenhuma caixa de leite na geladeira. Pensa-se em comprar leite no mercado mais próximo. Automaticamente já são conhecidos os passos para realizar essa tarefa. Ir até o local onde está a chave do carro para pegar a mesma. Ir até a garagem entrar no carro e sair da garagem. Até que, através de *n* passos, se chega novamente em casa.

Em uma abordagem de Inteligência Artificial, o ato de comprar leite no supermercado é um objetivo. Esse objetivo será alcançado concluindo pequenos passos. Esse método é conhecido como planejamento.

De acordo com [RUS1995] um plano é apenas um conjunto de ações para se alcançar um objetivo. A partir do modo em que o mundo se encontra é feito um planejamento de ações para que o objetivo seja alcançado.

O problema de buscar leite é muito simples, pois não muda durante sua realização. Ele fica mais complexo quando o estado do mundo se altera, obrigando que uma nova decisão seja feita para que um novo plano, se necessário, seja adotado.

Para exemplificar essa alteração de plano, pode-se imaginar um jogo de combate espacial entre naves. Cada nave possui uma equipe, tendo como objetivo a destruição das naves da outra equipe (similar ao protótipo). As mesmas perdem energia durante o combate, podendo reconstituir-se através de itens de energia espalhados pelo mundo. O objetivo de cada nave controlada por IA pode mudar durante a partida. No momento em que a nave está com pouca energia seu objetivo seria evitar as naves inimigas e buscar energia para se proteger. Uma vez reconstituída sua energia, a nave passa a ter o objetivo de atacar qualquer inimigo que cruze seu caminho.

Para alcançar o objetivo é necessária a realização de algumas ações. Tendo como

exemplo o objetivo de buscar energia, um conjunto de ações plausível poderia ser:

- afastar-se da área de combate;
- esperar ajuda de alguma nave companheira;
- ir em direção a uma fonte de energia, evitando adversários.

3.2 MEMÓRIA

Mais uma vez fazendo uma comparação com o cotidiano pode-se demonstrar a memória no exemplo em que se sente a necessidade de comprar leite. Se sempre que se decidesse comprar leite se fizesse uma pesquisa nos jornais para procurar o melhor preço, não seríamos tão inteligentes. Ao invés disso, para definir o local onde se vai comprar o leite simplesmente lembra-se o último lugar onde essa mercadoria foi comprada, ou ainda algum lugar indicado por um amigo.

Segundo [HOW2000] usar um modelo de inferência, ou seja, de tomada de decisão, em jogos que utiliza apenas o estado atual do mundo para obter suas conclusões limita a sensação de realidade e o desafio de enfrentar um oponente que usa técnicas de IA. De acordo com [LAM1999], um adversário que toma decisões sem utilizar o conhecimento obtido até então se restringe ao pensamento momentâneo.

Imaginando que a nave controlada pelo computador não utiliza essa técnica, e em um dado estado do mundo ela se encontra de frente com um outro jogador. No momento que ela dispara contra seu oponente ele se move à direita. No próximo momento ela atira na direita, pois é nessa posição que encontra-se o alvo, mas, seu oponente não hesita e vai à esquerda, desviando novamente dos projéteis lançados. Esses passos podem repetir infinitamente, sem que a nave controlada por IA atinja seu objetivo, que é destruir seu oponente. Se na mesma situação utiliza-se a técnica de memória, a nave iria perceber que seu oponente ora vai para direita e ora vai para esquerda, levando a tomar a decisão de atirar, por exemplo, duas vezes seguidas na mesma direção, surpreendendo seu adversário e tornando–a mais humana.

Um outro exemplo para a utilização de memória, seria na busca de algum item ou

objeto. Sempre que esse item é encontrado, sua posição é guardada na memória. Desse modo na próxima vez que esse item for necessário, o algoritmo de busca não iria procurar aleatoriamente, mas sim procuraria primeiramente nos locais onde o mesmo já foi encontrado. Em casos de jogos que utilizam o conceito de equipes, ou seja, objetos que ajudam um ao outro, é interessante utilizar um mecanismo de comunicação entre os membros da mesma equipe, permitindo assim que eles troquem informações.

Lamothe [LAM1999] lembra que não é justo permitir que os objetos controlados por IA conheçam o mundo sem explorá-lo, do modo que os outros jogadores devem fazer.

4 ANÁLISE E PROJETO DE JOGOS

Para a criação de um jogo é necessário um embasamento em algumas áreas. Em sua maioria, as equipes que desenvolvem jogos subdividem-se em:

- equipe de programação;
- responsáveis pela arte gráfica;
- produtores musicais;
- coordenadores gerais.

Isso demonstra que apenas um bom compilador e uma boa idéia não são o suficiente para a criação de um jogo. Além de uma boa ferramenta de desenvolvimento são necessários aplicativos de manipulação de imagens 2D e 3D bem como softwares de processamento de som. Em jogos em que há fases ou níveis, é necessária a criação de um aplicativo para edição das fases que estarão no jogo.

4.1 PROJETO DE JOGOS

A criação de um jogo tem seu início em uma idéia. Pensando ou sonhando se imagina uma aventura, um combate ou até mesmo um quebra-cabeça. Depois de que o cenário do jogo esta imaginado os objetivos, os inimigos, os objetos e principalmente na diversão que o jogo irá proporcionar é pensada. Desse modo o primeiro passo para um projeto de um jogo já foi dado, a idéia.

De acordo com [CLA2000] um método para se avaliar a qualidade de uma idéia para um novo jogo é contar aos amigos. Se eles não ficarem animados já é um forte indicio de que o jogo não é tão divertido quanto parece.

Depois de uma idéia que realmente impressiona, o próximo passo é procurar as pessoas que irão participar do projeto, no caso de uma nova equipe. Artistas gráficos, produtores musicais e de efeitos sonoros, projetistas e programadores compõe essa equipe.

O passo seguinte é composto por reuniões (*brainstorms*) com intuito de elaborar o projeto por completo. Todos os detalhes devem ser documentados com o máximo de detalhes (implementação). Essa é a fase mais importante de todo o projeto. Todos devem estar convencidos de que o projeto é original e promissor.

Tudo que foi decidido deve estar descrito no que é chamado de *Design Document* (Documento do Projeto). De acordo com [CLA2000] esse documento deve conter desenhos de telas, situações e características do jogo. Entre as informações que esse documento deve conter estão as opções de jogo (todos os passos para iniciar um jogo), os menus, ajuda, detalhes de fases, recursos de áudio e gráficos. Os níveis (fases) do jogo também devem ser descritos e ilustrados nesse documento. Segundo [MAS2000] para o sucesso de uma equipe é necessário que seus membros tenham uma visão comum do que se deseja realizar.Utilizando essa descrição é que cada membro identificará o seu objetivo ou função dentro do projeto.

Infelizmente a estatística aponta um número muito alto de fracassos no desenvolvimento de jogos. Segundo [LAR2000] são lançados por ano uma média de 1.200 a 3.000 jogos para PC's, sendo que poucos desses obtém sucesso. Esse número expressa apenas os projetos que foram concluídos; pelo menos uma mesma quantidade fracassou durante o projeto.

Dentre as razões apontadas por [LAR2000] para o fracasso num projeto, as mais freqüentes são:

a) falta de originalidade: muitos projetos são cópias de jogos de sucesso, e por isso enfrentam um mercado concorrido. Enfrentar uma grande empresa com orçamentos altos é algo complicado;

b) a má escolha dos membros da equipe: não só a incompetência de um membro leva ao fracasso de todo um time. O ego ou a falta de habilidade do trabalho em grupo também pode levar um projeto a uma morte prematura;

c) falha ao reconhecer seus limites: ao se projetar um jogo deve-se conhecer os limites da equipe. Se o desejo é produzir um jogo com os melhores sons e gráficos 3D deve-se prever um orçamento e um tempo maior para o projeto, o que muitas vezes não é possível. De acordo com [MAS2000] isso não significa que a fase de desenvolvimento não deva ter desafios. O que é necessário é saber dosar os desafios.

Um projeto de jogo se assemelha a um projeto normal de software. Tem os mesmos problemas de formação e integração de equipes, prazos e levantamento de informações e características necessárias para o sucesso. As principais diferenças são o público alvo e objetivos do produto.

4.2 ESTRUTURA DE UM JOGO

Ao implementar um jogo parte-se de um esqueleto que é utilizado pela maioria dos desenvolvedores [LAM1999].

Lamothe [LAM1999] subdivide a execução de um jogo em 8 seções:

1. **Inicialização:** aqui se deve alocar memória, alocar recursos, carregar dados de arquivo, assim como a inicialização de um programa normal;

2. **Entrar no *Game loop*:** tudo começa aqui, as ações são controladas aqui até as saída do programa;

3. **Ler entrada do jogador:** os comando executados pelo jogador são verificados para utilização nas seções de Inteligência Artificial e lógica;

4. **Executar algoritmos de Inteligência Artificial e lógica do jogo:** seção que detém a maior parte do código. A Inteligência Artificial, a Física e a lógica do jogo são executadas nessa parte. Os resultados obtidos são utilizados para se produzir o próximo quadro do jogo;

5. **Criar o próximo quadro do jogo:** os resultados obtidos nas seções 3 e 4 são graficamente criados, para serem exibidos na tela do computador;

6. **Sincronizar a Imagem:** devido à diferença de velocidade entre as máquinas e a complexidade de cada jogo, o objetivo dessa parte é exatamente a sincronização do jogo para que ele seja executado respeitando uma taxa de atualização. Um valor recomendado é 30 quadros por segundo;

7. **Final do Game Loop:** volta-se à seção 2;

8. **Terminar o aplicativo:** quando o usuário deseja parar de jogar, deve-se liberar todos os recursos utilizados, e voltar ao sistema operacional.

4.3 PERFOMANCE OU METODOLOGIA POLITICAMENTE CORRETA

O fato é que um jogo é diferente de qualquer aplicativo. Diferentemente de um aplicativo normal, a velocidade de execução de um jogo determina se ele vai ser deixado de lado ou vai ser utilizado (assim como seu grau de diversão, entre outros requisitos).

Dificilmente um jogador terá paciência de utilizar um jogo que é lento, onde os gráficos são executados praticamente quadro a quadro. Ao implementar um jogo tem-se em mente uma abordagem diferente, comparando com a implementação de um aplicativo convencional. Qualquer rotina que é desenvolvida deve ser construída de maneira a otimizar a performance, muitas vezes fazendo com que a mesma não seja elegante. Segundo [LAM1999] deve-se evitar ao máximo utilizar funções de alto nível, ou seja, que abstraem a complexidade de uma rotina ou parte do código. Ele afirma ainda que tudo deve ser reescrito para que a performance seja melhor.

O que muitas vezes não é visto como uma boa prática de programação pode ser a solução que apresenta a melhor performance. Isso torna o código fonte de um jogo um tanto que academicamente incorreto, mas o fato é que a implementação objetiva a melhor performance. Lamothe [LAM1999] cita como um exemplo disso um função que desenha um ponto em uma determinada posição da tela. A definição dessa função pode ser DesenhaPonto(x, y). Essa função é simples, o que faz com que a chamada da função seja mais demorada que a execução da própria função. A solução adotada por Lamothe é a definição de variáveís globais que recebem o valor de x e y, diminuindo assim o tempo da execução da chamada de função, pois não se faz necessário o tratamento de seus parâmetros.

Outros pequenos exemplos dessa preocupação podem ser demonstrados como o uso de alocação dinâmica de memória. Não se usa lista encadeada, se um vetor estático de *n* elementos pode resolver o problema. As estruturas utilizadas devem aproveitar o máximo dos processadores e sistemas operacionais. Um dos elementos que devem ser observados são os tipos de dados utilizados. De acordo com [LAM1999] preferencialmente deve-se utilizar dados de 32 bits, porque os processadores Pentium e mais novos tem uma arquitetura de 32 bits. A utilização de classes (orientação a objetos) é bem vista por Lamothe, contanto que não

se exagere nas hierarquias.

4.4 COLISÕES

Os objetos que constituem o mundo de um jogo possuem massa assim como uma posição no espaço [LAM1999]. Como a lei da física já afirmava que dois corpos não podem ocupar o mesmo espaço ao mesmo tempo, o mesmo deve ser respeitado ao criar jogos.

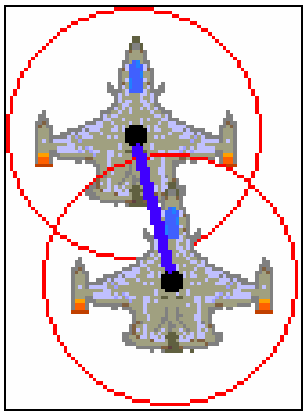
Para jogos 2D são inúmeros os métodos para a verificação de colisões. Partindo do princípio de que cada elemento possui uma posição (x, y), uma largura e uma altura tem se assim um quadrado. O método mais simples constitui-se na verificação de que algum retângulo invade o espaço de outro.

Essa solução é rápida, porém apresenta um resultado não realístico. A figura 1 demonstra o quadrado que representa a nave, o qual possui muitos pontos em que não há nave, podendo resultar em uma falsa colisão.

Figura 1 - Uma nave representada por um quadrado.

Outro método é representar o espaço utilizado por cada objeto como uma circunferência. Para determinar se houve uma colisão calcula-se à distância (distância entre pontos) entre os dois objetos. Para uma colisão existir o resultado desse cálculo deve ser menor ou igual a soma dos raios dos objetos multiplicados por uma constante que representa o quanto da circunferência é utilizada pelos objetos. Esse método é ilustrado na figura 2.

Figura 2 - Utilizando o método da circunferência.



Inúmeros são os métodos para a detecção de colisão podendo chegar ao nível de checar a colisão *pixel* a *pixel* (menor elemento da tela de um computador), obtendo assim um resultado perfeito.

Após a verificação da existência ou não de uma colisão e obtendo uma resposta positiva deve-se modelar o resultado da mesma.

Utilizando uma colisão entre naves para exemplificar, o resultado ocasionaria a mudança de direção, uma perda de velocidade e uma perda de energia dos objetos que colidiram. Para modelar os efeitos de uma colisão leva-se em consideração entre outros a massa, a velocidade e a direção dos objetos. Esses valores não expressam necessariamente um valor real. É necessário saber apenas que uma nave tem duas vezes mais massa que uma outra nave. As colisões entre naves e seus respectivos resultados não foram implementados no protótipo.

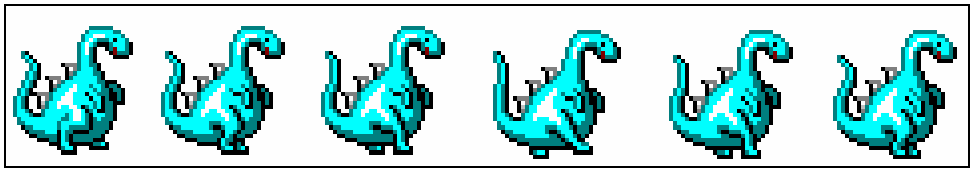
4.5 ARTE GRÁFICA

De acordo com Lamothe [LAM1999] a arte gráfica de um jogo é uma peça fundamental para o seu sucesso. É nesse momento que se define o aspecto do jogo, assim como de cada elemento. Telas de abertura, menus, objetos e fontes são criadas nesse instante.

A construção da arte gráfica de um jogo depende de seu tipo. Se ele for um jogo 3D, deve-se ter um modelo geométrico para cada objeto, que pode ser revestido por uma textura. A textura geralmente é um arquivo gráfico. Para obter um determinado estado de algum objeto se aplica alguma das transformações geométricas (escala, translação ou rotação).

Já em um jogo em duas dimensões, todo estado de cada objeto deve ser representado por uma ou mais imagens previamente construídas. Interpolando-as consegue-se a animação, como na figura 3.

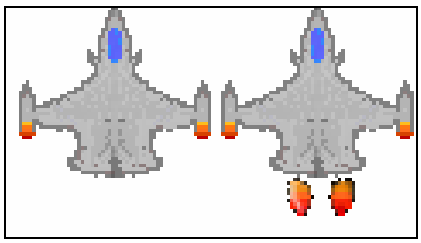
Figura 3 - Animação 2D.



As imagens 2D podem ser geradas por modeladores 3D ou por programas gráficos 2D. Até mesmo o texto utilizado no jogo com o objetivo de exibir informações é obtido a partir de uma imagem. Essa imagem contém todas as letras do alfabeto, assim como os números usados.

No protótipo cada modelo de nave é representado por duas imagens (figura 4). Uma imagem é utilizada quando a nave não está se movimentando. A outra é usada no momento em que a nave tem velocidade, pois essa imagem demonstra movimento através de chamas que saem da turbina do modelo. Ainda, para demonstrar o movimento, são desenhados pequenos pontos pelo caminho que a nave percorre. Esse tipo de ponto é chamado de sistema de partículas [LAM1999].

Figura 4 - Duas imagens representando estados diferentes.



Um sistema de partículas é usado em explosões, dando a impressão de que pequenos

pedaços dos destroços foram arremessados pela força do impacto. Lamothe [LAM1999] define um sistema de partículas como modelos físicos que modelam pequenas partículas, tendo utilização em explosões, vapores, ou luzes em geral. Também é utilizado um sistema de partículas no protótipo no momento em que uma nave explode.

4.6 EFEITOS SONOROS

Para dar mais realismo e emoção ao jogo é indispensável à utilização de efeitos sonoros em algumas situações e eventos.

De acordo com [LAM1999] o computador só consegue reproduzir sons a partir de uma forma digital, ou seja, uma seqüência de 0’s e 1’s. Esses podem ser digitalizados ou sintéticos.

Sons digitalizados são gravados, como uma voz humana, uma explosão ou um barulho de motor. O formato mais conhecido é o *WAV*.

Já os sintetizados são uma reprodução matemática de uma descrição [LAM1999]. Um exemplo desse tipo de som é o arquivo formato *MIDI* (*Musical Instrument Digital Interface*).

Esse tipo de arquivo descreve uma música ou som em relação ao tempo, ligando e desligando um canal de som conforme a descrição. Um exemplo desse tipo de descrição, em uma forma simplificada, pode ser:

- Ligue o canal 1 com a nota SI;
- Ligue o canal 2 com a nota DÓ;
- ...;
- Desligue todos os canais.

Cada canal representa um tipo de instrumento, e o computador que é encarregado de transformar essa seqüência de notas e instrumentos em uma forma digital para que possa ser reproduzida.

A diferença principal entre os dois modos de reprodução está na qualidade do som. Por representar o som de maneira fiel, o arquivo digital tem uma qualidade superior ao formato

sintetizado. Por ser tão real, o arquivo digital tem um tamanho muito maior comparando-o com o formato *MIDI*.

Em jogos a reprodução de sons digitalizados é utilizada para efeitos sonoros, ou seja, explosões, ronco de motores, entre outros. Já o formato sintetizado é utilizado para músicas, por não requerer muita memória e processamento para ser reproduzido.

Mas, com a recente sensação de arquivos *MP3*, alguns jogos já utilizam esse formato para suas músicas. Esse formato é caracterizado pela qualidade digital e pequeno tamanho de seu arquivo.

No protótipo foram utilizados arquivos digitalizados para a reprodução de efeitos sonoros. A música está no formato *MIDI.* Foi realizado um teste para a utilização de músicas no formato *MP3*, mas foi constatado que a reprodução desse tipo de arquivo requer uma quantidade significativa de recursos da máquina. No teste realizado o protótipo toca a música de forma lenta e com paradas repentinas, mas em momento algum afetou a performance geral do jogo.

A reprodução de arquivos *MP3* foi testada através da biblioteca *freeware* XAudio [XAU2000], e frente aos resultados obtidos foi retirada.

5 DESENVOLVIMENTO DO PROTÓTIPO

O protótipo do software desenvolvido tem o intuito de demonstrar a interação entre as diversas áreas necessárias para construção de um simples jogo. No projeto os passos recomendados para a construção de um jogo (ver seção 4.1), não foram seguidos totalmente. Os passos que necesitariam de um grupo como a sua própria formação, assim como *brainstorms* não foram feitos. Mas, esse relatório é tido com o *Design Document* pois descreve o protótipo de forma detalhada.

Segundo a classificação utilizada por Lamothe [LAM1999] (seção 2.1) o protótipo desenvolvido se enquadra no modelo Tipo/Plataforma. Em seu desenvolvimento foi caracterizado o uso de:

- manipulação de imagens 2D;
- integração de som e imagem;
- utilização da Inteligência Artificial para manipulação de naves racionalmente;
- possibilidade de jogar com adversários distribuídos geograficamente, através de uma rede de computadores;
- orientação a objetos.

5.1 ESPECIFICAÇÃO

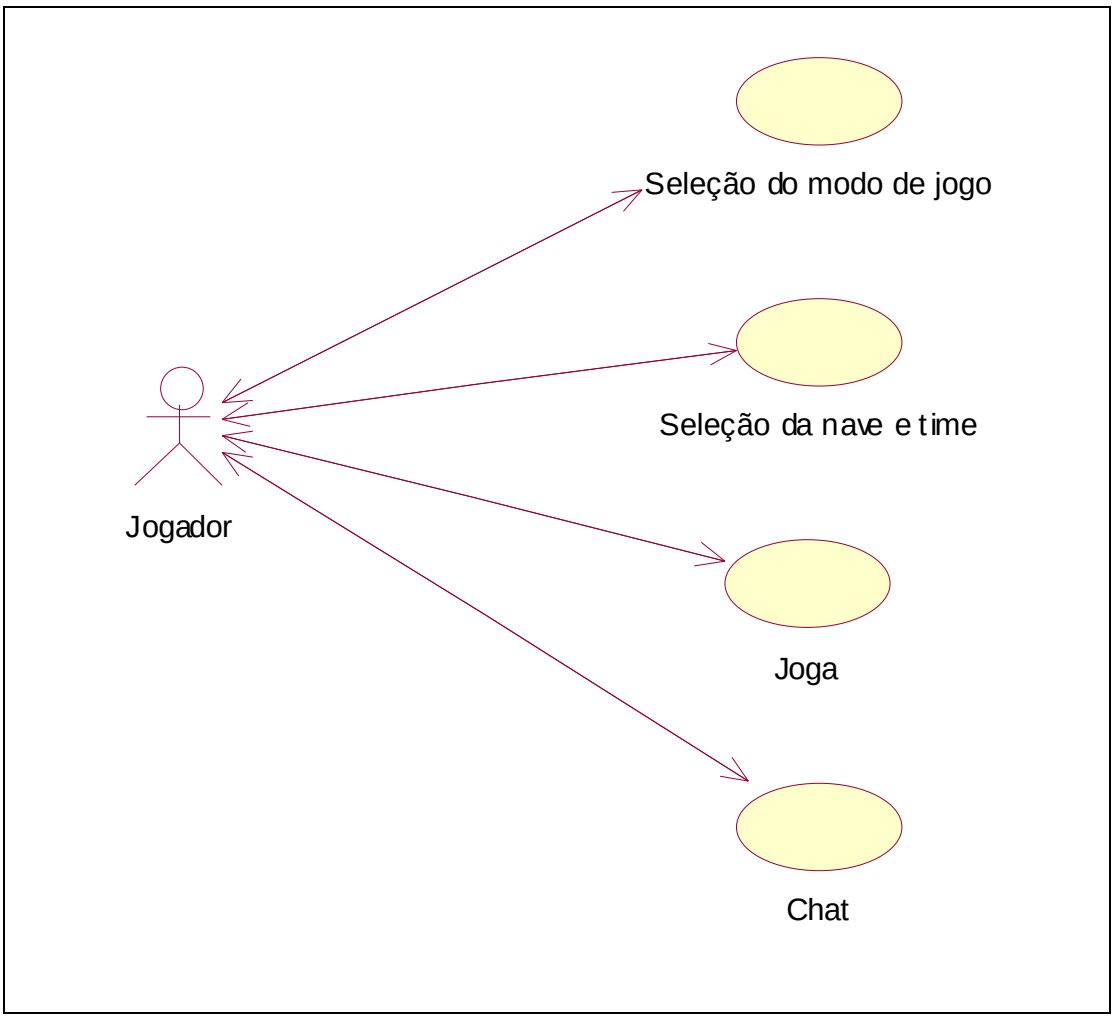
O protótipo é um jogo de combate espacial entre jogadores, através de uma rede de computadores. Antes de poder jogar deve-se criar uma sessão de jogo ou participar de uma existente.

Após isso cada jogador deve escolher seu nome, seu time e sua nave. Depois dessa etapas o jogo começa com o combate entre os participantes. Durante o combate é possível enviar mensagens para os outros jogadores, assim como ver a pontuação do jogo.

Os objetos e a lógica do jogo foram modelados segundo a UML. Segundo [FUR1998] *use cases* definem a funcionalidade do sistema percebida por atores externos. O jogador

interage com o protótipo segundo os *use cases* ilustrados na figura 5.

Figura 5 - Use case do protótipo.



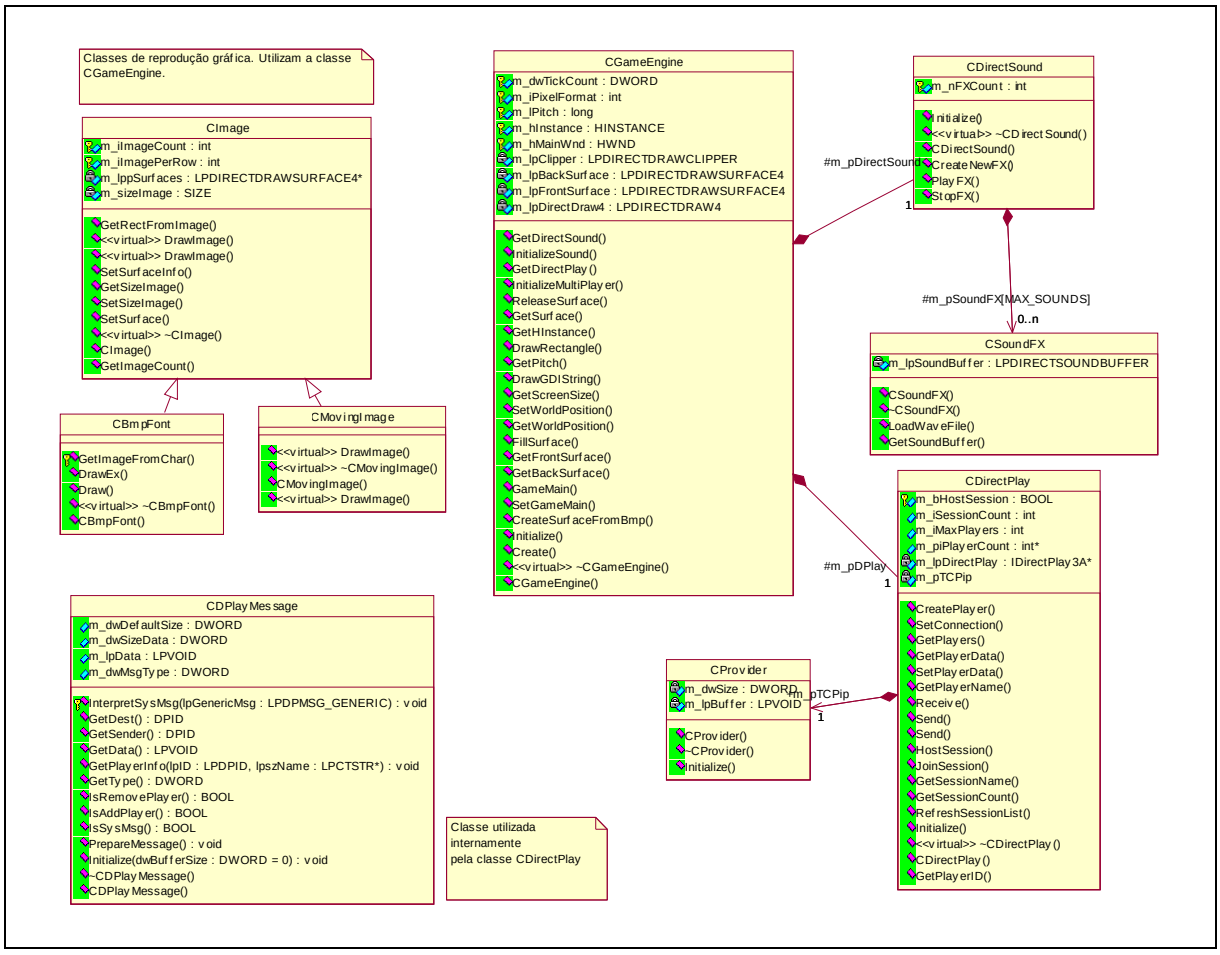
O ator principal, e único, é o próprio jogador. A descrição de cada *use case* é demonstrada na tabela 1.

Tabela 1 - Descrição do use case.

Use Case	Descrição
Seleção do modo de jogo	para participar de uma partida o jogador seleciona uma sessão existente ou cria sua própria sessão
Seleção da nave e time	antes de começar o jogo o jogador seleciona o modelo de nave que deseja utilizar bem como que time deseja representar
Joga	após a configuração o jogador participa do jogo
Chat	durante o jogo o jogador tem a possibilidade de enviar mensagens para os outros jogadores

Para facilitar a compreensão do modelo utilizado, as classes foram separadas em pacotes. Pacotes são um mecanismo da UML para separar elementos de acordo com critério lógico [MUL1997]. O primeiro pacote é chamado de construção de jogo. Ele pode ser utilizado na construção de qualquer jogo em 2D. Sua composição pode ser observada na figura 6.

Figura 6 - Pacote de construção de jogos 2D.



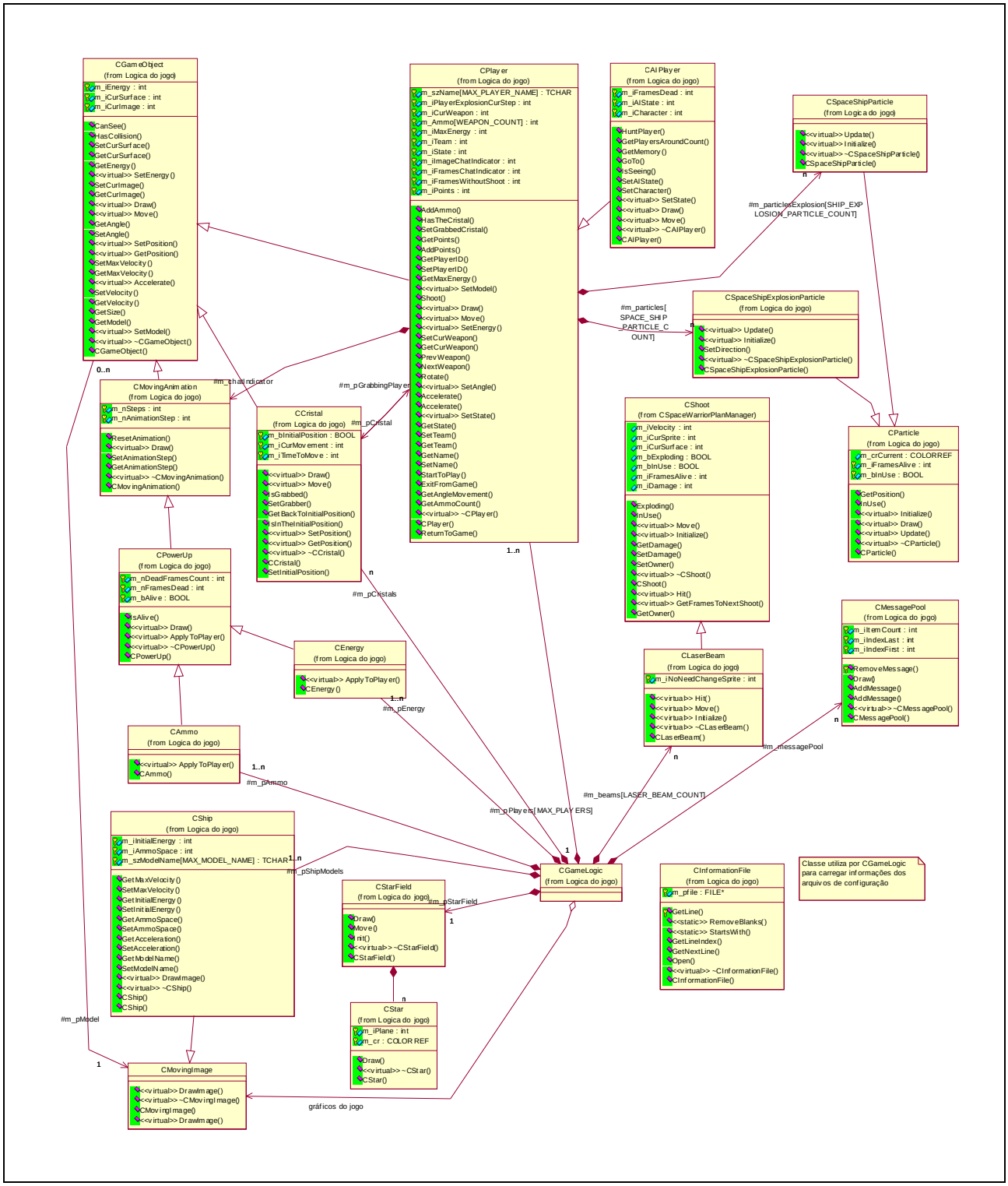
Nesse pacote a classe principal é denominada `CGameEngine` que é responsável pela visualização de imagens (utilizando a biblioteca *DirectDraw*), pela reprodução de efeitos sonoros e música assim como a transmissão de informações através uma rede de computadores. É uma classe base para qualquer jogo do gênero 2D, com reprodução musical e que possua ou não a opção de jogos entre vários jogadores. As classes `CImage` e `CMovingImage` são representações gráficas dos elementos do jogo. Podem ser estáticas isto é, são mostradas sempre na mesma posição ou podem ter sua posição em relação ao mundo.

A classe `CDirectPlay` é responsável pela transmissão de dados pela rede, utilizando a biblioteca *DirectPlay*. A classe `CDirectSound` é responsável pela reprodução de músicas e efeitos sonoros, através da biblioteca *DirectSound*.

O outro pacote desenvolvido foi nomeado lógica do jogo. Ele contém as classes que

caracterizam o protótipo. Essa caracterização é demonstrada por classes que somente tem uso no protótipo. São exemplos a IA utilizada, as naves, os cristais e energias. Elas são dependentes das classes do pacote de construção de jogos. A classe mais importante desse pacote é `CgameLogic, que` controla o jogo. O pacote é ilustrado pela figura 7.

Figura 7 - Pacote de modelagem lógica do jogo.



A lógica e as regras do jogo são implementadas na classe `CGameLogic` que é o

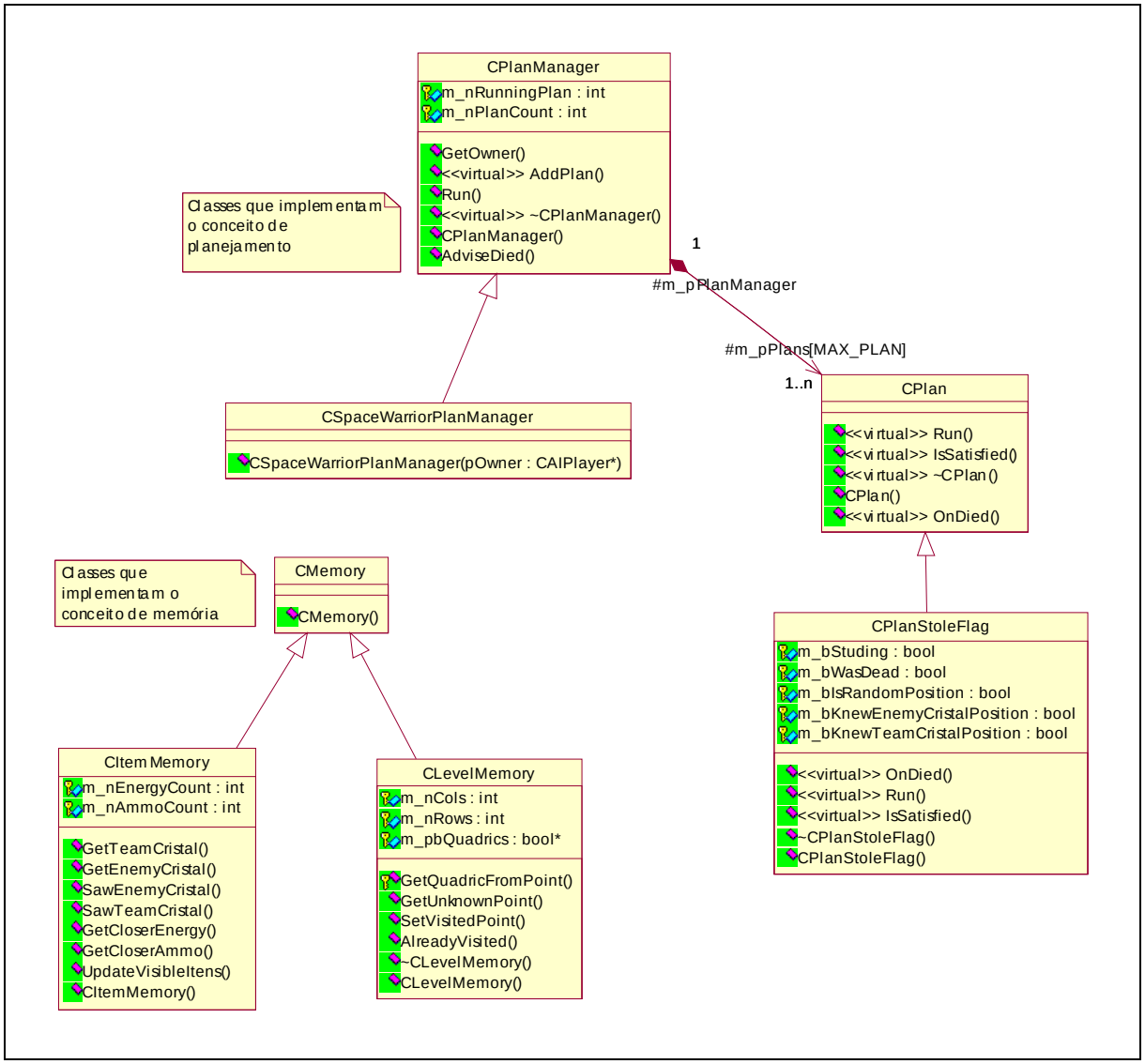
coração do jogo. Essa classe controla os objetos do jogo checando colisões, pontos e até mesmo a comunicação pela rede (através da classe `CDirectPlay` que está agregada a classe `CGameEngine`). Os objetos do jogo estão implementados nas classes:

- `CGameObject`: classe base que contém informações básicas como a sua posição;
- `CPlayer`: implementa os jogadores;
- `CAIPlayer`: são os jogadores controlados pelo computador;
- `CPowerUp`: são os itens que estão distribuídos pelo nível. Essa é apenas a classe base. A energia e a munição são implementadas pelas classes `CEnergy` e `CAmmo` respectivamente, respeitando as regras de herança (Orientação a objetos);
- `CCristal`: são os cristais das equipes.

A ilustração da classe `CGameLogic` é ilustrada de forma completa no anexo 1.

O último pacote corresponde aos algorítmos de IA. A classe `CAIPlayer` (pacote de lógica do jogo) depende desse pacote para sua simulação. O diagrama de classe para o pacote pode ser visto na figura 8.

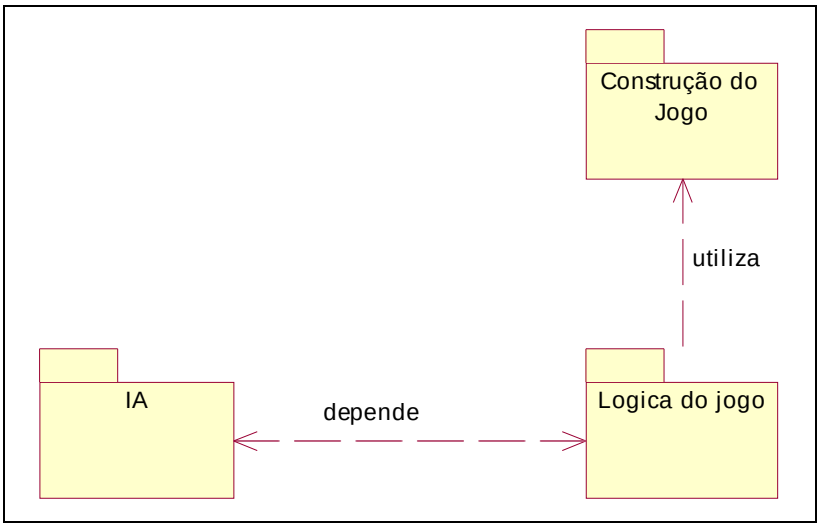
Figura 8 - Pacote de classes de IA utilizadas.



A classe `CPlanManager` e `CSpaceWarriorPlanManager` tem o intuito de utilizar IA através da técnica de planejamento. As classes `CItemMemory` e `CLevelMemory` são especializações da classe `CMemory`, simulando a idéia de memória (seção 3.2).

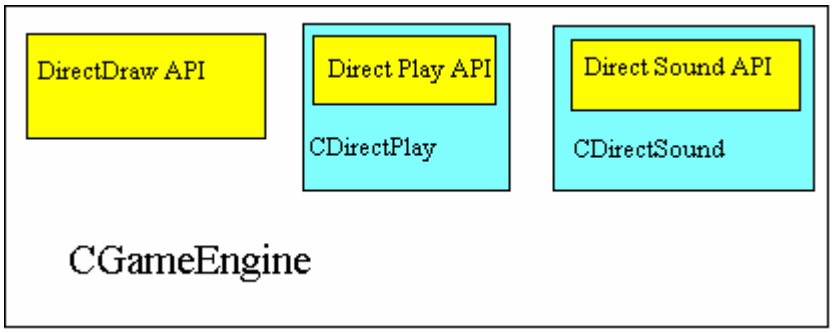
Os pacotes de IA e lógica são mutuamente dependentes. Ao contrário do pacote construção do jogo que é independente. A relação entre pacotes é ilustrada pela figura 9.

Figura 9 - Relação entre os pacotes.



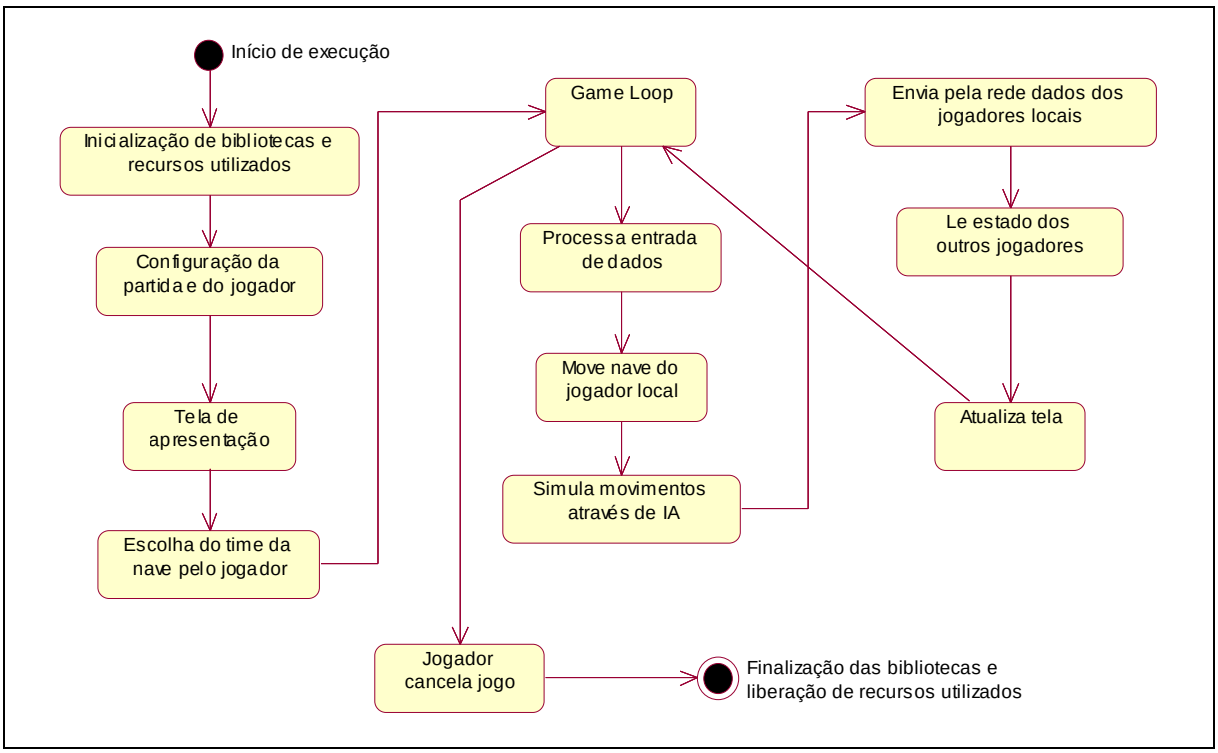
As classes `CDirectPlay`, `CDirectSound` e `CGameEngine` são abstrações das bibliotecas *DirectX* e foram desenvolvidas observando a organização ilustrada na figura 10. Elas utilizam internamente essas bibliotecas, que muitas vezes são complexas.

Figura 10 - Organização das classes utilizadas.



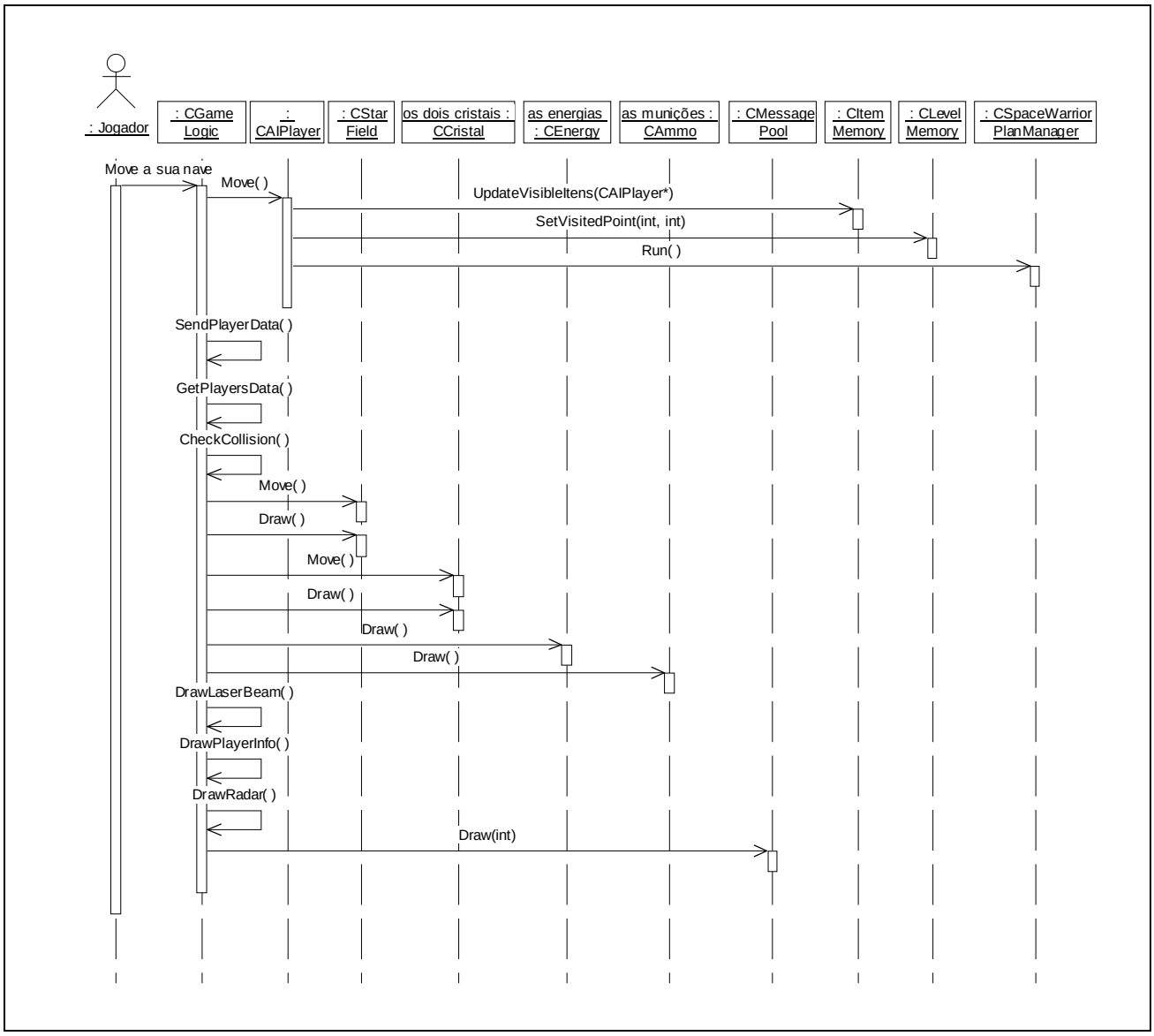
Um diagrama de transição de estados demonstra os possíveis estados de um sistema, assim como possíveis eventos que podem alterá-lo. O diagrama da figura 11 demonstra os estados do protótipo.

Figura 111 - Diagrama de transição de estados.



No *GameLoop* do jogo, tem-se a troca de mensagens entre as classes conforme o diagrama de sequência ilustrado na figura 12.

Figura 12 - Diagrama de sequência do GameLoop.



5.2 IMPLEMENTAÇÃO

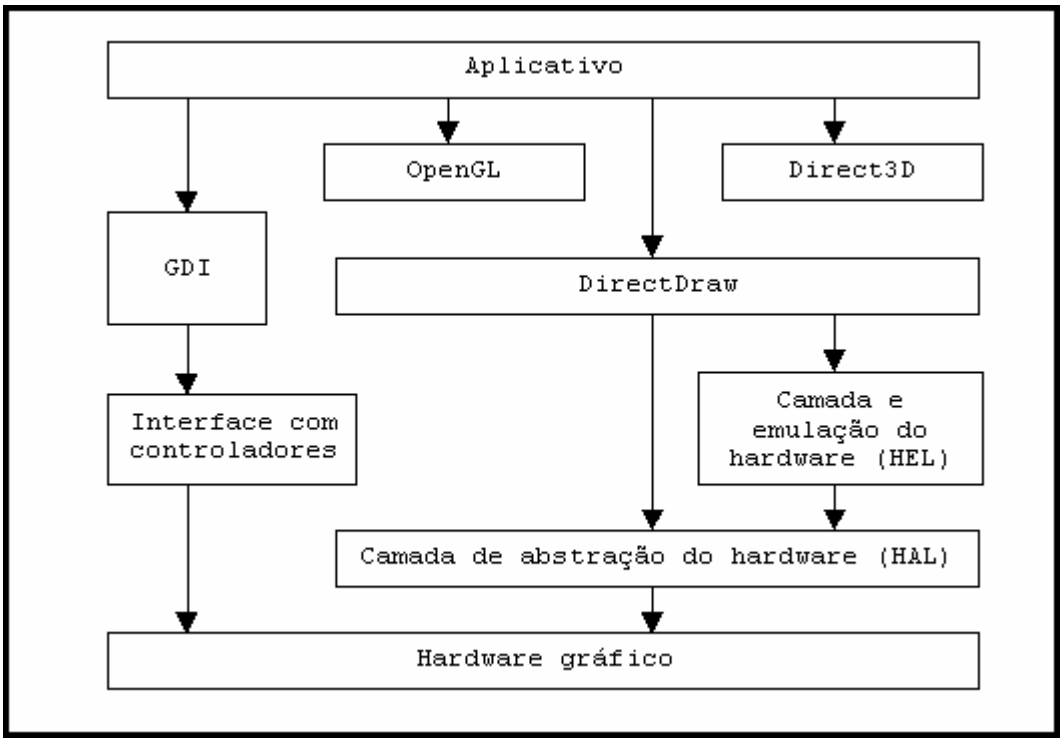
A linguagem C++ foi utilizada na implementação do protótipo pois permite a a metodologia de orientação a objetos. O protótipo foi implementado usando o ambiente de desenvolvimento Microsoft Visual C++ 6.0. Na implementação foi respeitada a notação hungariana, que também é utilizada na própria API (Application Programming Interface) do

Windows e do *DirectX*. Ela define regras para a nomeação de métodos e variáveis. Os nomes das classes também seguem essa notação, o que faz com que todas as classes comecem com a letra ‘C’ (maiúscula).

Para a reprodução de sons e gráficos e para enviar informações através de uma rede de computadores foi utilizada a biblioteca *DirectX*, em sua versão mais recente, número 7 (sete). Seu pacote gráfico, *Direct Draw*, abstrai a diferença existente entre os diferentes modelos de placa de vídeo. Ela oferece ainda a emulação de características que só estão presentes em placas com aceleração gráfica. A relação do *Direct Draw* com o sistema operacional Windows é ilustrado na figura 13.

A biblioteca permite ainda a manipulação de imagens (superfícies) de segundo plano. O próximo estado do jogo é desenhado na superfície de segundo plano. Após isso essa superfície é aplicada (colada) sobre a superfície primária fazendo assim a atualização da imagem de forma muito eficaz [KOV2000].

Figura 13 - Relacionamento entre serviços gráficos do Windows.



Fonte: adaptado de [FIL 2000].

As imagens utilizadas estão no formato *BMP* e foram criadas utilizando os aplicativos gráficos *Paintbrush* (Microsoft) e *Photo Shop Pro* (Jasc). Em jogos com objetivo comercial as imagens utilizadas são armazenadas em um único arquivo para dificultar sua extração e para utilizar algum método de compactação visando a economia de espaço no disco rígido. No protótipo cada imagem está armazenada em arquivo bitmap. Dessa maneira sua alteração e visualização é extremamente fácil.

Os modelos de nave utilizados são obtidos através de um arquivo texto que os descreve (ver figura 14). Essa descrição contém informações como sua velocidade e energia até as imagens gráficas utilizadas para sua representação. A representação de um modelo de nave segue um padrão. Esse padrão necessita de dois arquivos *BMP* para a nave, um com a imagem da nave sem movimento e o outro com movimento. Além disso, a descrição do modelo precisa definir alguns valores, como a velocidade máxima ou a resistência. Desse modo a inclusão de um novo modelo de nave ao jogo é simples, não necessitando gerar uma nova versão (compilação) do programa.

Figura 14 - Formato do arquivo que descreve um modelo de nave.

```
// define o nome do modelo
Name: Avenger

// a velocidade maxima da nave
Max Velocity: 10

// a aceleracao do modelo
Acceleration: 4

// a energia total
Energy: 20

// a capacidade de armazenar municao
Ammo space: 20

// o tamanho de cada imagem que representa a nave
cx: 84
cy: 80

// as imagens utilizadas pelo modelo
Image Stop: Avenger_ok.bmp
Image Fly: Avenger_Fly.bmp
Image explosion: Explosion.bmp

// informacoes da imagens
// imagens por linha
Image per row: 8
// quantidade de linhas
Image row count: 3
```

As fases e configurações do jogo também são modulares. Para altera-las é necessário alterar um arquivo texto que as define. Para criar uma fase é só criar um novo arquivo com as informações da mesma. Para cada tipo de arquivo texto há um diretório e uma extensão, segundo a tabela 2.

Tabela 2 - Arquivos de configuração.

Informação	Diretório	Extensão do arquivo
Modelos de naves	models	mod
Níveis ou fases	levels	lvl
Configuração do jogo	conf	Cfg

Existem itens espalhados pela fase, que podem aumentar a energia ou munição de uma

nave. Para definir uma fase deve-se informar seu tamanho, e a posição de cada item.

O protótipo utiliza todas as seções com que Lamothe [LAM1999] subdivide um jogo (seção 4.2). Primeiramente as bibliotecas são inicializadas. Após isso o jogador define que sessão deseja participar, assim como seu time, o modelo de sua nave e seu nome. Depois acontece o que Lamothe [LAM1999] chama de *Game Loop*, que é onde ocorrem as ações e atualizações do jogo. O *Game Loop* por sua vez realiza outras ações, segundo a seguinte ordem:

1. Verifica as teclas pressionadas pelo usuário;
2. Atualiza os jogadores controlados por IA, simulando seu raciocínio;
3. Manda pela rede, o estado atualizado do jogador, assim como os jogadores controlados pelo computador;
4. Recebe as informações dos outros jogadores;
5. Desenha todos os objetos do jogo.

Após o *Game Loop*, e conseqüentemente o termino da aplicação, é realizada a finalização das bibliotecas, liberando assim os recursos utilizados.

O protótipo passa por vários estados, como a tela de abertura, a tela de configuração do jogador e tela de ação do jogo. A metodologia utilizada por Lamothe [LAM1999] é caracterizada pelo uso de variáveis para controlar o estado atual do jogo, fazendo com que todo o código do jogo fique na mesma função. Esse método tende a dificultar o compreendimento do código fonte. No protótipo a classe `CGameLogic` possui um ponteiro para a função do jogo de acordo com o estado atual. Desse modo cada estado do jogo possui uma função. Portanto a alteração do ponteiro que indica a função atual do jogo é o suficiente para a alteração do próprio estado do jogo. Essa abordagem pode ser considerada como uma otimização, pois não são necessários testes repetitivos para saber em que estado está o jogo (figura 15).

Figura 15 - Alterando o estado atual do jogo.

```
int CGameLogic::StartupMain()
{
        ...

        int iKey = GetKeyPressed();

        if (iKey == VK_ESCAPE || iKey == VK_RETURN)
        {
                SAFE_DELETE(m_pStartupScreen);

                LoadShipModels();
                g_theGameEngine.SetGameMain(GameSelectModel);

                m_iState = STATE_PLAYING;

        }

        ...
}
```

Para utilizar as bibliotecas *DirectX* foram implementadas algumas classes com o intuito de simplificar sua utilização (seção 5.1), que muitas vezes é complexa.

As classes `CGameEngine` e `CGameLogic` são utilizadas através de uma variável global (uma para cada classe) pois são utilizadas extensivamente no jogo.

O desenvolvimento do protótipo teve seu inicio na construção da classe `CGameEngine` para manipulação e criação de imagens gráficas via a *DirectX*.

Para inicializar a biblioteca *Direct Draw* se faz necessário informar que janela o *Direct Draw* deverá utilizar. Depois disso deve-se informar o modo no qual deseja-se executar, que pode ser tela cheia (*fullscreen*) ou normal. Utilizando-se o modo fullscreen todo o poder da placa de vídeo pode ser utilizado, pois nenhum outro processo estará utilizando a mesma. Nesse modo pode-se ainda, definir a quantidade de cores apresentáveis ao mesmo tempo (*color depth*), e a resolução (quantidade de pixels).

Já no modo normal o poder da placa de vídeo é dividido entre os processos que estão executando e não é permitido alterar a *color depth* nem a resolução da tela.

O protótipo está implementado para trabalhar no modo *fullscreen*, embora a maioria de jogos existentes no mercado funciona nos dois modos. A resolução utilizada é 800x600 e o *color depth* é 16 bit ou 16 milhões de cores.

Ao terminar a inicialização do *DirectX*, a classe `CGameEngine` tem o buffer primário (representa a tela atual) e o secundário da tela (para desenhar a próxima tela). Todo desenho na tela é realizado por intermédio dessa classe. Pode-se observar, na figura 16, como se utiliza a classe `CMovingImage` para desenhar arquivos no formato *BMP* segundo sua posição no mundo.

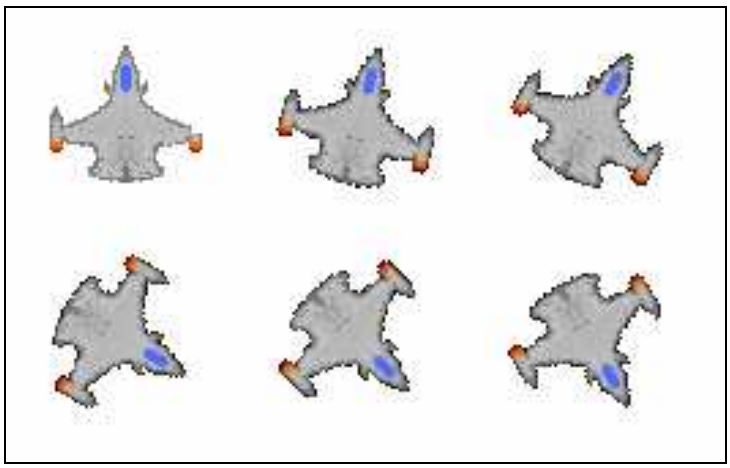
Figura 16 - Exemplo da utilização da classe CMovingImage.

```
void CMovingImage::DrawImage(int x, int y, int iImage, int iSurface)
{
        ...
        // desenha no buffer secundario
        HRESULT hr = g_theGameEngine.GetBackSurface()->Blt(&rectDest,
                                        m_lppSurface[iSurface], &rectSrc,
                                        DDBLT_KEYSRC | DDBLTFAST_WAIT, NULL);
}
```

O identificador g_`theGameEngine` é a instância global da classe `CGameEngine`. O método `GetBackSurface` retorna uma referência para o buffer secundário da tela.

Os objetos do protótipo são 2D e foram criados em arquivos no formato *BMP*. Para simular uma nave que está mudando de direção foram utilizados vários gráficos da mesma nave, com diferentes graus de rotação. No total foram utilizadas 24 imagens de cada nave, tendo uma diferença de 15^{o} entre elas. A figura 17 ilustra algumas dessas imagens.

Figura 17 - Algumas imagens 2D utilizadas no protótipo.



Com intuito de obter maior realismo na ação do jogo foram utilizados sistemas de partícula. Uma partícula pode ser descrita como um ponto no espaço euclidiano [NED2000]. Além disso ela possui uma cor, que pode ser alterada durante o tempo para produzir o efeito desejado. O exemplo clássico de um sistema de partículas é a modelagem de uma chama.

No protótipo foi utilizado um sistema de partícula simulando o fogo expelido pelos propulsores das naves. Cada partícula começa representando a cor vermelha e com o tempo se aproxima do preto até desaparecer. Essas partículas não possuem movimento. Já para simular os fragmentos de uma nave explodindo foram utilizadas partículas em movimento. Sua trajetória faz com que se distancie da nave (ponto original).

Para possibilitar um maior realismo e diversão, foram utilizados arquivos sonoros no formato *WAV*. Assim no momento em que duas naves se colidem, ou quando um dos cristais é capturado algum efeito sonoro é reproduzido (classe `CDirectSound`). A classe `CDirectSound` utiliza a biblioteca *Direct Sound* que é compatível com inúmeras placas de som, além de ter uma boa performance [LAM1999]. O código fonte que inicializa a biblioteca é demonstrado na figura 18.

Figura 18 - Inicialização da biblioteca Direct Sound.

```
BOOL CDirectSound::Initialize(HWND hwnd)
{
        HRESULT hr = DirectSoundCreate(NULL, &m_lpDirectSound, NULL);

        if (SUCCEEDED(hr))
        {
                hr = m_lpDirectSound->SetCooperativeLevel(hwnd, DSSCL_NORMAL);
        }
        return (SUCCEEDED(hr));
}
```

A biblioteca *Direct Sound* permite a reprodução de arquivos digitais ou sintetizados. Para isso deve se inicializar o buffer de som (*DirectSoundBuffer*) de acordo com o seu tipo, e chamar método que reproduz o buffer (ver figura 19).

Figura 19 - Método que reproduz arquivo *WAV*.

```
BOOL CDirectSound::PlayFX(int iIndex, BOOL bLoop)
{
        if (iIndex >= m_nFXCount)
                return FALSE;

        // volta a posicao inicial do buffer
        if (FAILED(m_pSoundFX[iIndex]->GetSoundBuffer()->SetCurrentPosition(0)))
                return FALSE;


        DWORD dwFlags = 0;

        // tocar infinitamente??
        if (bLoop)
                dwFlags = DSBPLAY_LOOPING;

        if (FAILED(m_pSoundFX[iIndex]->GetSoundBuffer()->Play(0, 0, dwFlags)))
                return FALSE;

        return TRUE;
}
```

Mas como a parte mais divertida do jogo é a possibilidade de enfrentar colegas que estão separados geograficamente, a opção de *multiplayer* foi projetada para utilizar o protocolo TCP/IP, o que permite sua utilização através da Internet (seção 2.2). O protótipo utiliza a biblioteca *Direct Play*, através da classe `CDirectPlay`, para possibilitar esse tipo de partida. Essa biblioteca permite que o meio de comunicação (modem, TCP/IP, IPX ou

serial) seja abstraído quando se está desenvolvendo. Segundo [KOV2000] essa biblioteca apresenta conceitos descritos a seguir.

Cada tipo de conexão é conhecido como provedor de serviço ou tipo de conexão. Ela permite a enumeração de tipos de conexão disponíveis. Para inicializar-se a biblioteca deve indicar o meio de conexão utilizado e em seguida verificar quais sessões de jogo estão abertas (figura 20).

Figura 20 - Inicializando a biblioteca Direct Play.

```
BOOL CDirectPlay::Initialize(LPCGUID lpguid)
{
        DPSESSIONDESC2  sessionDesc;
        int i;
        m_lpguid = (LPGUID)lpguid;

        HRESULT hr = CoCreateInstance(CLSID_DirectPlay, NULL, CLSCTX_INPROC_SERVER,
                                                        IID_IDirectPlay3A, (LPVOID*)&m_pDPlay);
        if (FAILED(hr))
                goto errorFound;

        m_pDPlay->EnumConnections(lpguid, ConnectionsCallback, this, 0);
        RELEASE(m_pDPlay);

        hr = CoCreateInstance(CLSID_DirectPlay, NULL, CLSCTX_INPROC_SERVER,
                                                        IID_IDirectPlay3A, (LPVOID*)&m_pDPlay);
        if (FAILED(hr))
                goto errorFound;

        // try to connect
        hr = m_pDPlay->InitializeConnection(m_pTCPip->m_lpConnection, NULL);
        if (FAILED(hr))
                goto errorFound;

        // enum the session running
        memset(&sessionDesc, 0, sizeof(sessionDesc));
        sessionDesc.dwSize          = sizeof(sessionDesc);
        sessionDesc.guidApplication = *lpguid;

        for (i=0; i < m_iSessionCount; i++)
        {
                m_sessions[i].Clear();
        }
        m_iSessionCount = 0;

        hr = m_pDPlay->EnumSessions(&sessionDesc, 0, (ENUM_SESSIONS)SessionsCallback,
                                        this, DPENUMSESSIONS_AVAILABLE  | DPENUMSESSIONS_ASYNC);
        m_bHostSession = FALSE;

        return TRUE;

errorFound:
        RELEASE(m_pDPlay);
        return FALSE;
}
```

O parâmetro passado ao método `Initialize` (figura 20) é do tipo `LPCGUID` que é

um ponteiro para o tipo de dado *GUID* (*global unique identifier*). *GUID* é um tipo de dado definido pela Microsoft e ocupa 128 bits. Seu objetivo é que não haja um valor igual a outro em todo mundo. Nesse método é passado um ponteiro para um *GUID* definido para o protótipo. Dessa maneira a biblioteca pode separar os dados transmitidos de vários jogos, pois cada um tem seu identificador global único.

Depois de escolhido o tipo de conexão é estabelecida uma conexão. Essa conexão é estabelecida segundo o provedor de serviço. No caso de uma conexão via modem, é necessário à indicação do número do telefone a ser discado.

Já no caso do protótipo que utiliza o tipo de conexão TCP/IP, é necessário informar o endereço IP da máquina onde existe uma sessão (quando não se utiliza uma rede local). Uma sessão é uma partida sendo disputada por jogadores. O *Direct Play* permite também um método para enumeração das sessões abertas. Após a escolha da sessão que se deseja participar é necessário criar um jogador.

Um jogador é representado por uma identificação única, um número obtido através da biblioteca *Direct Play*. Após esse último passo o jogador criado já pode participar da partida. Durante uma partida inúmeros pacotes de dados são enviados entre os participantes. Esses pacotes contêm informações relacionadas ao jogo em questão ou a mensagens padrões como: um novo jogador, um jogador que desistiu de jogar, entre outras. Para controlar as mensagens padrões, também conhecidas como de sistema, há um jogador conhecido como *host* [KOV2000], ou servidor, que recebe o pacote de informação de cada jogador e faz a sua distribuição entre os outros participantes da partida.

Quando se deseja jogar sozinho, ou há um número impar de jogadores pode-se utilizar *bots* (*robots*) para completar a partida, que são implementados na classe `CAIPlayer`. O *bot* pode ser configurado para executar uma função, como perseguir um certo membro da outra equipe, ficar no ataque ou na defesa. Mas para que o *bot* tenha uma certa inteligência, ele tem memória, guardando os locais de energia, arma, do cristal inimigo entre outras coisas. Além disso, ele é capaz de elaborar um plano para atacar o inimigo no momento certo. Para isso foram utilizadas técnicas de planejamento e memória, provenientes da área de Inteligência Artificial.

Os planos são controlados pela classe `CPlanManager`. Ela contém um conjunto de instâncias da classe `CPlan`. A classe `CPlan` contém um mecanismo básico para que `CPlanManager` possa decidir que plano utilizar (figura 21).

Figura 21 - Executando planos.

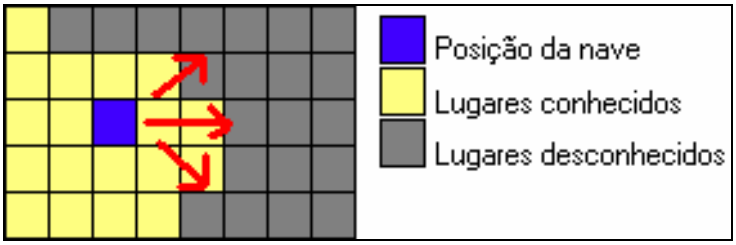
```
void CPlanManager::Run()
{
        // se tem algum plano executando
        if (m_nRunningPlan != NO_RUNNING_PLAN)
        {
                m_pPlans[m_nRunningPlan]->Run();

                // verifica se ele ainda pode ser executado
                if (!m_pPlans[m_nRunningPlan]->IsSatisfied())
                        m_nRunningPlan = NO_RUNNING_PLAN;
        }
        else
        {
                // encontra um novo plano para o estado atual do mundo
                for (int i=0; i < m_nPlanCount; i++)
                {
                        if (m_pPlans[i]->IsSatisfied())
                        {
                                m_nRunningPlan = i;
                                return;
                        }
                }
        }
}
```

Enquanto uma nave controlada por IA voa pela fase seus conhecimentos são atualizados. Os itens que são visíveis na posição atual da nave são armazenados na base de conhecimento, tornando mais fácil sua busca quando for necessária. A nave controlada possui a mesma visão que a de um jogador humano, ou seja equivalente ao tamanho da tela. Desse modo a nave controlada não tem vantagem sobre um humano. Além da memória para os itens espalhados pelo espaço existe ainda a memória destinada aos locais da fase já conhecidos.

No momento em que a nave simulada precisa de algum item e o mesmo não se encontra em sua base de conhecimento, ela começa a procurar. Essa procura irá acontecer nos pontos da fase por onde ela ainda não passou. Para armazenar os locais por onde a nave já conheceu a fase foi dividida em pequenos quadrados de 40 unidades de medida (figura 22). Para marcar um quadrado como conhecido, a nave precisa ter uma visão completa do mesmo a partir do ponto em que ela se encontra. Uma vez marcado como conhecido o quadrado não

volta ao estado inicial (desconhecido).

Figura 22 - Memória da fase



Essas memórias são utilizadas pelos planos. Um plano é composto de passos para terminar sua execução (seção 3.1). O plano para roubar o cristal inimigo foi implementado tendo como primeiro passo encontrar o cristal inimigo. Ao encontrá-lo é feita uma análise das naves que o cercam. Se o time da nave controlada por IA estiver em menor número ela espera um momento mais propício para o ataque. Se o número de naves for equivalente ela parte ao ataque. Em caso de não existência de naves inimigas ao redor do cristal o mesmo é roubado. O método que realiza esse raciocínio na classe `CAIPlayer` é ilustrado na figura 23.

Figura 23 - Raciocínio através da classe `CAIPlayer`

```
void CAIPlayer::Move()
{
        if (GetState() >= STATE_INITIAL_DEAD)
        {
                if (!(--m_iFramesDead))
                {
                        ReturnToGame();
                }
        }
        else
        {
                // atualiza a memoria
                m_aiMemory.itemMemory.UpdateVisibleItens(this);

                m_aiMemory.lvlMemory.SetVisitedPoint(GetPosition().x,
                                                     GetPosition().y);

                // decide o proximo movimento
                m_pPlanManager->Run();

                CPlayer::Move();
        }
}
```

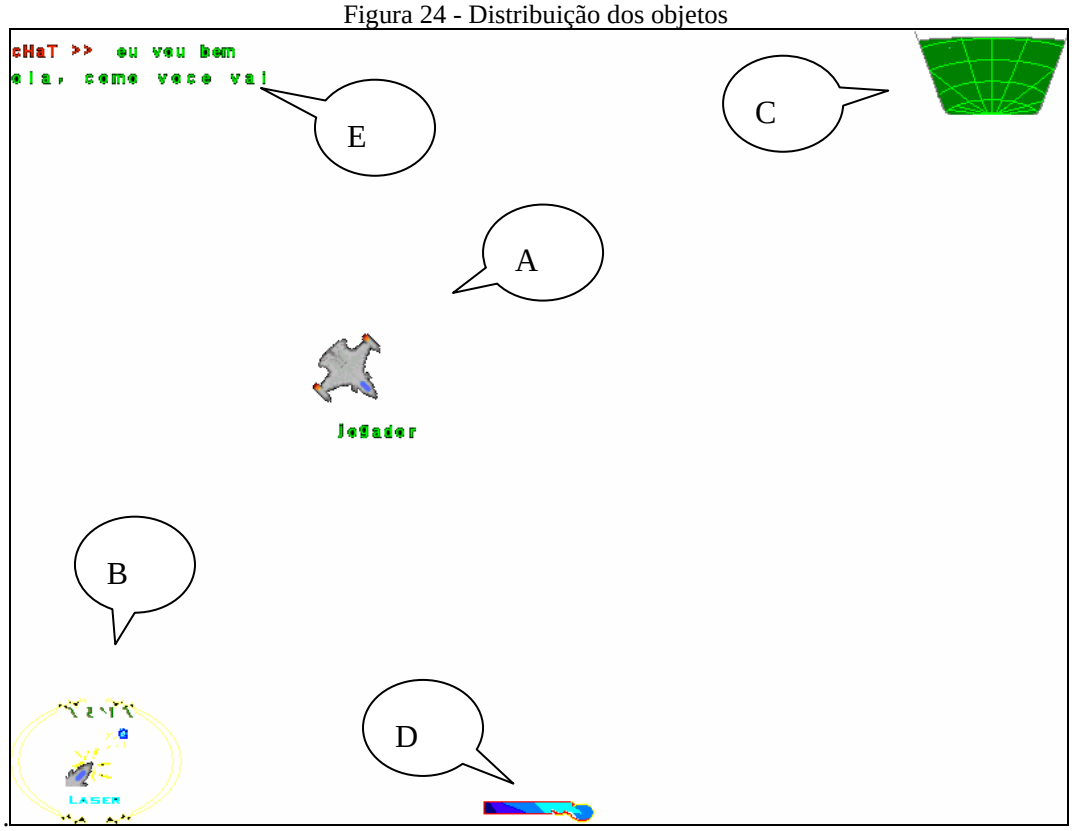
Após o roubo do cristal a nave tenta voltar até a posição de seu cristal, para assim capturar o cristal (ver seção 5.3). No meio da execução de algum plano é possível que haja

uma interrupção, ou seja, um ataque inimigo. Nesse caso a nave tenta destruir seu obstáculo e prosseguir a execução normal do plano.

Existem inúmeros métodos que podem aperfeiçoar o comportamento das naves controladas por IA. Watson [WAT1996] utiliza algoritmos genéticos para permitir que uma nave altere a sua estratégia durante o jogo, buscando seguir o modelo de comportamento e ações das naves que estão em melhor situação na partida. Poderia ainda ser utilizada um mecanismo de comunicação entre as naves permitindo um compartilhamento de memória. Desse modo uma nave poderia "comunicar" sua companheira a localização de algum item.

O protótipo tem a distribuição de objetos segundo a figura 24.

Figura 24 - Distribuição dos objetos



O objeto indicado pela letra A representa a nave do jogador, que fica sempre no centro da tela. O objeto da letra B informa ao jogador a arma que está em uso atualmente, assim como a munição restante. O radar, que é o objeto C, mostra onde o jogador está posicionado em relação ao mundo, bem como a posição de seus inimigos e seus companheiros de equipes,

diferenciados pela cor. A letra D indica a quantidade de energia restante da nave do jogador, que ao alcançar o valor 0 (zero) ocasionará a destruição da mesma. E por fim o objeto apontado pela letra E indica a área destinada ao *chat* e a informações de possíveis eventos do jogo.

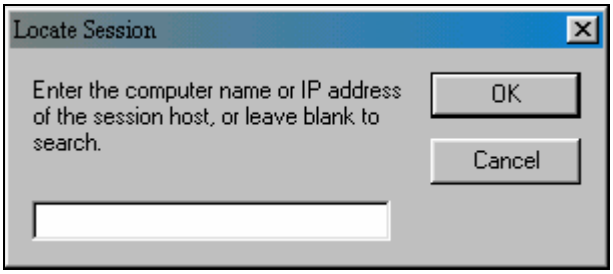
O universo é um retângulo, onde estão distribuídos asteróides, itens que aumentam a energia e resistência das naves, bem como armas. O jogo chegará ao fim após um certo número de pontos ou após um tempo pré-determinado.

5.3 FUNCIONAMENTO DO PROTÓTIPO

Para utilizar o jogo se faz necessário a instalação das bibliotecas *DirectX* em sua versão 7.

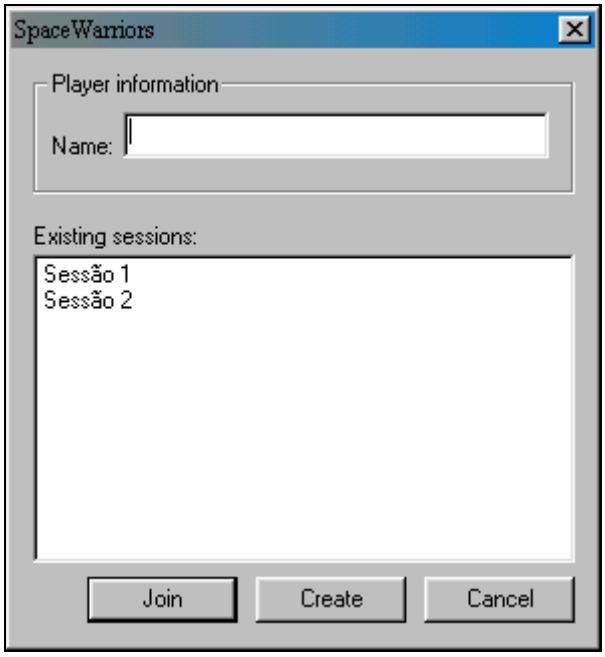
Para se jogar uma partida é necessário que o jogador crie uma partida ou participe de uma sessão em andamento. Para participar de uma sessão em andamento é necessário o conhecimento do endereço IP da máquina servidora (onde foi criada uma sessão de jogo). Essa informação é dada na tela ilustrada pela figura 25. Essa tela pertence ao *DirectX*, e contém informações e opções de acordo com o meio de comunicação selecionado.

Figura 25 - Endereço IP do servidor.



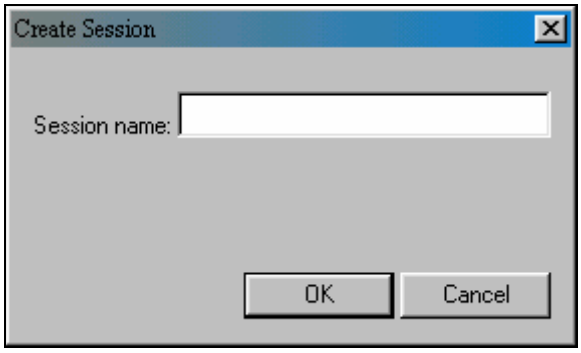
Se a máquina servidora for local as sessões abertas são mostradas numa lista na tela seguinte (figura 26). Nessa tela o usuário informa seu nome, e pode decidir criar uma sessão a seu gosto, ou então participar de uma sessão já criada.

Figura 26 - Sessões abertas.



Se o usuário decidir criar uma sessão ele deve informar o nome da mesma (figura 27). Após esse passo o jogo começa.

Figura 27 - Criando uma sessão.

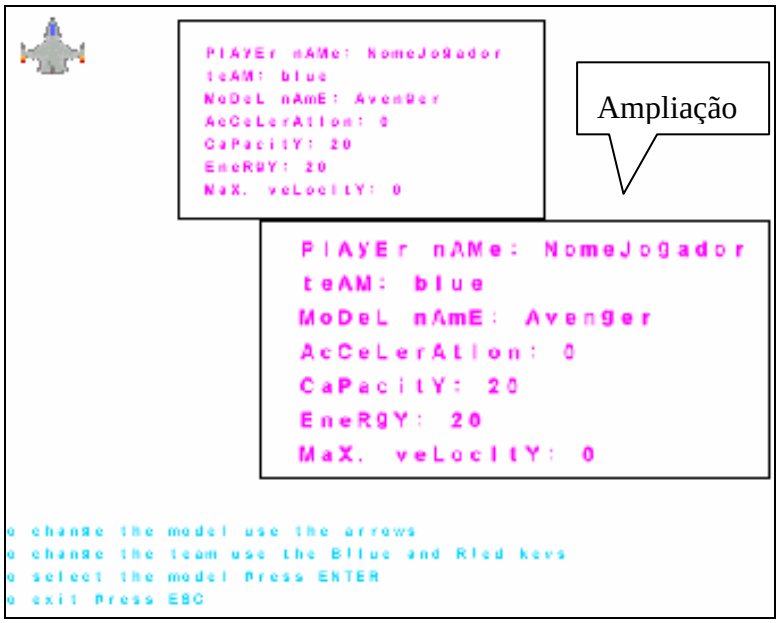


Nesse momento a tela de apresentação do protótipo (figura 28) aparece bastando teclar ENTER para continuar as configurações.

Figura 28 - Tela de abertura do jogo (cores invertidas).

Para começar o jogo o usuário seleciona em que time deseja jogar e que modelo de nave irá utilizar (figura 29). Para jogar no time vermelho é necessário teclar a letra ‘r’ (*red*) e para participar no time azul a letra ‘b’ (*blue*). Para escolher o modelo de nave utiliza-se as setas para esquerda e direita. Os modelos de nave que podem ser selecionados são equivalentes aos arquivos com a extensão *mod* localizados no diretório destinado aos modelos (ver seção 5).

Figura 29 - Definição do time e modelo da nave (cores invertidas).

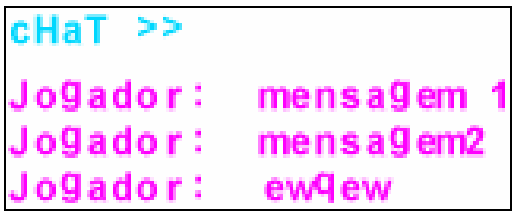


Depois das configurações o jogo começa. Em qualquer momento da partida o jogador pode mandar uma mensagem aos outros jogadores. Essa mensagem pode ser destinada aos

jogadores de sua equipe ou aos jogadores adversários. O padrão é enviar a mensagem a todos os jogadores. Para utilizar o mecanismo de chat deve-se teclar F1. Desse modo a tela normal de jogo apresenta em seu topo uma seção que demonstra a mensagem sendo editada. Ao terminar a mensagem digita-se ENTER para confirmar sua emissão aos demais jogadores. Ela irá aparecer do mesmo modo como as outras mensagens e eventos do jogo são mostrados (figura 30).

Para enviar uma mensagem aos jogadores da própria equipe deve-se usar o identificador *team* antes da mensagem. Por exemplo: para enviar a mensagem “Oi” para os membros da própria equipe deve-se digitar: “*team* Oi <ENTER>”. Do mesmo modo o identificador *enemy* é usado para enviar mensagens somente aos jogadores adversários.

Figura 30 - Área de *chat* do jogo (cores invertidas e ampliado).

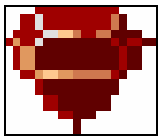


Ao ficar em modo de chat o jogador não poderá movimentar-se ou disparar projéteis, ficando dessa forma indefeso. Para permitir que os outros jogadores percebam isso, e possam decidir se irão atacá-lo ou não, existe um indicador no canto superior esquerda da nave que pode ser observado na figura 31.

Figura 301 - Identificador de chat.

O objetivo principal do jogo é fazer o maior número de pontos por equipe, que se obtém de forma mais eficaz capturando o cristal inimigo. Para capturar o cristal, é necessário roubar o cristal inimigo e leva-lo até o local de origem do cristal da própria equipe, tendo este que estar em sua posição. O cristal é ilustrado na figura 32. Esse tipo de jogo é conhecido no Quake2 [IDS2000] como *Capture the Flag (CTF).*

Figura 32 - Cristal do time vermelho (ampliado).



Se nesse meio tempo o cristal do seu time for roubado deve se destruir a nave que o possui. Ao destruir essa nave, o cristal fica na posição em que se encontra. Ele volta a posição original no momento em que algum jogador do seu time tocá-lo. Mas, capturar o cristal inimigo não é o único método de se acumular pontos. Os pontos são adquiridos conforme a tabela 3.

Tabela 3 - Relação de pontos por ação.

Ação	Pontos
Destruir uma nave inimiga	2
Destruir a nave inimiga que carrega o cristal	3
Roubar o cristal inimigo	1
Recuperar o cristal do time	3
Capturar o cristal inimigo (voltar até a base com outro cristal)	8 para a nave que capturou e 1 para cada jogador do seu time

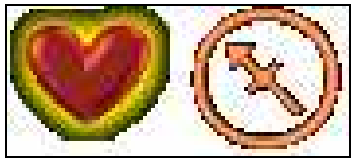
Os pontos podem ser consultados a qualquer momento da partida para isso deve-se teclar a letra ‘X’. Desse modo o placar (figura 33) ficará visível no centro da tela. Para que o placar desapareça deve-se teclar ‘X’ novamente.

Figura 313 - Placar da partida.

Para controlar a nave deve-se utilizar as setas. A seta para cima acelera a nave enquanto que a seta para baixo diminui a velocidade. A seta para esquerda gira a direção da nave para esquerda ao contrário da seta da direita. Para disparar um tiro utiliza-se a tecla CONTROL.

Conforme a ação do jogo ocorre a nave pode perder energia ou gastar sua munição. Ambos podem ser ressarcidos, bastando para isso pegar o item correspondente, os quais estão distribuídos pela fase. Suas representações gráficas são ilustradas na figura 34.

Figura 34 - Energia e Munição (ampliadas).



Ao ficar sem energia a nave explode. Para voltar ao jogo basta pressionar ENTER que o jogador será posicionado em um local aleatório da fase. Para sair do jogo basta apertar ESC e confirmar pressionando a tecla ‘Y’ (*yes*).

6 CONCLUSÕES

A seguir apresentam-se as conclusões finais, dificuldades encontradas no decorrer do trabalho, limitações do protótipo e sugestões para trabalhos futuros.

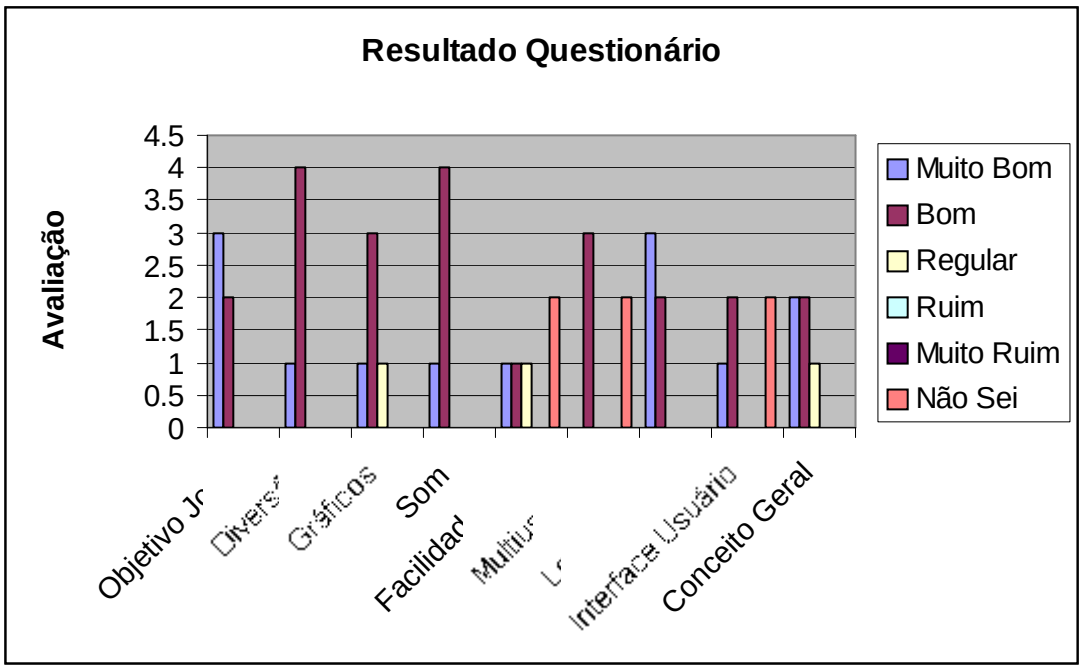
6.1 RESULTADOS ALCANÇADOS

O protótipo de ambiente de entretenimento 2D cumpriu com os objetivos propostos e abordou as características especificadas, como a utilização de técnicas multimídia e Inteligência Artificial.

Os testes e avaliações realizados revelaram o gosto dos jogadores em relação ao jogo. Eles opinaram em relação ao objetivo do jogo, a qualidade das artes gráficas e produção musical, entre outros, dando sugestões para uma maior diversão.

Foi elaborado um questionário (anexo 2) para avaliação do protótipo. Esse questionário foi aplicado a 6 (seis) pessoas logo após uma partida. Todos entrevistados têm conhecimentos em jogos *multiplayer*, tornando se assim essa uma pesquisa qualitativa. O resultado do questionário é ilustrado pela figura 35.

Figura 35 - Resultado do questionário.



As maiores deficiências apontadas pelo questionário são a interface com o usuário e a facilidade de uso. É necessária uma nova elaboração das telas de escolha de nave e configuração assim como também um mecanismo de ajuda durante a utilização do protótipo. Dentro do propósito do trabalho o resultado é satisfatório, mas para uma possível comercialização são necessárias melhoras em todos as áreas abordadas pelo questionário.

A área de jogos é uma das áreas do desenvolvimento de software mais difíceis de se obter sucesso. Todos os anos diversas empresas fracassam em sua sobrevivência. A habilidade em programação não é o único requisito para o sucesso de um projeto. O trabalho em equipe, o planejamento e a qualidade do desenvolvimento do projeto são básicos para o sucesso de um jogo. A importância para essa área vem aumentando significativamente o que resultou em cursos com o objetivo de formar profissionais para o desenvolvimento de jogos em todas as suas etapas. Entre as mais conhecidas estão: *Full Sail Real World Education* [FUL2000] e *Digi Pen Institute of Technology* [DIG2000].

6.2 DIFICULDADES ENCONTRADAS E LIMITAÇÕES

Para a utilização do protótipo é necessária a existência da biblioteca DirectX, em sua versão 7, cuja distribuição é gratuita.

O protótipo não executa em máquinas que utilizem o sistema operacional Windows NT, pois a biblioteca *DirectX* parou de ser distribuída para esse tipo de sistema operacional em sua versão 3.0. O tratamento da perda do controle total do *Direct Draw* não foi realizado, resultando numa falha de execução ao trocar de aplicativo (ALT + TAB).

6.3 EXTENSÕES

Melhoria da interface gráfica com usuário permitindo o uso do mouse para selecionar e configurar a partida. Permitir que o usuário configure os controles conforme seu desejo, podendo assim utilizar um *joystick* ou *mouse*.

Possibilitar um mecanismo de ajuda durante a utilização do protótipo, disponibilizando informações sobre a utilização do mesmo.

Implementar uma biblioteca para reproduzir arquivos no formato *MP3* que apresente uma performance aceitável para um jogo. Implementar a simulação de colisões entre naves observando aspectos da Física.

Avançar na utilização das técnicas de IA para tornar o jogador controlado pelo computador mais competitivo.

REFERÊNCIAS BIBLIOGRÁFICAS

[BAR2000] BARUTCUOGLU, Zafer. **Writing intelligent games**. Endereço Eletrônico: http://www.gamedev.net/, 2000.

[CLA2000] CLARK, Ryan. **Game Design - Tips for the one-man army**. Endereço Eletrônico: http://www.amadev.net/articlegd1.html : 2000.

[DIG2000] DIGIPEN, Institute of Technology. **DigiPen Institute of Technology**. Endereço Eletrônico: http://www.digipen.edu : 2000.

[UOL2000] ULTIMA, Online Home. **Ultima online home**. Endereço Eletrônico: http://www.uo.com : 2000.

[FER1985] FERREIRA, Aurélio Buarque de Holanda. **Minidicionário da língua portuguesa**. Rio de Janeiro : Editora Nova Fronteira, 1985.

[FIL2000] FILHO, Wilson de Pádua Paula. **Multimídia – Conceitos e Aplicações**. Editora LTC. 2000.

[FUL2000] FULLSAIL, Real World Education. **Full Sail Real World Education**. Endereço Eletrônico: http://www.fullsail.com/index1.html, 2000.

[FUR1998] FURLAN, José David. **Modelagem de objetos através da UML**. São Paulo : Makron Books, 1998.

[HOW2000] HOWER, Geoff. **A pratical guide to building a complete game AI.** Endereço Eletrônico: http://www.gamedev.net/reference/programming/ai/article784.asp, 2000.

[IDS2000] Id Software, Id. **Id Software**. Endereço Eletrônico: http://www.idsoftware.com, 2000.

[KOV2000] KOVACH, Peter J.. **Inside Direct3D**. Washington : Microsoft Press, 2000.

[LAM1999] LAMOTHE, André. **Tricks of the Windows game programming gurus**. Indianapolis : Sams, 1999.

[LAR2000] LARAMÉE, François-Dominic. **How to screw up a perfectly good game company in ten easy steps**. Endereço Eletrônico: http://www.gamedev.net/reference/articles/article910.asp, 2000.

[MAS2000] MASON, McCuskey. **Lone Wolf Killers Part I: The design phase**. Endereço Eletrônico: http://www.gamedev.net/reference/articles/article913.asp : 2000.

[MIC1999] MICHAEL, David. **Designing for online communities**. Endereço Eletrônico: http://www.gamedev.net/reference/design/archive/article889.asp, 1999.

[MIC2000] DIRECTX, Microsoft. **DirectX API**. Endereço Eletrônico: http://www.microsoft.com/directx, 2000.

[MUL1997] MULLER, Pierre-Alain. **Instant UML**. Birmingham : Wrox, 1997.

[NED2000] NEDEL, Luciana Porcher. **Animação por computador: evolução e tendências**. VIII ERI - Escola de Informática da SBC Sul. Pág. 87-114, 15-19 de maio de 2000.

[RAB1996] RABUSKE, Renato Antônio. **Inteligência artificial**. Florianópolis : Ed. da UFSC, 1996.

[ROG1997] ROGERSON, Dale. **Inside COM**. Washington : Microsoft Press, 1997.

[RUS1995] RUSSEL, Stuart; NORVIG Peter. **Artificial Inteligence : a modern approach**. New Jersey : Prentice-Hall, 1993.

[WAT1996] WATSON, Mark. **AI Agents in virtual reality worlds**. New York : John Wiley & Sons, 1996.

[XAU2000] XAUDIO, Sound Library. **XAudio sound library**. Endereço Eletrônico: http://www.xaudio.com : 2000.