CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Jogos Digitais

Jair Bortolucci Júnior

Desenvolvimento de um jogo baseado em um simulador de sistema de produção de bovinos de corte

Americana, SP

2014

CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Jogos Digitais

Jair Bortolucci Júnior

Desenvolvimento de um jogo baseado em um simulador de sistema de produção de bovinos de corte

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Jogos Digitais, sob a orientação Me. Vitor Brandi Junior

Área de concentração: Jogos Digitais

Americana, S. P.

2014

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS

Dados Internacionais de Catalogação-na-fonte

Bortolucci Junior, Jair

B748d Desenvolvimento de um jogo baseado em um simulador de sistema de produção de bovinos de corte. / Jair Bortolucci Junior. – Americana: 2014.

44f.

Monografia (Graduação em Tecnologia em Jogos Digitais). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.

Orientador: Prof. Me. Vitor Brandi Junior

1. Jogos digitais I. Brandi Junior, Vitor II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.6

Jair Bortolucci Júnior

Desenvolvimento de um jogo baseado em um simulador de sistema de produção de pecuária de corte

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Jogos Digitais pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Jogos Digitais

Americana, 5 de dezembro de 2014.

Banca Examinadora:

Vitor Brandi Júnior (Presidente)
Graduado
Faculdade de Tecnologia de Americana

Gustavo Carvalho Gomes de Abreu (Membro)
Graduado
Faculdade de Tecnologia de Americana

Adnan Bakri (Membro)
Especialista
Faculdade de Tecnologia de Americana

RESUMO

Este trabalho objetivou o desenvolvimento de um jogo digital baseado em simulação computacional. No contexto da área de pecuária de corte foi desenvolvido um jogo que permite ao usuário simular algumas das atividades gerenciais do sistema produtivo de manejo de rebanhos. Para tanto, fora utilizado o *framework* de simulação MaCSim, desenvolvido no Laboratório de Matemática Computacional da Embrapa e um modelo advindo do projeto PECUS. O jogo foi integrado ao simulador e pôde representar os dados numéricos da simulação por meio das animações, possibilitando a visualização dos fenômenos gerados pelos modelos matemáticos de forma mais interativa.

Palavras Chave: Jogo baseado em simulação; *Serious Games*; Visualização de Simulação

ABSTRACT

This paper aimed the development of a digital game-based computer simulation. Within the context of beef cattle was developed a game that allows the user to simulate some of the management activities of the productive system of livestock management. Therefore, had been used the simulation framework MaCSim, developed at the Laboratory of Computational Mathematics from Embrapa, arising from the PECUS model's project. The game was integrated into the simulator and could represent the numerical simulation data through animations, make possible the visualization of phenomena generated by mathematical models in a more interactive way.

Keywords: *Game-based simulation; Serious Games; Simulation and Visualization*

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

Em consequência da dinâmica de sobrevivência que é imposta a humanidade, o estilo de vida do ser humano está cada vez mais acelerado, implicando em rotinas repetitivas e estressantes. Nesse sentido, momentos de lazer tornam-se indispensáveis para a saúde do corpo e da mente. Essa busca por divertimento e momentânea fuga da realidade tornou aos poucos o Jogo Digital uma forma de entretenimento poderosa. Os Jogos Digitais são tecnologias capazes de imergir seus usuários em mundos fantásticos e situações improváveis, proporcionando maior imersividade em contraste a outros meios de lazer, como livros ou filmes. O jogo é capaz de conjuntar as histórias dos livros e cenas dos filmes além de possibilitar o comando sobre o desenvolvimento da história.

Aos poucos, os Jogos Digitais caminharam para fora do âmbito do entretenimento, demonstrando potencial para outras finalidades. Educadores encontraram nos Jogos Digitais a possibilidade de transferir conhecimento de forma interativa e divertida, um fato conhecido por ludicidade. Isso fez com que os jogos se desenvolvessem como ferramentas educacionais, ampliando suas áreas de aplicação.

Nesse contexto, os Jogos Digitais passaram a ser utilizados como ferramentas de treinamento, virtualizando variadas situações reais, em que o treinamento poderia ser custoso, arriscado ou até mesmo anti-ético. Por esse fato, essa ferramenta de ensino, hoje chamada de *Serious Game*, cada vez mais é utilizada em diversos contextos pedagógicos.

A adoção de Jogos Digitais como meio de comunicação de conhecimento deve possuir correspodência com a realidade. Para tanto, o envolvimento de Jogos Digitais com modelos matemáticos que aproximam o funcionamento de um sistema real emergiu como solução para essa proposta.

1.2 OBJETIVOS GERAIS

O objetivo deste trabalho foi desenvolver um jogo digital baseado em um simulador computacional com a finalidade de elucidar conceitos relacionados ao

processo de integração entre um jogo e um simulador. Por meio da integração entre o jogo e o simulador é pretendido verificar a possibilidade de virtualizar um modelo matemático e transparecê-lo por meio da animação/visualização do jogo.

1.3 OBJETIVOS ESPECÍFICOS

Os objetivos propostos são:

- Estudar conceitos sobre Modelagem Matemática, Simulação Computacional, *Serious Games*, sobre o simulador MaCSim desenvolvido na Embrapa e sobre o modelo de sistema produtivo de pecuária de corte PECUS.
- Desenvolver um Jogo Digital em linguagem C++ que virtualiza um ambiente de pecuária de corte.
- Integrar o jogo desenvolvido ao simulador MaCSim.

1.4 JUSTIFICATIVA

Como afirma Zee et al (2012), o desenvolvimento de Jogos Digitais baseados em simuladores computacionais demonstra potencial pedagógico, no entanto ainda é intrincado e existe pouco direcionamento para tal atividade. Tendo em vista o enunciado de Zee et al, a importância deste trabalho se evindência no direcionamento e demonstração do processo de integração entre um jogo digital e um simulador computacional.

O fato do jogo simular um ambiente de pecuária de corte se justifica pelo estágio realizado na Embrapa, no qual foi possível constatar a escasses de *Serious Games* na agricultura.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo enuncia os conceitos necessários para o entendimento deste trabalho, apresentando as áreas de Modelagem e Simulação Computacional, *Serious Games e o framework* MaCSim e modelo PECUS.

2.1 MODELAGEM E SIMULAÇÃO COMPUTACIONAL

O presente tópico introduzirá conceitos básicos sobre modelagem e simulação computacional. Primeiramente, o conceito de sistemas e simulação computacional é apresentado. Em seguida, é enunciado o processo de modelagem de sistemas.

2.1.1 CONCEITO DE SISTEMA E SIMULAÇÃO COMPUTACIONAL

Para o entendimento de simulação computacional é importante o conceito de sistema usado comumente dentro desse contexto. Segundo Forrester (1968), um sistema pode ser entendido como um conjunto de partes que interagem umas com as outras buscando um mesmo objetivo. Para exemplificar, pode-se imaginar um automóvel, que é composto por diversas partes (motor, pneu, volante etc). Todas as partes funcionando juntas devem convergir para o objetivo de prover transporte. Um sistema pode também conter pessoas ou fenômenos físicos e se relacionar com outros sistemas (FORRESTER, 1968).

Nesse sentido, muitas das atividades efetuadas por humanos podem ser entendidas como sistemas, possibilitando que os estudos individuais de cada atividade possam ser inter-relacionados e entendidos como um todo. Esse estudo do relacionamento das partes permite obter informações importantes que dificilmente poderiam ser inferidas analisando-se cada parte individualmente. No entanto, alguns sistemas podem ser difíceis de serem observados ou serem muito complexos (FORRESTER, 1968).

É nesse ponto que a simulação computacional emerge como uma ferramenta poderosa para auxiliar no estudo de sistemas. De acordo com Kelton e Sadowski (1998), simulação computacional refere-se a métodos que possibilitam o estudo de sistemas do mundo real por meio de avaliação numérica, utilizando programas computacionais que imitam os processos e características desses sistemas, na maioria das vezes, ao longo do tempo. A partir dessa abordagem, podemos inferir que uma análise realizada por meio de simulação computacional pode ser testada diversas vezes com diferentes parâmetros, a fim de que se encontre o melhor resultado para ser seguido na prática. Isso implica em muitas outras vantagens como destaca Freitas Filho (2001):

- O estudo por meio de simulação pode proporcionar aos analistas um nível de detalhamento muito alto sobre o objeto de estudo, permitindo a visualização de comportamentos sutis.
- O comportamento do sistema por meio de simulação pode ser visualizado por animação.
- Simular dinâmica de sistemas pode economizar tempo e recursos financeiros.

2.1.2 MODELAGEM

A modelagem de sistemas é parte fundamental no processo de simulação. O presente tópico enuncia os principais aspectos no processo de modelagem, a fim de que se compreenda essa atividade de forma geral.

2.1.2.1 MODELAGEM DE UM SISTEMA

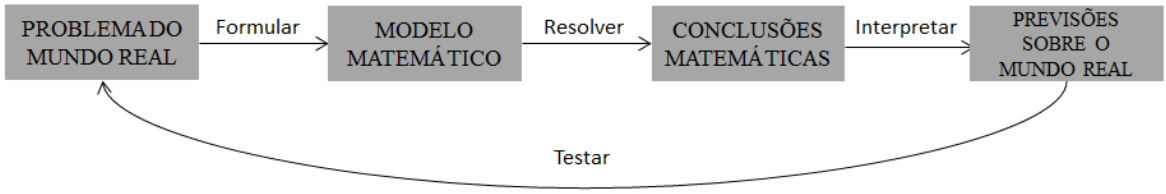
Para que o sistema real seja simulado computacionalmente existe uma etapa no processo da simulação que consiste na modelagem do sistema. Modelar é uma atividade que envolve a observação, estudo e posterior descrição dos aspectos essenciais do sistema a fim de que o modelo possa representar, de forma simplificada, os comportamentos que ocorrem no sistema real (FREITAS FILHO, 2001). O nível de detalhamento do modelo depende diretamente dos objetivos que pretendem ser atingidos, já que um sistema real possui muitas variáveis.

Comumente, os sistemas são modelados por meio de ferramentas matemáticas, resultando em modelos matemáticos.

2.1.2.2 MODELOS MATEMÁTICOS E PROCESSO DE MODELAGEM

De acordo com Stewart (2010), um modelo matemático é um conjunto de equações e/ou inequações que descrevem o comportamento de um fenômeno do mundo real. O processo para descrever um modelo matemático está representado na figura 1.

Figura 1 – Processo de modelagem matemática



Fonte – STEWART, 2010, p.15

Como ilustra a figura 1, o primeiro passo no processo de modelagem é tornar o sistema passível de ser representado matematicamente. Todas as variáveis do sistema são analisadas e associadas com hipóteses já formuladas ou fenômenos físicos que influenciam no funcionamento do sistema. Por meio de ferramentas matemáticas, equações que relacionam as variáveis são formuladas, compondo o modelo matemático. Com o modelo formulado, são aplicados métodos matemáticos para obter resultados numéricos (STEWART, 2010).

Os resultados numéricos por sua vez serão interpretados e relacionados com fenômenos reais. Por fim, o modelo é submetido a métodos estatísticos para ser comparado com dados do sistema real para verificar se ele está correspondendo com exatidão ao fenômeno estudado (STEWART, 2010).

2.1.2.3 EQUAÇÕES DIFERENCIAIS

Modelos matemáticos, em sua maioria, são representados por equações diferenciais. Equações diferenciais são equações em que a função é a incógnita e a algumas de suas derivadas (taxa de variação instantânea) são conhecidas (STEWART, 2006). Sua solução pode ser obtida analiticamente ou por métodos de aproximação numérica, tais como método de Euler ou método de Runge-Kutta[1]. Por representarem sistemas complexos, modelos matemáticos formalizados em equações diferenciais raramente possuem solução analítica, remetendo a necessidade da aplicação de métodos numéricos (GARCIA, 1997). Um exemplo de modelo matemático no formato de equação diferencial são as equações de Lotka-Volterra.

As equações de Lotka-Volterra são pares de equações diferenciais que modelam o comportamento de presas e predadores coexistindo em um mesmo ambiente (DAS;GUPTA, 2011), demonstradas a seguir:

Figura 2 – Equações de Lotka-Volterra

$$\frac{dx}{dt} = ax - bxy \quad (1)$$

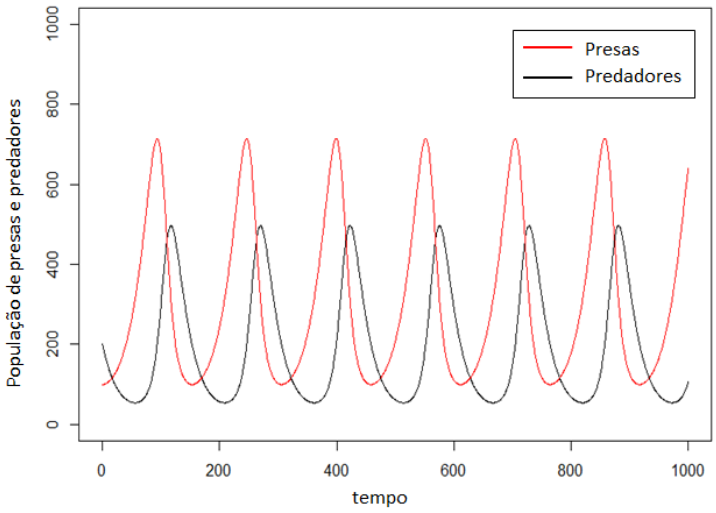
$$\frac{dy}{dt} = cxy - dy \quad (2)$$

Fonte: Elaborada pelo autor

A equação 1 representa o número de presas ao longo do tempo e 2 o número de predadores, os coeficientes a, b, c, d são constantes. Para resolução desse modelo é aplicado o método de integração numérica de Euler. Os resultados são apresentados na Figura 3.

[1] O método de Runge-Kutta é um método de integração numérica para aproximação da solução de equações diferenciais ordinárias.

Figura 3 – Gráfico que representa os resultados numéricos do modelo presa-predador



Fonte: Elaborada pelo autor

Pode-se observar a partir da plotagem dos resultados, que o número de predadores e presas varia inversamente ao crescimento do outro, visto que a ausência de predadores favorece o crescimento das presas e a grande população de presas proporciona o crescimento da população de predadores.

2.2 *SERIOUS GAMES*

Este tópico descreve pontos importantes para o entendimento de *Serious Games* de uma forma geral, abordando aspectos conceituais e técnicos. Primeiramente é apresentado um conceito geral seguido de um breve histórico. Em seguida são apresentados pontos relacionados a concepção e desenvolvimento.

2.2.1 CONCEITO INICIAL

A medida que o desenvolvimento tecnológico impulsionou a evolução de computadores pessoais a indústria de jogos eletrônicos cresceu paralelamente a esse mercado. Esse crescimento pode ser explicado pela característica de entretenimento diferenciada que o jogo possui em comparação às demais mídias. Os jogos podem envolver jogadores em situações fantasiosas e extraordinárias,

como por exemplo, colocar o jogador no papel de um soldado na Segunda Guerra Mundial, sem no entanto, apresentar qualquer perigo físico.

A imersividade inerente ao jogo fez com que esta tecnologia passasse a ser utilizada como ferramenta para comunicar conhecimento passível de ser aplicado no mundo real. Os jogos, em geral, são capazes de modelar um ambiente que possui um desafio e um objetivo a ser alcançado, assim como na vida real. A grande vantagem do jogo é a possibilidade de permitir o erro, fazendo com que o jogador possa realizar todas as ações possíveis dentro desse cenário, sem sofrer qualquer tipo de consequência, além de aprender quais são as melhores decisões a serem tomadas e quais devem ser evitadas.

A partir disso, Bergeron (2006) explica que indústrias, educadores e mecanismos de *marketing* interessaram-se pelo potencial do jogo e iniciaram o desenvolvimento de aplicações sérias, surgindo então o *Serious Game*. Segundo Bergeron (2006), esta categoria de jogos são aplicativos computacionais que simulam um ambiente específico, caracterizado por:

- Possuir capacidade de comunicar conhecimento passível de ser aplicado no mundo real.
- Ser divertido.
- Possuir um mecanismo de recompensa.
- Possuir um desafio e um objetivo.

Embora o enunciado exposto anteriormente pareça genérico para tratar o termo *Serious Games*, existem outras definições e terminologias. Como afirma Crookall (2010), os debates relacionados aos *Serious Games* são recentes, especialmente a respeito da terminologia, pois algumas definições variam em relação a outras, fazendo com que as palavras *Serious* e *Game* usadas juntas sejam um oxímoro. Um enunciado que evidencia o fato descrito acima é o de Abt (1970) que descreve que a principal característica de um *Serious Games* é sua proposta pedagógica, destacando que o entretenimento não é inicialmente relevante.

2.2.2 BREVE HISTÓRICO

É importante entender os passos que levaram os jogos a serem utilizados como ferramentas educativas. Este tópico apresenta de forma resumida a história

dos *Serious Games* dividida em dois contextos; jogos aplicados ao âmbito militar e, jogos na área Médica.

2.2.2.1 JOGOS DE SIMULAÇÃO MILITAR

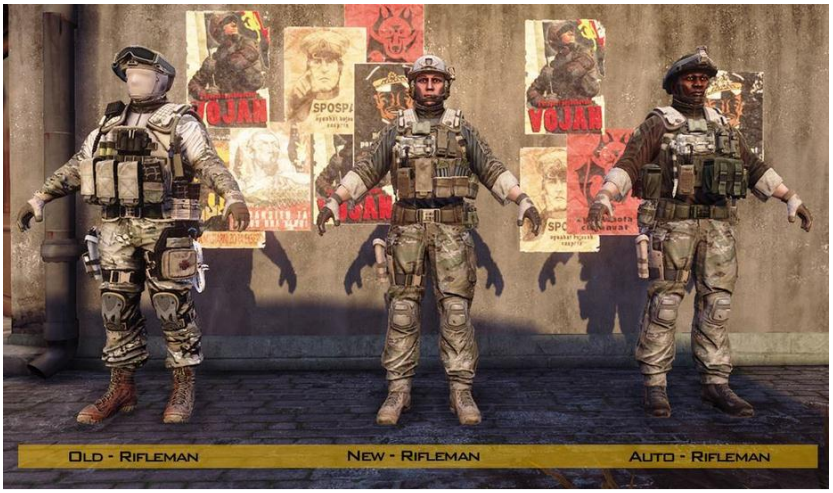
A comercialização de jogos como simuladores teve início no domínio militar, com o desenvolvimento de um simulador de voo mecânico criado por Edwin Albert Link no ano de 1929. Inicialmente esse simulador foi utilizado apenas como meio de diversão, pois era considerado caro pelos aviadores. No entanto, com a grande taxa de acidentes causados em treinamentos e com o começo da Segunda Guerra Mundial, o simulador de voo de Link foi aceito como tecnologia de treinamento e passou a ser amplamente utilizado (BERGERON, 2006). O simulador passou a ser comercializado com a identificação ANT-18 ou simplesmente *blue box*. A figura 4 ilustra operadores auxiliando um piloto em treinamento no *blue box*.

Figura 4- Treinamento de voo utilizando o simulador blue box

Fonte: Bergeron, 2006, p. 3

Devido ao desenvolvimento tecnológico, os jogos de simulação em computadores começaram a surgir em grande escala. O primeiro deles foi desenvolvido em 1952, pela *Rand Air Defense Lab* em Santa Monica. Em 1980, surge o *U.S.Army's Simulation Network* (SIMNET), jogo que simulava batalhas entre veículos terrestres e aéreos. Atualmente, jogos de simulação militares continuam a ser desenvolvidos, atualmente o instituto MOVES, escola de pós-graduação Naval, está trabalhando no desenvolvimento do jogo America's Army (BONK et al, 2005). A imagem a seguir ilustra a etapa de criação de personagens durante o jogo.

Figura 5 - Criação de personagem do simulador America's Army



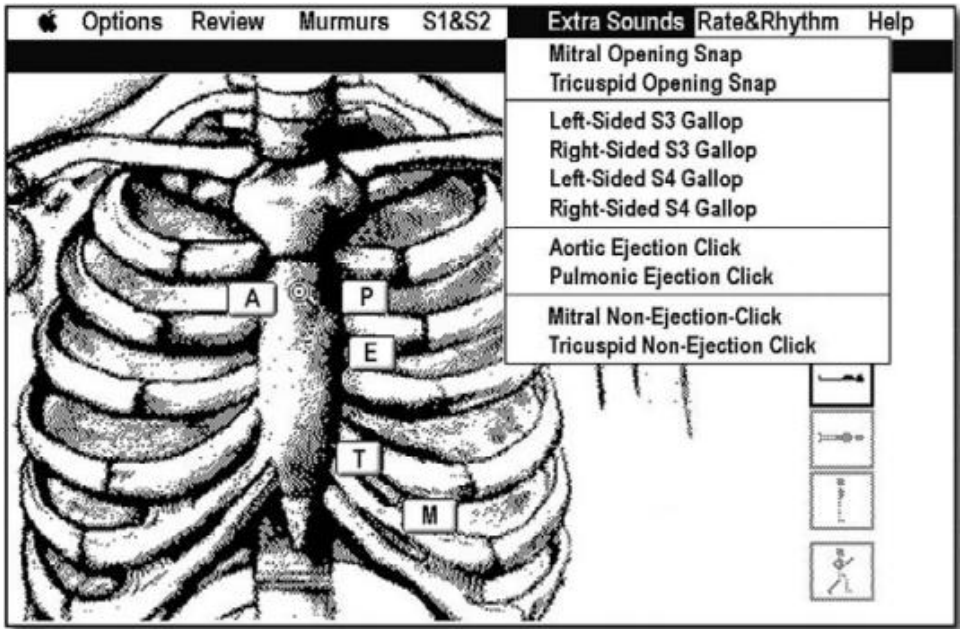
Fonte: Site oficial do jogo America's Army

2.2.2.2 *SERIOUS GAMES* NA MEDICINA

Não foi apenas a indústria militar que utilizou Serious Games como forma de treinamento. A área médica também contribuiu para o desenvolvimento de jogos como ferramenta de ensino, primeiramente utilizando sistemas tutores inteligentes que foram desenvolvidos por volta de 1967, em Oregon State University, Massachusetts *General Hospital Laboratory of Computer Science e University of Illinois* (BERGERON, 2006).

Em 1980 foi desenvolvido pelo *Harvard's Decision Systems Group* (DSG), o HeartLab, primeiro software educativo usado na medicina. O HeartLab tinha o objetivo de transmitir conceitos sobre auscultação cardíaca e foi considerado sofisticado na época de seu lançamento. A figura 6 ilustra o software em funcionamento, exibindo a cavidade torácica e os pontos que emitem sons durante o ciclo cardíaco (BERGERON et al, 1989):

Figura 6 – Software HeartLab



Fonte: Bergeron, 2006, p. 6

2.2.3 TECNOLOGIAS DE DESENVOLVIMENTO

As tecnologias envolvidas no desenvolvimento de *Serious Games* cada vez mais são otimizadas para permitirem ganho de tempo, diminuir a complexidade e aumentar rendimento no processo de desenvolvimento. A escolha das ferramentas é parte fundamental e muitas vezes, pode decidir o futuro do produto final (BERGERON, 2006). Embora todos os componentes do jogo sejam importantes para um produto final de qualidade, questões relacionadas às ferramentas de implementação merecem atenção especial.

2.2.3.1 FERRAMENTAS DE IMPLEMENTAÇÃO

No âmbito deste trabalho, implementação refere-se ao processo de construção do funcionamento dos mecanismos do jogo, desconsiderando o desenvolvimento da arte e dos efeitos sonoros. Nesse sentido, tecnologias como *game engines* ou linguagens de programação são ferramentas de implementação.

Escolher a ferramenta adequada depende de muitos fatores, tais como experiência do grupo de desenvolvimento, limitações financeiras ou prazo para conclusão do projeto. O tempo que será gasto na implementação é um fator

importante no planejamento do jogo, com isso, Bergeron (2006) destaca que antes de efetivamente escolher uma linguagem de programação ou *engine* para iniciar o desenvolvimento é importante uma ferramenta de prototipação.

O protótipo tem importância no sentido de demonstrar de forma simplificada o funcionamento do jogo, permitindo que este seja validado e corrigido e, se necessário, descartado. Nesse caso, *game engines* são importantes e facilitam a implementação do protótipo, pois possuem elementos pré-fabricados e *interfaces* amigáveis, possibilitando que todos os membros da equipe possam manipular e modificar o protótipo.

Na tarefa de desenvolvimentos de jogos, o protótipo é utilizado como meio de concretização da ideia, normalmente direcionado a construção da mecânica, sem se preocupar com otimizações ou padrões de desenvolvimento, resultando em um produto não comercializável. A partir disso, torna-se necessário o uso de ferramentas de implementação que concedam potencial para o desenvolvimento, tornando necessária a tarefa de codificação (BERGERON, 2006).

2.2.3.2 CODIFICAÇÃO

Existem muitas linguagens de programação disponíveis e viáveis para serem utilizadas no desenvolvimento de jogos eletrônicos. A escolha da linguagem de programação a ser utilizada para codificação envolve muitos aspectos que afetam diretamente o futuro do processo de implementação. Segundo Bergeron (2006), a linguagem C++ tem sido mais utilizada para o desenvolvimento de *Serious Games* devido sua popularidade entre desenvolvedores, implicando em uma algumas vantagens no processo de implementação do jogo:

- Existem muitos padrões de desenvolvimento que podem facilitar a comunicação entre as pessoas que compõe o grupo de programadores.
- Possui a *Standard Template Library*, que provê algoritmos e estruturas de dados que podem ser usadas em várias plataformas, agilizando e facilitando alguns pontos da implementação.

2.2.4 ESPECIFICIDADES DO DESENVOLVIMENTO DE UM *SERIOUS GAME*

A elaboração de um documento conceitual ou *Game Design Document* (GDD), tarefa que precede o desenvolvimento do jogo, tem por objetivo construir uma representação geral do jogo e dos recursos que o desenvolvimento irá demandar. Em geral, o desenvolvimento conceitual de um *Serious Game* envolve todos os processos de um jogo convencional focado apenas no entretenimento. No entanto, o planejamento de um *Serious Game* exige algumas tarefas específicas devido a seu carater educativo (BERGERON, 2006).

Para que o jogo seja capaz de comunicar conhecimento, são necessários esforços no sentido de entender o ambiente que será abordado no jogo. Estudar o ambiente permite a coleta de informações importantes, afetando diretamente o modo como o funcionamento do jogo será utilizado para representar o ambiente real. Estudar o ambiente ajuda a evitar elementos que possam interferir no objetivo da comunicação do conhecimento (ZEE; HOLKENBORG; ROBINSON, 2011).

Em geral, é importante que um especialista de domínio acompanhe o desenvolvimento do jogo, avaliando a representação da entrada e saída das informações e verificando se os componentes estão sendo eficientes como veículos de informação.

2.3 MACSIM

Este tópico introduz os principais conceitos sobre o *framework* de simulação MaCSim, desenvolvido como ferramenta para projetos de pesquisa do Laboratório de Matemática Computacional da Empresa Brasileira de Pesquisa Agropecuária (Embrapa).

2.3.1 *FRAMEWORK* DE SIMULAÇÃO MACSIM

No âmbito da atividade de modelagem e simulação computacional cada vez mais surge a necessidade de ferramentas que facilitem a implementação e manipulação de modelos. Moore et al. (2007), destaca que a facilidade de manipulação, reúso eficiente e padronização são os principais requisitos que as

ferramentas de modelagem deveriam atender. São muitas as vantagens alcançadas a partir dessa abordagem, tais como facilidade de comunicação entre especialistas de domínio, esforço mínimo para acoplar os modelos em hierarquia e a possibilidade de integração com diferentes aplicações cliente (MANCINI et al, 2013).

Nesse contexto, *frameworks* de simulação (FS) permitem o desenvolvimento de modelos eficazes, em relação às necessidades propostas por Moore. Um FS possui uma linguagem de programação genérica e, a grande vantagem de seu uso, demonstra-se na capacidade de utilizar esta linguagem para a implementação dos modelos. A partir disso, o limite para a implementação do modelo é diretamente proporcional aos recursos da linguagem de programação (MANCINI et al., 2013).

O MaCSim foi desenvolvido para atender equipes de pesquisadores da Embrapa, engajadas no processo de especificação de modelos. Em virtude da grande variedade de especialistas de domínio trabalhando em conjunto, o MaCSim possibilita padronização na prototipação dos modelos, facilitando a comunicação pessoal e posterior integração entre os modelos (MANCINI et al., 2013). A primeira aplicação do MaCSim, está sendo realizada no contexto do projeto PECUS[2], a partir do qual está sendo desenvolvido um modelo para mitigação de gases causadores do efeito estufa.

2.3.2 ARQUITETURA DO FRAMEWORK

O MaCSim foi implementado a partir do paradigma de orientação a objetos, em linguagem C++, e possui a características de suportar a construção hierarquica de modelos, modularidade e possibilidade de reúso dos modelos. Uma das possibilidades do MaCSim é permitir o desenvolvimento da aplicação cliente separada do simulador. Assim que um simulador é construído a partir do MaCSim, este pode ser compilado em uma biblioteca de vínculo dinâmico (DLL) e invocado em qualquer tipo de aplicação que suporte chamada de DLL, como por exemplo, uma aplicação desenvolvida em linguagem C++ (MANCINI et al, 2013).

A estrutura do MaCSim é composta por dois pacotes que dividem a estrutura de controle da estrutura de modelagem, sendo estes, *Simulation Engine* e

[2] Informações sobre o projeto PECUS podem ser acessadas no endereço eletrônico: https://www.embrapa.br/busca-de-projetos/-/projeto/38213/projeto-da-rede-pecus

Simulation Object. O pacote *Simulation Engine* compõe classes que implementam as funcionalidades de controle da simulação, algumas das classes mais importantes são *Control, Event e Event Manager*. A classe *Control* possui métodos que possibilitam a manipulação do estado da simulação. As classes *Event* e *Event Manager*, funcionando em conjunto, provêem a capacidade de agendamento de eventos, em qualquer tempo da simulação. O pacote *Simulation Object* possui as classes que fornecem recursos base para implementação dos modelos, sendo *Model* a classe mais importante desse pacote. A classe *Model* caracteriza um modelo genérico, que pode ser especializado de acordo com a necessidade do modelista (MANCINI et al, 2013).

A capacidade de comunicação do MaCSim com diferentes aplicativos clientes é possibilitada pela classe *Interface*, que se posiciona fora dos pacotes descritos anteriormente. Por apresentar maior importância para o objetivo deste trabalho, a classe *Interface* será apresentada no tópico seguinte.

2.3.3 CLASSE *INTERFACE*

A classe *Interface* funciona como porta de acesso aos recursos do MaCSim. Em sua composição estão instâncias das classes *Event_Manager*, *Control* e *Model*. Essas instâncias provêem acesso aos métodos públicos, suficientes para coordenar o processo de simulação. Para possibilitar a comunicação com uma aplicação, a *Interface* implementa um *wrapper*[3] com alguns dos métodos das instâncias de *Event_Manager, Control e Model.*

Dentre os métodos que podem ser acessados, os mais importantes para tornar possível o funcionamento de uma simulação são descritos a seguir:

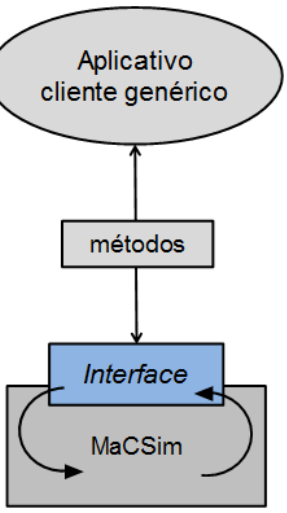
- Método *BuildModel*, que invoca a criação de uma instância do modelo já previamente estruturado.
- Método *SetSimulationTimes*, permite a inicialização dos valores do tempo inicial e final, bem como o tamanho do passo de tempo da simulação.
- Método *AddControlEvent,* possibilita o agendamento de um evento de controle de simulação em um tempo especificado.

[3] Em Ciência da Computação o termo wrapper refere-se ao empacotamento de um método por outro.

- Método *SetInputValue,* permite inicializar os valores das variáveis da simulação, por meio de índice ou nome.
- Método *GetOutPutValue,* permite o acesso aos valores de saída (valores calculados) da simulação.
- Método *Run,* executa a simulação se todas as inicializações estiverem corretas.
- Método *setAfterIterateCallBack* e *setBeforeIterateCallBack*, dois métodos importantes na comunicação entre o MaCSim e o aplicativo cliente durante a execução da simulação. Permitem verificar o estado da simulação por meio de *CallBacks,* antes e depois de uma iteração.

A Figura 7 ilustra o funcionamento da classe *Interface* em comunicação com um aplicativo cliente. Por meio de um ponteiro da classe *Interface*, o aplicativo cliente pode invocar os métodos do MaCSim a qualquer momento, de acordo com a necessidade. A *Interface* possui os ponteiros necessários para retornar os métodos solicitados, sem a exposição destes para a aplicação.

Figura 7- Diagrama que representa a comunicação entre o aplicativo cliente e o MaCSim



Fonte: Elaborada pelo autor

2.3.4 CONSTRUÇÃO DE UM SIMULADOR NO MACSIM

A implementação do modelo a partir da classe *Model* é a primeira tarefa do processo de construção do simulador. A partir de um modelo já especificado, o especialista de domínio fará uma especialização da classe *Model* para transpor os modelos matemáticos em formato de código C++. Variáveis e constantes do modelo são implementadas como objetos da classe *Variable*, essa classe possui uma variável de dupla precisão e métodos de atribuição e retorno. As equações do modelo serão implementadas como funções da linguagem de programação e acessadas durante a simulação por meio de um ponteiro de função (MANCINI et al, 2013).

A imagem a seguir exibe um fragmento de código que corresponde a um modelo matemático escrito no MaCSim, o método exibido acessa valores de variáveis por meio do método *Value* e efetua o cálculo:

Figura 8 - Equação do modelo de crescimento de bovino de corte escrita no MaCSim

```
double Cls_Oltjen_Model::d_dt_GORD(){

    double a1 = In_DMI->Value();
    double a2 = Par_alfa->Value();
    double a3 = Aux_EBW->Value();
    double a4 = In_NEm->Value();
    double a5 = In_Neg->Value();
    double a6 = St_PROT->SearchOutputPort("dS_dt")->Value();
    return ((a1-a2*pow (a3, 0.73 )/a4)* a5-a6*c_E_PROT)/c_E_FAT;

}
```

Fonte: Elaborada pelo autor

Após a prototipação dos modelos, o código é incluído nos pacotes do *framework*. Uma instância da classe especializada, que corresponde ao modelo desenvolvido pelo usuário, será criada dentro da classe *Control*. Com a chamada do método *BuildModel* o modelo será construído e o simulador estará pronto para ser configurado e utilizado.

2.4 PECUS

O projeto PECUS tem o objetivo de estudar a dinâmica de sistemas de produção de pecuária no âmbito do território brasileiro, em diferentes biomas. As pesquisas tem o intuito de mitigar a emissão de gases causadores do efeito estufa proporcionado pelo cenário da pecuária.

Para auxiliar a pesquisa e aumentar o entendimento do comportamento do sistema pecuário, foram desenvolvidos modelos matemáticos. Cada modelo abstrai o comportamento de um sistema intrínseco ao sistema produtivo de pecuária. Os modelos representam:

- Crescimento de bovinos
- Crescimento de pastagem
- Dinâmica da água no solo
- Emissões de gases
- Pastejo

O inter-relacinamento dos modelos possibilita a representação simplificada do comportamento de um sistema de produção de pecuária.

3 PROPOSTA DE TRABALHO

A partir do relacionamento entre as áreas de Simulação Computacional e Jogos Digitais, este trabalho propôs elucidar o desenvolvimento de um jogo digital baseado em simulação computacional. Para tanto, é proposto o desenvolvimento de um *Serious Game* que virtualiza o ambiente de produção de pecuária de corte utilizando como base o *framework* de simulação MaCSim e o modelo PECUS.

Primeiro, será explicado a proposta de funcionamento do jogo e em seguida, o acoplamento deste com o simulador.

3.1 FUNCIONAMENTO DO JOGO

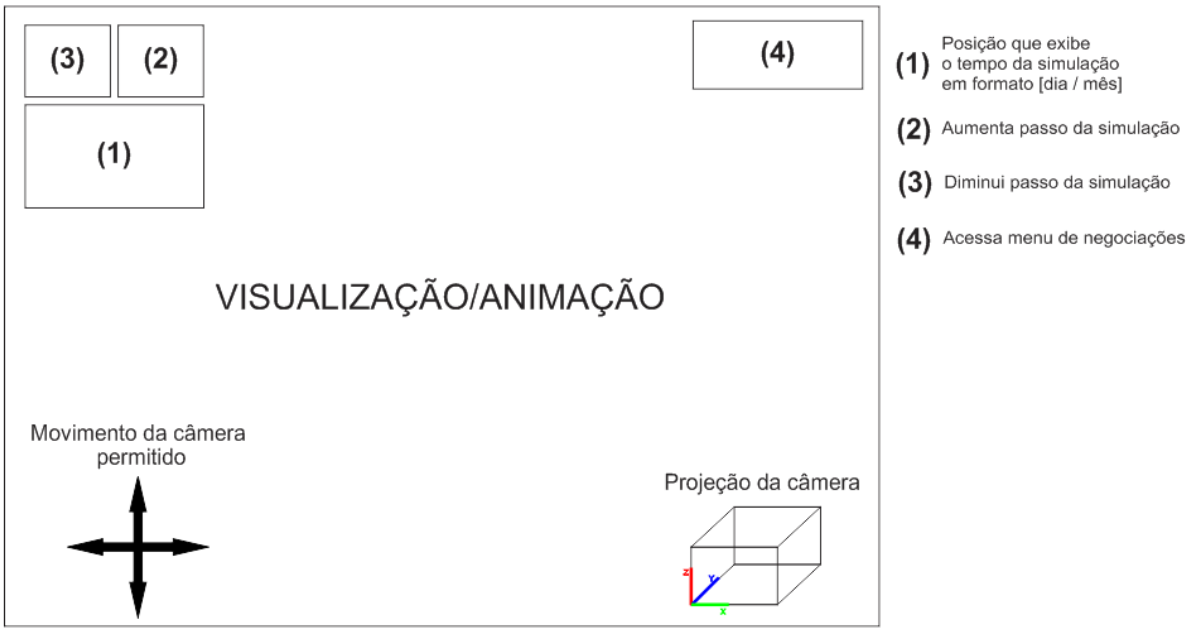
O jogo virtualiza o ambiente de pecuária de corte, possibilitanto ao usuário realizar algumas das tarefas no âmbito da atividade de manejo de rebanhos. Nesse sentido, são possibilitadas ações de compra, venda e suplementação de animais, rotacionamento entre pastagens, adubação e compra/venda de piquetes (área de pastagem individual).

São múltiplos os objetivos que podem ser atingidos durante uma partida, tais como, peso ótimo de abate para o rebanho, máxima lucratividade ou minimização da emissão de gases causadores do efeitos estufa. O estado do ambiente virtual é alterado de acordo com as decisões tomadas pelo usuário ao longo do tempo. As ações permitidas ao jogador podem ser acessadas durante qualquer momento do jogo, através de um menu de negociações.

O jogo não possui um *feedback* direto dizendo ao usuário qual seu desempenho (se ganhou ou perdeu). Todas as ações tomadas pelo jogador durante a partida são registradas em uma lista dentro do jogo, que posteriormente, serão analizadas por um especialista de domínio. Relacionando as decisões do usuário com os estados apresentado pelo jogo, o especialista pode inferir o desempenho da partida, de acordo com os objetivos que eram pretendidos.

A visualização do jogo é isométrica[4] e possibilita que o usuário navegue pelo ambiente virtual em todas as direções. A figura 9 representa uma tela do jogo durante o *gameplay*, destacando a organização da *interface*, a projeção da câmera e a área de visualização:

Figura 9 - Tela de *gameplay*



Fonte: Elaborada pelo autor

3.2 INTEGRAÇÃO ENTRE O JOGO E O SIMULADOR

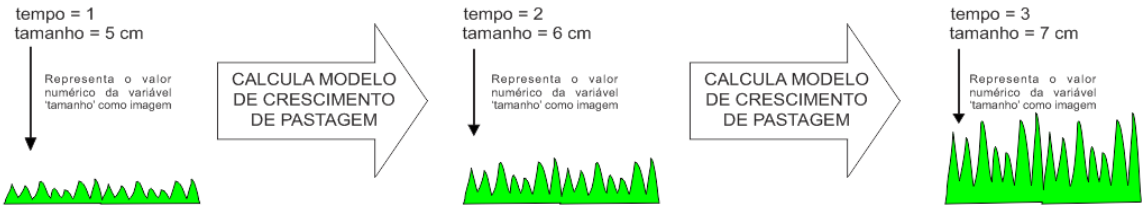
A fim de que o jogo seja capaz de comunicar conhecimento, é estritamente necessário um meio de correspodência com a realidade. Para atender a esse requisito é proposto integração entre o *framework* MaCSim e modelo PECUS com o jogo. A integração tem por objetivo vincular os dados numéricos gerados pela simulação aos elementos gráficos do jogo. A medida que a simulação evolui, alterando o estado das variáveis dos modelos, os elementos gráficos se alteram, buscando o objetivo de transparecer o significado dos valores numéricos.

Como exemplo, a figura 10 a seguir ilustra esse processo, a partir do crescimento de pastagem, em que existem duas variáveis, tempo e tamanho. A

[4] A visualização isométrica mantém a câmera posicionada acima dos objetos, em uma inclinação aproximadamente de 30° em relação a horizontal e 45° em relação a vertical.

variável tempo armazena o tempo transcorrido pela simulação e a variável tamanho, o valor do cálculo do modelo de crescimento de pastagem:

Figura 10 - Processo de representação de imagens por meio das variáveis



Fonte: Elaborada pelo autor

A medida que o cálculo do modelo é realizado, a variável de estado que armazena o valor do tamanho da pastagem aumenta, fazendo com que a imagem que representa graficamente a pastagem também aumente, possibilitando a visualização direta do fenômeno de crescimento da pastagem, sem exigir a necessidade de interpretação dos dados numéricos.

4 DESENVOLVIMENTO

O capítulo que segue, relata o desenvolvimento do jogo proposto. O primeiro tópico esclarece as tecnologias utilizadas. Em seguida, para facilitar o entendimento do funcionamento do código de forma geral, cada pacote que comporta as classes do jogo será explicado individualmente, para posteriormente todos os pacotes serem inter-relacionados de forma mais clara.

4.1 TECNOLOGIAS DE DESENVOLVIMENTO

Os critérios para escolha das ferramentas de desenvolvimento basearam-se em requisitos que facilitassem o desenvolvimento de jogos digitais, além de levar em consideração a experiência e comodidade com o uso de determinadas tecnologias. O ambiente de desenvolvimento escolhido foi o CodeBlocks, versão 12.11, e a linguagem base utilizada para codificação foi C++. Como plataforma de desenvolvimento, optou-se pelo sistema operacional *Windows*.

A exigência por uma biblioteca gráfica que proporcionasse liberdade na implementação da estrutura gráfica do jogo culminou na escolha da API (*Application Programming Interface*) OpenGL (*Open Graphics Library*). Para desenvolvimento dos demais elementos da aplicação, tais como entrada e saída de dados, *callbacks*[5] e *interface* da janela do jogo, foi utilizada a *Windows* API (WinAPI).

4.2 PACOTES DE CLASSES

A divisão da estrutura do código por meio de pacotes, permite que as classes implementadas sejam organizadas por características semelhantes, facilitando a manutenção e organização durante o desenvolvimento. Os pacotes que compõem as classes para funcionamento do jogo são; *Engine, Game* e *Simulator.* Cada pacote é explicado nos tópicos seguintes.

[5] Uma callback é uma instrução (código executável) passada como argumento para um método. A chamada do método pode implicar na execução da instrução.

4.2.1 *ENGINE*

O pacote *Engine* é composto pelas classes que implementam o funcionamento dos elementos mais genéricos do jogo, tais como gerenciamento de memória, de textura e estados. Este pacote pode ser entendido como um motor, fazendo os mecanismos do jogo funcionarem. Inserida nesse pacote, a classe `Class_Engine`, implementa o núcleo de gerenciamento de todos os recursos do jogo, possuindo métodos de inicialização, atualização, renderização e alocamento/desalocamento de memória.

Uma instância de `Class_Engine` realiza a chamada dos métodos `Init`, `Update` e `Render`, executando a atualização correspondente de todos os objetos do jogo. O cabeçalho da classe `Class_Engine` é demonstrado na figura 11, exibindo os métodos citados anteriormente:

Figura 11 – Cabeçalho da classe *Engine*

```
class Class_Engine {
    public:
        Class_Engine(Class_Time* pTime_arg) {
            pObj_Time          = pTime_arg;
            pObj_GameStates    = new Class_GameStates();
            bInit              = false;
        }
        virtual ~Class_Engine();
    bool bInit;
    void Init();
    void Update();
    void Render();
    void Destroy();

    private:
        Class_Time* pObj_Time;
        Class_GameStates*   pObj_GameStates;
};
```

Fonte: Elaborada pelo autor

O objeto `pObj_Time`, instância da classe `Class_Time`, é responsável pelo controle do tempo de cada *frame*, verificando e corrigindo possíveis alterações na

taxa de variação do tempo de um *frame*. A instância `pObj_GameStates` é uma das mais importantes dessa classe, por ser responsável pela manipulação dos estados do jogo. Entende-se por estados todas as cenas/telas que podem ser visualizadas durante o acesso a menus ou *gameplay.*

Para criação da estrutura de estados, foi desenvolvida uma estrutura polimórfica[6], sendo `Class_State` a classe pai. Essa classe possui os métodos virtuais que inicializam, atualizam e renderizam um estado. A figura a seguir demonstra o cabeçalho dessa classe, que possui a prototipação dos métodos `Load`, responsável por alocar memória para os objetos do estado, `Update`, que atualiza o estado e `Render` que desenha os elementos na tela:

Figura 12 - Cabeçalho da classe State

```
class Class_State {

    public:
        Class_State(int);
        virtual ~Class_State();

        int  id;
        bool loaded;

        virtual void Load()   = 0;
        virtual void Update() = 0;
        virtual void Render() = 0;

        void ItsRunning(bool);

    private:
        bool bItsRunning;

};
```

Fonte: Elaborada pelo autor

Todos os estados do jogo (menu, *gameplay*, menu de compra/venda etc) são especializações dessa classe e implementam os métodos virtuais, possibilitando que qualquer tipo de cena seja desenvolvida com relativa facilidade de acoplamento.

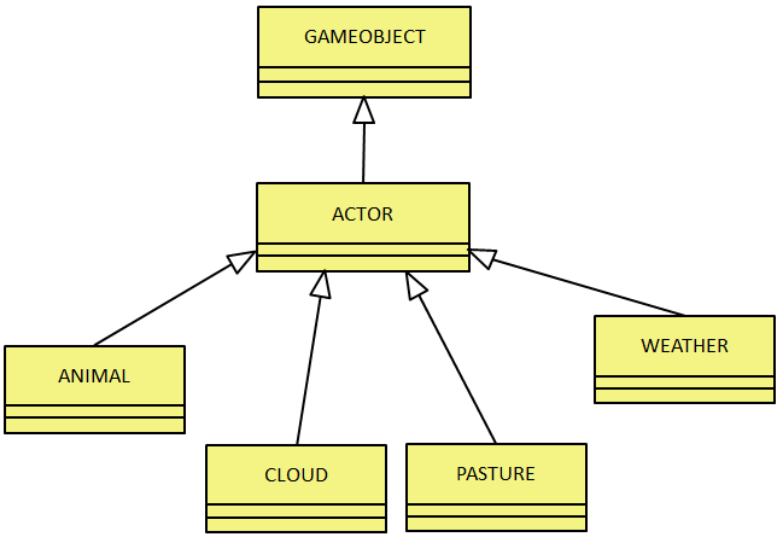
[6] Polimorfismo refere-se à capacidade de uma classe representar o comportamento de classes que a referenciam.

4.2.2 *GAME*

Todas as classes relacionadas aos mecanismos do jogo (atores, cenas, *interface etc)* estão no pacote *Game*. Dentro desse pacote, existem dois tipos mais importantes de classes base, a classe abstrata `Class_GameObject` e a classe `Class_Scene`.

A classe `Class_GameObject` é a classe base para todos os elementos que podem ser visualizados durante o *gameplay*, possuindo uma variável que armazena o tipo do objeto, que o caracteriza dentro da hierarquia do código e os métodos virtuais de inicialização, atualização e renderização. Essa classe, por sua vez, é especializada na classe `Class_Actor` e esta, por fim, em qualquer outro elemento do jogo. O diagrama a seguir ilustra a hierarquia das classes a partir da `Class_GameObject`:

Figura 13 - Diagrama da hierarquia das classes



Fonte: Elaborada pelo autor

As classes `Class_Animal`, `Class_Weather`, `Class_Pasture` e `Class_Cloud` são alguns dos objetos que modelam o funcionamento do jogo.

A classe `Class_Scene` é a classe pai de todas as cenas/telas do jogo, para tanto, é uma especialização da classe `Class_State`. A especialização mais importante da `Class_Scene` é a classe `Class_Level` que comporta todas as

instâncias que serão atualizadas durante o g*ameplay*, dentre elas a câmera, listas de atores e objetos da *interface*.

4.2.3 *SIMULATOR*

O pacote *Simulator* possui apenas uma classe, sendo esta a mais importante para o propósito do jogo. A classe `Class_Simulator` implementa a *interface* que permite a comunicação entre os mecanismos implementados no pacote *Game* e o simulador MaCSim. Essa classe possui um ponteiro para o MaCSim e é responsável pela configuração da simulação que será executada durante o jogo.

O método `Load` faz a chamada dos métodos do MaCSim, configurando a simulação. Esse método é exibido na figura 14:

Figura 14 – Método *Load* da classe Simulator

```
void Class_Simulator::Load() {

    globalStock = 4;

    double Inputs[] = {1,-23,-80,60,0.8,30,25,15, globalStock, 0.4,1,
                       300,1000,1000,0,3.0,15,2.0,2.2,2.2,1.8,1.5,65,
                       65,85,1.35,1.1,0.79,0.7,0.6,0.25, 86.5, 23.2,
                       14.4, 0.09, 0.06, 0.002};

    pObj_Simulator->BuildModel();

    step = (15.0);

    pObj_Simulator->SetSimulationTimes(0, (0.0166666666667) * step);

    for(int i = 0; i < 36; ++i) {
        pObj_Simulator->SetInputValue(i, Inputs[i]);

    }
    pObj_Simulator->simulatorControl_prtHid->Initialize();

}
```

Fonte: Elaborada pelo autor

A variável `globalStock` armazena o número de animais presentes na pastagem e o array `Inputs` os valores de inicialização da simulação. Os métodos seguintes correspondem respectivamente a construção do modelo (`BuildModel`),

inicialização do tempo da simulação (`SetSimulationTimes`), inicialização dos valores de entrada (`SetInputValue`) e inicialização do ponteiro de controle (`Initialize`).

Os valores das variáveis de estado da simulação (cálculo da diferencial) e o agendamento de eventos devem ser acessados externamente a classe `Class_Simulator`, para isso, foram criados métodos individuas para acesso a cada variável e evento. A imagem a seguir demonstra um método que acessa o valor da variável de estado que corresponde ao peso do animal e outro que adiciona animais ao rebanho, os demais métodos seguem o mesmo padrão de implementação:

Figura 15 – Métodos de acesso às variáveis e ao controle da simulação

```
double Class_Simulator::EBW() {
    return (pObj_Simulator->GetOutputValue("EBW"));
}

void Class_Simulator::AddAnimal() {
    pObj_Simulator->AddInputEvent(currentTime+step,  8,
                                  pObj_Simulator[0]->GetInputValue(8)+1);
}
```

Fonte: Elaborada pelo autor

O método `EBW` faz a chamada do método `GetOutPutValue` para acessar a variável de estado do peso do animal, o parâmetro é uma *string* que deve ser o nome correspondente a variável na implementação do modelo. O método `AddAnimal` invoca o método `AddInputEvent` para agendar um evento para o simulador, seus parâmetros são o tempo de ocorrência do evento, tipo de evento e valor da nova variável.

Por fim, o método que atualiza o estado da simulação, realizando os cálculos, alterando o valor do passo de tempo e executando os eventos é o `Update`. A figura a seguir apresenta o método:

Figura 16 – Método Update da classe Simulator atualizando o estado da simulação

```
void Class_Simulator::Update() {

    pObj_Simulator->ExecuteEventsUntilTime(pObj_Simulator->currentTime_hid);


    pObj_Simulator->RungeKutta_Method(&pObj_SimulatorcurrentTime_hid,
                                      pObj_Simulator->timeStepOfSimulation_hid);

    pObj_Simulator->currentTime_hid += pObj_Simulator->timeStepOfSimulation_hid;

    currentTime      = pObj_Simulator->currentTime_hid;

    pObj_Simulator->currentIterationOfSimulation_hid++;

}
```

Fonte: Elaborada pelo autor

No fragmento de código apresentado na figura 16, são chamados os métodos `ExecuteEventsUntilTime` e `RungeKutta_Method` que respectivamente executam os eventos e calculam as diferenciais utilizando o algoritmo de integração numérico Runge-Kutta. As demais linhas de código atualizam o tempo da simulação.

4.3 INTERAÇÃO ENTRE O SIMULADOR E OS OBJETOS DO JOGO

Cada pacote, apresentado anteriormente, possui classes que executam conjuntos de tarefas específicas, visando um mesmo objetivo. O funcionamento destes pacotes foram apresentados individualmente sem inter-relação entre eles. Para que os objetos do jogo se relacionem com a simulação, as classes dos pacotes *Game* e *Simulator* se relacionam, trocando informações.

A classe `Class_Level` é o local em que ocorrem as interações entre as demais classes. Ela possui o ponteiro da classe `Class_Simulator` e dos objetos que compõem o jogo, tais como animais, pastagem, imagens de *background* etc. Em seu método `Update`, os valores de saída da simulação são passados como argumento aos objetos. Para exemplificar, é mostrado a seguir o fragmento de código que realiza o crescimento da pastagem em função do valor da variável da simulação:

Figura 17 – Laço que percorre os objetos de pastagem

```
vector<Class_Pasture*>::iterator it;
for(it = pObj_PastureList.begin(); it != pObj_PastureList.end(); ++it) {
        (*it)->PastureGrowth(LeafMass_arg);
        (*it)->Update();
}
```

Fonte: Elaborada pelo autor

O laço percorre a lista de ponteiros dos objetos da pastagem fazendo a chamada do método `PastureGrowth`. Esse método recebe como argumento o valor da variável calculada a partir do método `LeafMass` e o armazena em uma variável como é mostrado na figura a seguir:

Figura 18 – Método *PastureGrowth*

```
void Class_Pasture::PastureGrowth(double LeafMass_arg) {
    dHafter = dH;
    dH      = dHi + (LeafMass_arg*dGrowthProportion);
}
```

Fonte: Elaborada pelo autor

A variável `dH` é multiplicado por um valor constante, `dGrowthProportion`, a fim de que se reduza o tamanho do valor de `LeafMass_arg`, que em alguns casos pode ser muito elevado. O valor `dH` é então atribuido a posição e dimensão da imagem, para que a animação acopanhe a alteração da variável durante o jogo. O fragmento de código a seguir demonstra a atualização da pastagem:

Figura 19 – Método Update atualizando a posição e dimensão da imagem da pastagem

```
void Class_Pasture::Update() {

    Position(x, y + ((dHafter-dH)/2), z);
    Dimension(400, ((float)dH));

}
```

Fonte: Elaborada pelo autor

A atualização dos demais objetos do jogo seguem o mesmo padrão de código do crescimento da pastagem. A figura 20 demonstra o código de atualização do crescimento do animal, evidenciando o mesmo padrão:

Figura 20 – Fragmentos de código da atualização do animal

```
for(vector<Class_Animal*>::iterator it = pObj_Animal.begin(); it != pObj_Animal.end(); ++it) {
        (*it)->Growth(pObj_Simulator->EBW());
        (*it)->UpdateCameraShift(pObj_Camera->Speedx(), pObj_Camera->Speedy());
        (*it)->Update();
}
void Class_Animal::Update() {

    z = -(int(930-y)/40)-2;

    Dimension((float)dWeight+10, (float)dWeight);

}
void Class_Animal::Growth(double Weight_arg) {

    dRealWeight = Weight_arg;
    dWeight = dRealWeight * dWeightProportion;

}
```

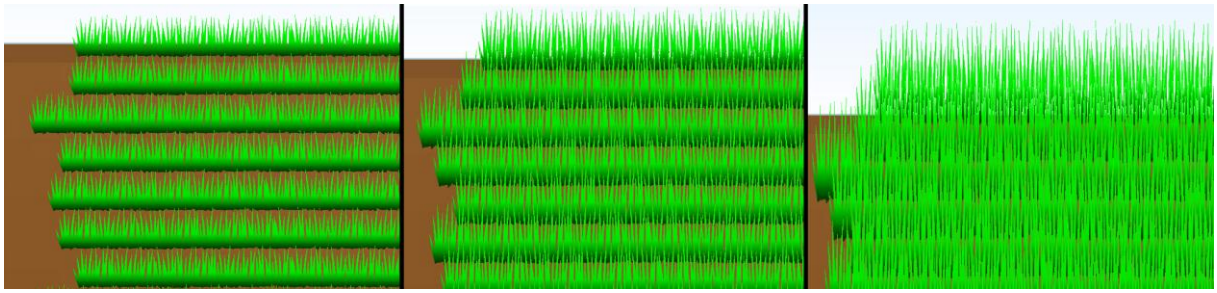
Fonte: Elaborada pelo autor

A lista de objetos de animais é percorrida, chamando o método que recebe como argumento o valor do peso do animal calculado pela simulação (`Growth`), que posteriormente é relacionada a dimensão da imagem do animal no jogo (`Update`).

4.4 ANÁLISE DOS RESULTADOS

A integração entre o jogo e o simulador foi realizada sem apresentar problemas técnicos. O relacionamento entre as variáveis de saída da simulação e os objetos do jogo também atigiram as espectativas. Cada variável de estado (peso do animal, tamanho da pastagem etc) vinculada à imagem do objeto pôde ser representada graficamente. A figura a seguir ilustra o desenvolvimento da pastagem durante a simulação:

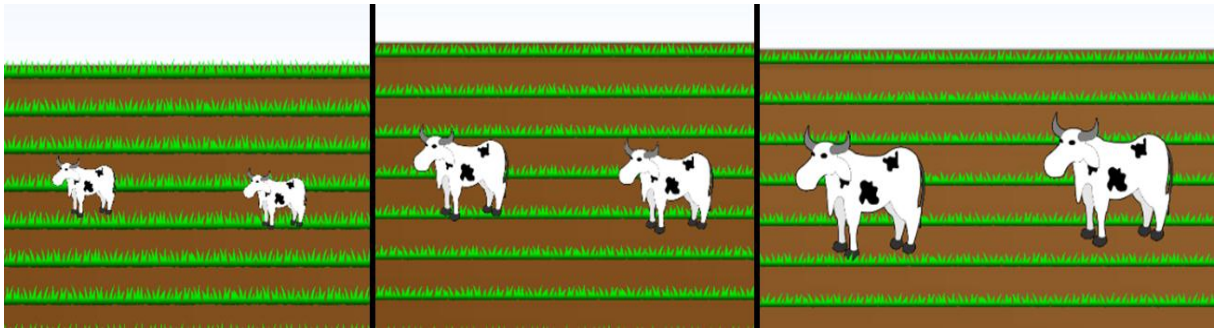
Figura 21 – Pastagem em três estados de crescimento



Fonte: Elaborada pelo autor

A medida que o tempo evolui no jogo, o cálculo do modelo possibilita que a pastagem se desenvolva como evidenciado na figura 21. O desenvolvimento corporal do rebanho também representou com sucesso as variáveis. A seguir, uma ilustração demonstra o crescimento do bovino durante o pastejo:

Figura 22 – Desenvolvimento corporal do animal em três estados

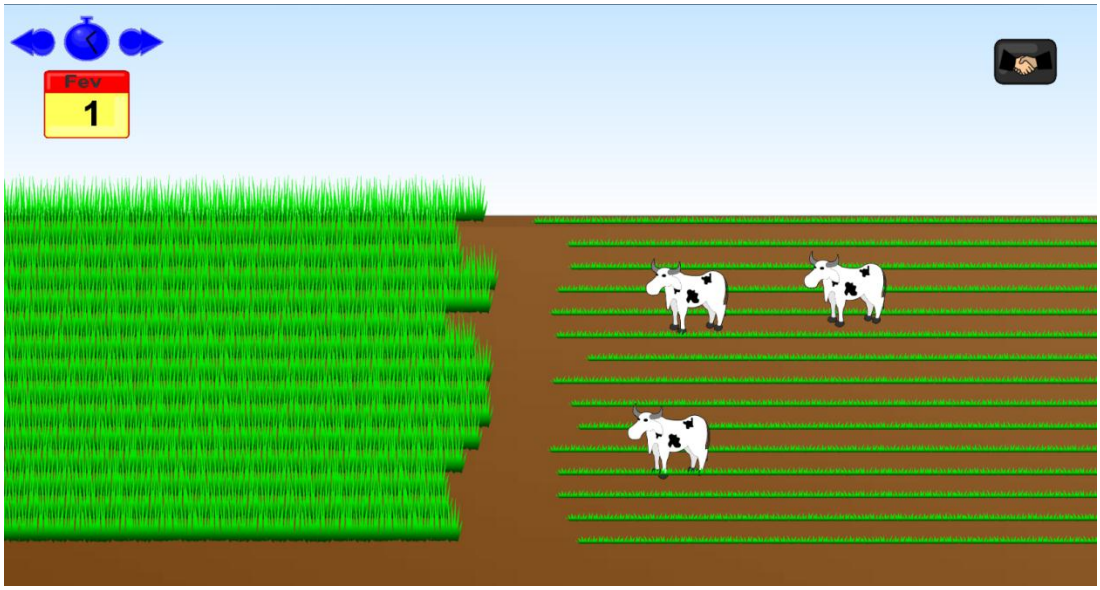


Fonte: Elaborada pelo autor

É possível visualizar o desenvolvimento corporal do animal depois de um certo período de pastejo.

Um fator interessante, e esperado, é a representação de fenômenos não implementados diretamente, mas que ocorrem devido ao cálculo dos modelos e o relacionamento entre os objetos do jogo. Um exemplo desse tipo de fenômeno é o crescimento da pastagem, que ocorre proporcionalmente ao número de animais ou, o crescimento do animal, que é restringido, caso esteja em um pasto com pouca densidade de folhas. A figura 23 exibe uma tela de *gameplay*, em que existem dois piquetes, apenas um dos piquetes possui animais:

Figura 23 – Tela de *gameplay*



Fonte: Elaborada pelo autor

O piquete sem animais apresenta grande densidade de folhas, no passo que o piquete com o rebanho apresenta poucas folhas. Nesse sentido, o jogo é capaz de transparecer um sistema de produção, permitindo ao usuário visualizar cada estado do ambiente e, posteriormente, inferir a melhor decisão a ser tomada.

5 CONSIDERAÇÕES FINAIS

A partir da análise dos resultados são inferidas algumas conclusões. Como assunção de maior importância para o trabalho, constatou-se possível a integração entre um jogo digital e um simulador computacional. Como assunções secundárias, pode-se destacar a possibilidade de virtualizar um ambiente de sistema de produção por meio de um jogo digital, utilizando-se equações diferenciais como base. Como consequência da última afirmação, conclui-se que o jogo desenvolvido possui caráter educativo, sendo capaz de transmitir conceitos relacionados a pecuária de corte, além de possibilitar a transmição de conceitos sobre modelagem matemática.

Pela necessidade de se possuir um especialista de domínio para avaliar os resultados durante o jogo, uma possibilidade de trabalho futuro se encontra na aplicação de inteligência artificial. Um algoritmo de inteligência artificial poderia inferir se o jogador realizou boas ações, evitando a figura do especialista.

REFERÊNCIAS BIBLIOGRÁFICAS

ABT, Clark C. **Serious Games.** New York: University Press Of America, 1987.

BERGERON, Bryan. **Developing Serious Games.** Hingham: Charles River Media, 2006. 452 p.

BERGERON, Bryan; GREENES, Robert; **CLINICAL SKILL-BUILDING SIMULATIONS IN CARDIOLOGY: HEARTLAB AND EKGLAB.** Boston: Elsevier Ireland Ltd, v. 30, 1989.

BONK, J.C.; DENNEN, V.P. **Massive Multiplayer Online Gaming: A Research Framework for Military Training and Education.** Washington, DC. March, 2005.

CROOKALL, David. **Serious Games, Debriefing, and Simulation/Gaming as a Discipline.** 2011. Disponível em: <http://sag.sagepub.com/>. Acesso em: 10 out. 2014.

DAS, S; GUPTA, P.K. **A mathematical model on fractional Lotka–Volterra equations. Journal of Theoretical Biology**, Varanasi, n. 277, v.1, p. 1 - 6, may. 2011.

FORRESTER, J. W. **Principles of system.** 2. ed. Cambridge: Wright-allen Press, 1968. 197 p.

FREITAS FILHO, P. J. Introdução à Modelagem e Simulação de Sistemas. 1. ed. Florianópolis: Visual Books, 2001. 312 p.

GARCIA, Claudio. **Modelagem e simulação.** São Paulo: Edusp, 1997. 458 p.

Kelton, W. D.; Sadowski, R. P.; Sadowski, D. A. **Simulation with Arena**. 2. ed. [S. L.]: Mac Graw Hill, 1998. 611 p.

MANCINI, A.L; BARIONI, L.G; LIMA, H.N; SANTOS, J.W.; SILVA, R.D.R; SANTOS, E.H, DIAS, F.R.T. **Arcabouço para desenvolvimento de simuladores de sistemas dinâmicos contínuos e hierárquicos**. Boletim de Pesquisa e Desenvolvimento (CNPTIA)**,** Campinas: Embrapa Informática Agropecuária, 24-Fev. 2014. <http://www.infoteca.cnptia.embrapa.br/handle/doc/981039> Data de acesso: 16 out. 2014.

Moore, A.D., Holzworth, D.P., Herrmann, N.I., Huth, N.I., Robertson,M.J. (2007). **The Common Modelling Protocol: A hierarchical framework for simulation of agricultural and environmental systems**. Agricultural Systems 95, 37–48.

STEWART, J. **Cálculo I**. 6. ed. São Paulo: Cengage Learning, 2010. 688 p.

STEWART, J. **Cálculo II**. 5 ed. São Paulo: Thomson, 2006. 581 p.

ZEE, Durk-jouke van Der; HOLKENBORG, Bart; ROBINSON, Stewart. Conceptual modeling for simulation-based serious gaming. **Decision Support Systems.** [S. L.], p. 33-45. maio 2011.