UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
SISTEMAS DE INFORMAÇÃO

UM JOGO DIGITAL COMO FERRAMENTA DE APOIO NO ENSINO E PRÁTICA DE MINERAÇÃO DE DADOS

TRABALHO DE CONCLUSÃO DE CURSO

Julio Pinto Coelho Ribeiro Jardim

Santa Maria, RS, Brasil

2019

UM JOGO DIGITAL COMO FERRAMENTA DE APOIO NO ENSINO E PRÁTICA DE MINERAÇÃO DE DADOS

Julio Pinto Coelho Ribeiro Jardim

Trabalho de Conclusão apresentado ao Curso de Sistemas de Informação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Sistemas de Informação

Orientador: Prof. Dr. Joaquim Vinícius Carvalho Assunção

Santa Maria, RS, Brasil

2019

Jardim, Julio Pinto Coelho Ribeiro

Um jogo digital como ferramenta de apoio no ensino e prática de Mineração de Dados / por Julio Pinto Coelho Ribeiro Jardim. – 2019.
43 f.: il.; 30 cm.

Orientador: Joaquim Vinícius Carvalho Assunção
Monografia (Graduação) - Universidade Federal de Santa Maria, Centro de Tecnologia, Curso de Sistemas de Informação, RS, 2019.

1. Mineração de Dados. 2. Jogo Digital. 3. Python. 4. Pygame. 5. R. I. Assunção, Joaquim Vinícius Carvalho. II. Título.

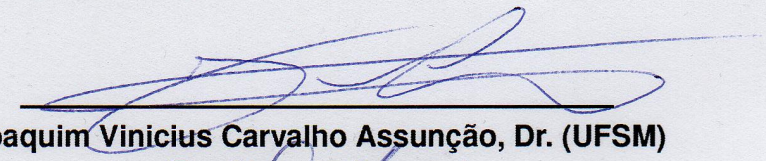
E-mail: jcoelho@inf.ufsm.br

Julio Pinto Coelho Ribeiro Jardim

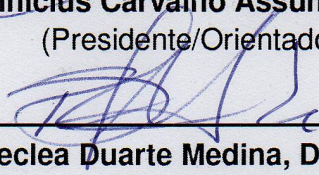
UM JOGO DIGITAL COMO FERRAMENTA DE APOIO NO ENSINO E PRÁTICA DE MINERAÇÃO DE DADOS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Sistemas de Informação.

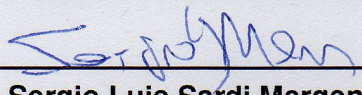
Aprovado em 4 de julho de 2019:



Joaquim Vinicius Carvalho Assunção, Dr. (UFSM)
(Presidente/Orientador)



Roseclea Duarte Medina, Dr. (UFSM)



Sergio Luis Sardi Mergen, Dr. (UFSM)

Santa Maria, RS
2019

Ao meu avô Rubens, e ao meu tio Álvaro. Ao avô pela influência na vontade de estudar engenharias (que, feliz ou infelizmente acabou virando vontade de estudar informática), pelos passeios de barco, por sempre tentar agradar os netos. Ao tio, pelo bom humor cativante, o sorriso contagioso, a influência na música e as boas horas de conversas sem sentido. Os dois fizeram uma soma imensa na minha vida, e agora só podem fazer falta.

AGRADECIMENTOS

Agradeço ao grupo TauraBots, pelas oportunidades de aprendizado, pelas possibilidades de participar das competições nacionais e internacionais de robótica, pelas amizades ali feitas, pelas viagens de 30 horas em um ônibus onde, infelizmente, só tocava pagode. Vou sempre dar o devido valor a esses anos de experiência acumulados nesse grupo.

Agradeço aos professores Rodrigo Guerra, cujo incentivo no grupo de robótica me ajudou a me aperfeiçoar nos estudos, tanto da robótica quanto de outras aplicações da computação, e ao meu orientador, Joaquim Assunção, que aceitou o desafio de orientar um trabalho de graduação de um aluno desconhecido. Espero que considere satisfatória a conclusão deste trabalho.

Aos amigos que me ajudaram e me apoiaram durante todos esses longos e cansativos anos de graduação, que não só estavam presentes durante os bons momentos, mas também me ajudaram a passar por fases difíceis (piada não planejada), junto dos quais eu me sentia parte de outra família. Em especial, aos amigos Matheus Figueiredo, e seus (meus) pais, Marco e Adriana, e ao amigo Felipe Bertagnoli, que durante esses todos anos se motraram verdadeiros amigos. E por último, mas não menos importante, ao amigo Endri Melegarejo, que, após mais da metade da minha vida em tempo de amizade, já posso considerar como um irmão. A esses, minha eterna gratidão.

À minha família (biológica), que, por mais que perguntasse constantemente "quando vai se formar?", tiveram a paciência de entender que cada um funciona em uma velocidade diferente. Essas pessoas acham que eu não fiz mais do que a minha obrigação, então o agradecimento está implícito.

E por fim, agradeço à minha namorada, Mirian Pacheco, que aguentou relutantemente a minha falta de tempo nesse último semestre da graduação.

RESUMO

Trabalho de Conclusão de Curso
Sistemas de Informação
Universidade Federal de Santa Maria

UM JOGO DIGITAL COMO FERRAMENTA DE APOIO NO ENSINO E PRÁTICA DE MINERAÇÃO DE DADOS

AUTOR: JULIO PINTO COELHO RIBEIRO JARDIM
ORIENTADOR: JOAQUIM VINÍCIUS CARVALHO ASSUNÇÃO
Local da Defesa e Data: Santa Maria, 4 de julho de 2019.

O ensino de Mineração de Dados, por ser muito similar à estatística, envolve bastante teoria e cálculo, o que acaba por tornar as aulas menos atrativas para os alunos. Isso é, também, devido à falta de trabalhos práticos que demonstrem visualmente os resultados. Geralmente as disciplinas práticas conseguem, por causa dos trabalhos propostos, atrair mais a atenção dos acadêmicos. Com o intuito de tentar minimizar esse problema, propomos o uso de um jogo digital. A decisão do desenvolvimento de um jogo se deu por conta da facilidade de entendimento, e do fato de jogos serem intuitivos ao ponto de, muitas vezes, não ser necessário fornecer explicações básicas sobre o funcionamento dos mesmos. Há algumas plataformas para práticas de mineração de dados, porém cada uma apresenta uma motivação diferente. O foco deste trabalho é um jogo, cujo objetivo é fomentar a aplicação de técnicas de todas as etapas do *KDD*, para servir como motivador. Para que possa ser usado dessa forma, o jogo apresenta uma mecânica de jogo simples, com regras claras. O jogador é capaz de discernir claramente as diferentes interações entre os personagens, para que possa, posteriormente, colocar em prática os seus conhecimentos obtidos em aulas. Com isso em mente, foi projetado um jogo de combate em plataforma 2D, para que a apresentação das interações entre os personagens seja feita de forma rápida e clara. Assim, o jogador poderá comparar seus resultados mais precisamente com o resultado esperado. Para facilitar a implementação do jogo, foi escolhida a linguagem *python*, com a biblioteca gráfica *pygame*. Isso possibilita, também, que o jogador faça as modificações que achar necessárias no jogo. Apesar de o jogo apresentar melhora no desempenho durante a execução com uso de *dataset* gerado com base em algoritmo de mineração de dados, ainda é necessário realizar testes mais extensos para averiguar a performance geral. Todavia, ao final da execução do jogo, espera-se que o jogador tenha uma ideia mais clara sobre o funcionamento dos algoritmos de mineração de dados que estudar, ao ter visualizado uma interação mais eficiente entre os personagens. A vitória no jogo deve, também, motivar o jogador fazer aplicação de outros algoritmos de mineração de dados.

Palavras-chave: Mineração de Dados, Jogo Digital, Python, Pygame, R.

ABSTRACT

Undergraduate Final Work
Graduation Program in Informatics
Federal University of Santa Maria

A VIDEO GAME AS A TOOL FOR ASSISTING THE TEACHING AND PRACTICE OF DATA MINING

AUTHOR: JULIO PINTO COELHO RIBEIRO JARDIM
ADVISOR: JOAQUIM VINÍCIUS CARVALHO ASSUNÇÃO
Defense Place and Date: Santa Maria, July 4th, 2019.

The teaching of Data Mining, for being very similar to statistics, involves a lot of theory and calculations, which makes the classes less attractive to the students. That is, also, because of the lack of practical assignments that show a visual feedback to the results. Usually, the hands-on lectures gather more attention, because of the suggested assignments. With the intention of trying to solve this problem, we propose the usage of a video game. The decision for the development of a game was because of the easy comprehension, and the fact that games are so intuitive, to the point of not needing a previous explanation about the most basic functionalities. There are some platforms for data mining practicing, although each one shows a different approach to the problem. The goal of this work is a game, focused on promoting the application of all of KDD stages' techniques, serving as a motivator. For it to be used as such, the game needs to have a simple mechanic, with clear rules. The player is able to clearly distinguish the different interactions between the characters, so they can, afterwards, apply the knowledge they acquired in class. With that in mind, the game was designed to be a 2D platform combat game, so the interactions between characters can be shown as quickly and clear as possible. This way, the player can compare more precisely their results with the expected results. To make implementation easier, the chosen programming language was *python*, using the graphical library *pygame*. That also allows the player to make adjustments to the game as they see fit. Although the execution of the game, using a data mining generated dataset as input, showed an improvement on the performance, there is still the need to run extensive tests, in order to evaluate the general performance. Even so, at the end of the game execution, we hope the player can have a clearer idea about the application of the data mining algorithms, after having seen a more efficient interaction between the characters. The victory in the game should also induce the player to apply other data mining algorithms.

Keywords: Data Mining, Video game, Python, Pygame, R.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ANEXOS

LISTA DE ABREVIATURAS E SIGLAS

DM	*Data Mining*
KDD	*Knowledge Discovery in Databases*
UFSM	Universidade Federal de Santa Maria

SUMÁRIO

1 INTRODUÇÃO

Durante a graduação, é comum a falta de motivação em continuar os trabalhos, principalmente os puramente teóricos, onde o discente não consegue ter um *feedback* visual dos resultados de seus estudos. Em cursos de computação e informática, esse fenômeno acontece com mais frequência nas disciplinas nas quais os trabalhos propostos não propõem aplicações reais dos conteúdos propostos na ementa. Isso contribui para um aumento da taxa de evasão do curso, e, consequentemente, reduz o aproveitamento das disciplinas.

O estudo de Mineração de Dados, *Knowledge Discovery in Databases* (KDD), *Data Science*, e áreas afim, envolve bastante cálculo estatístico, e normalmente os algoritmos têm resultado puramente numérico, geralmente representando uma distribuição dos dados no conjunto trabalhado, o que, posteriormente, é utilizado para alguma análise, com o objetivo de descobrir alguma relação não trivial e ainda desconhecida sobre os elementos dessa base de dados.

Durante a pesquisa, foram encontrados alguns jogos, ou plataformas, que visam fomentar o estudo de inteligência artificial e/ou ciência de dados. Um desses jogos é o *RoboCode*, cujo objetivo é desenvolver um *bot*[1] para controlar um carro de combate, para competir com outros, e a batalha acontece em tempo real. O problema, porém, é que o desenvolvedor do *bot* deve programar todo o comportamento do robô, o que torna o processo muito trabalhoso, além de não ser focado no problema em questão.

Em contraste com a ideia proposta pelo *RoboCode* (ROBOCODE, 2007), existe uma plataforma chamada *Kaggle* (KAGGLE, 2010). Nessa plataforma, é possível entrar em competições, cujo objetivo é minerar dados e descobrir conhecimentos sobre os mesmos. Infelizmente, o retorno visual que essa plataforma oferece é uma medida de erro do arquivo submetido em relação aos dados esperados. Além da avaliação simplista, a plataforma não é feita para aprendizagem, mas competição.

O problema, então, é que nenhuma dessas ferramentas implementa, simultaneamente, um ambiente gráfico com resposta em tempo real, junto com a possibilidade de exportar dados para a prática de técnicas de Mineração de Dados. Vê-se então, a necessidade de projetar tal ferramenta.

Com isso em mente, este trabalho propõe o desenvolvimento de um jogo digital, afim de servir de fator motivacional e de instrução. Durante as aulas ou trabalhos, especificamente da

[1] O *bot* do Robocode é feito exclusivamente em Java ou .NET

disciplina de Mineração de Dados, os acadêmicos podem fazer uso do jogo, para que tenham um objetivo concreto ao realizar os estudos das teorias e técnicas. Utilizando essas técnicas nas definições de movimentos dos personagens na tela, o jogador pode ter o *feedback* visual em tempo real do desempenho do algoritmo implementado. Esse trabalho descreve um ambiente gráfico de jogo, porém em um jogo de plataforma, em conjunto com a parte de análise de *datasets* providos pelo usuário, similarmente aos desafios propostos pelo *Kaggle*.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo são expostos os conceitos básicos nos quais este trabalho se baseia.

2.1 *Jogos Digitais*

Um dos principais aspectos que diferem jogos digitais de jogos analógicos é a forma como eles podem ser ensinados dinamicamente ao jogador. Em jogos analógicos, como por exemplo jogos de tabuleiro ou cartas, o jogador necessita ler um manual de instruções a respeito do jogo, para que possa, após entender todas as regras, aplicar seus recém obtidos conhecimentos para jogar o jogo. Isso posterga o início do jogo até o momento em que a leitura do livreto de regras, ou a explanação por algum terceiro, é encerrada. Jogos digitais, por outro lado, podem empregar uma forma diferente de ensinar o jogo: Utilizando da própria dinâmica do jogo, a mecânica e as regras do mesmo podem ser repassadas intuitivamente ao jogador. Podemos usar como exemplo o estágio inicial de um popular jogo do console *SNES*, chamado *"MegaMan X"*. Nesse estágio inicial, apresentado na figura 2.1 o jogador aprende, sem nenhuma informação textual, sobre as mecânicas do jogo, e as regras sobre sofrer dano, pular, etc.

Outro aspecto importante dos jogos digitais é a dinamicidade do jogo. Dependendo do tipo de jogo, o jogador não precisa passar tempo esperando alguma ação ser feita (salvo nos casos onde há conversas, telas de carregamento, ou *cutscenes*), e pode, então, desfrutar de um jogo mais rápido, no qual não há interrupções que desviem sua atenção.

A figura 2.1 ilustra os primeiros elementos que o jogador reconhece no jogo *"MegaMan X"*. À esquerda da tela, há uma barra de vida do personagem. Ao receber dano, a barra reduz em um tamanho determinado, e o jogador reconhece, então, que existem perigos no jogo, que devem ser evitados. O personagem é um *ciborgue* azul, e, assimilando as características antropomórficas do mesmo, o jogador o assimila rapidamente como o personagem principal do jogo.

A finalidade principal dos jogos, tanto digitais quanto analógicos, é o entretenimento. Outros usos podem, contudo, serem incutidos aos jogos, o que ampliam as possibilidades de uso dos mesmos. Isso acaba por transformá-los em ferramentas a serem empregadas em diversos cenários. Um desses possíveis usos é como ferramenta para auxílio na educação.

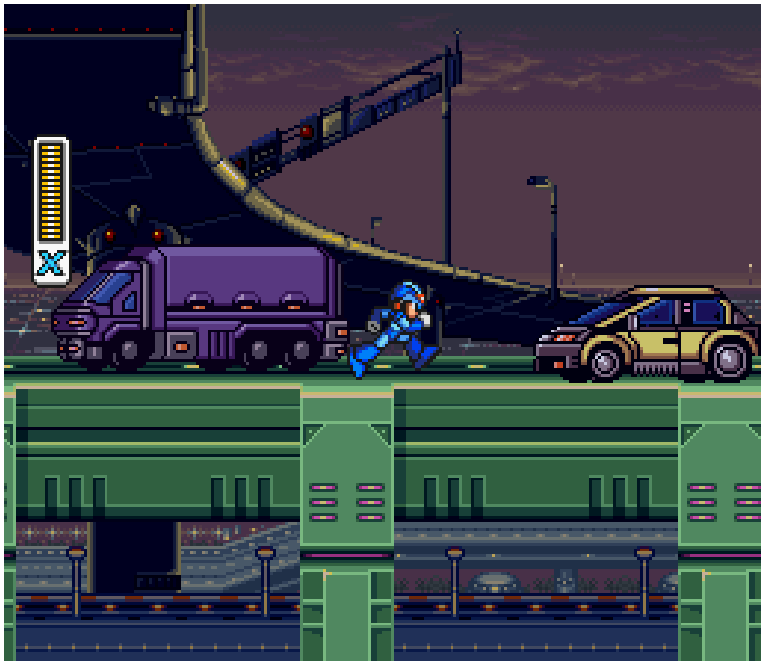


Figura 2.1: Tela do estágio inicial do jogo MegaMan X

2.2 Jogos na Educação

Os jogos digitais, além do propósito inicial do entretenimento, podem ser utilizados como ferramenta poderosa no auxílio ao aprendizado. Muitos jogos já foram criados com o intuito de ensinar, principalmente crianças, em diversas áreas do conhecimento. Um bom exemplo de jogo educativo é *A Montanha do Tesouro*, apresentado na figura 2.2, desenvolvido pela The Learning Company. O jogo apresentava desafios matemáticos, e a solução dos desafios recompensava o jogador com algum tesouro. O jogador, então, se sentia motivado a resolver os problemas matemáticos, e avançava gradativamente o nível de dificuldade.

Figura 2.2: Tela do jogo A Montanha do Tesouro

De acordo com (TAROUCO et al., 2004), "Os jogos podem ser ferramentas instrucionais eficientes, pois eles divertem enquanto motivam, facilitando o aprendizado e aumentam a capacidade de retenção do que foi ensinado, exercitando as funções mentais e intelectuais do jogador". Os jogos educacionais são constituidos, de acordo com (DEMPSEY; RASMUSSEN; LUCASSEN, 1996), por "qualquer atividade de formato instrucional ou de aprendizagem que envolva competição e que seja regulada por regras e restrições".

Essa possibilidade de uso no aprendizado e treino motivou o desenvolvimento deste trabalho, que se dá por um jogo, feito em Python e Pygame, para servir de apoio nas práticas de Mineração de Dados.

2.3 *Python* e *Pygame*

Esse trabalho foi dividido em duas partes: O desenvolvimento de um jogo digital, no estilo plataforma 2D, e os testes com algumas técnicas de *KDD* e *DM*. Para a parte de implementação do jogo, foi escolhida a linguagem Python, pela simplicidade e facilidade de uso, além de ser uma linguagem de alto nível de abstração. Por ser uma linguagem interpretada, tudo o que é necessário para a execução do jogo é um interpretador *Python*, na versão 3.x, e a instalação da biblioteca *Pygame*. Isso torna desnecessária a compilação a cada modificação, e permite que o jogo seja executado em qualquer máquina.

A linguagem por si só não conta com bibliotecas gráficas por padrão. A biblioteca escolhida pra trabalhar com gráficos é a *Pygame*. *Pygame* é um conjunto de módulos em *Python*, que encapsulam a biblioteca *Simple DirectMedia Layer* (SDL), escrita na linguagem C, possibilitando o desenvolvimento fácil e acessível de jogos e programas com ambientes gráficos.

Apesar do desempenho consideravelmente inferior, quando comparado à linguagem C, Python é uma linguagem de uso extremamente fácil, com várias sub-rotinas já implementadas, o que possibilita a implementação mais rápida de aplicações. Por ter um alto nível de abstração, geralmente os códigos-fonte de programas escritos em Python são muito mais compactos que os códigos para os mesmos programas escritos em C. Ao comparar com Lua, vemos que ambas as linguagens têm alto nível de abstração, mas, por não dispor de mecanismos nativos para implementação de classes, Lua acaba exigindo um estudo muito mais profundo pra que o programador possa escrever um código com resultado similar ao de um programa escrito em Python. Além disso, a biblioteca mais utilizada em Lua para criação de games, Love2D, requer que, a cada modificação, o programador re-comprima os arquivos para execução e teste. Esses

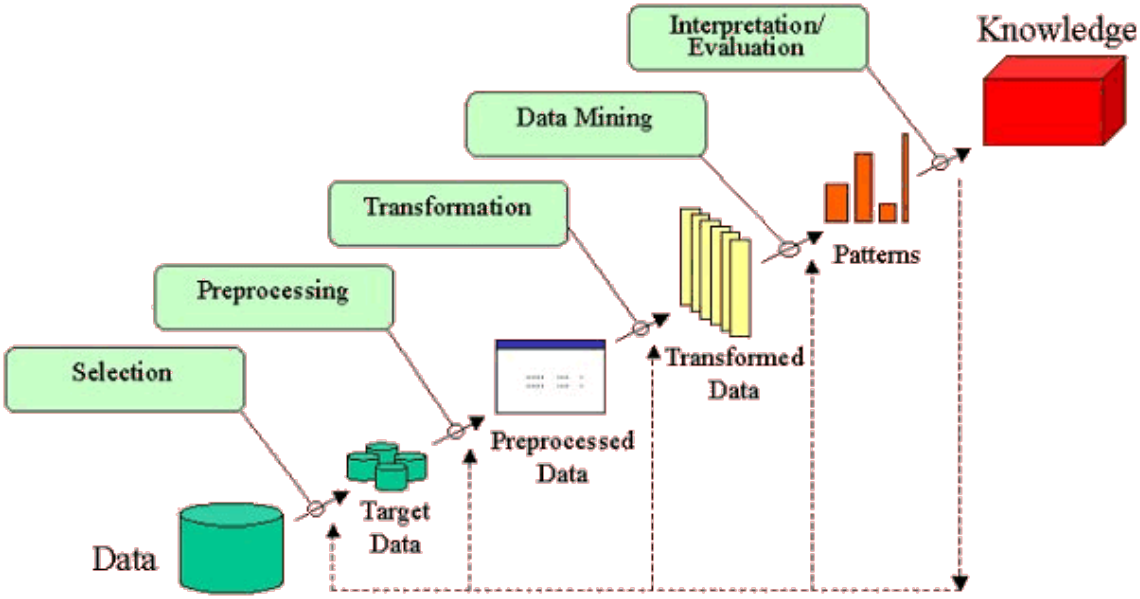


Figura 2.3: *Knowledge Discovery in Databases (FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 1996)*

foram os principais motivos para a escolha da linguagem Python neste trabalho.

2.4 Mineração de Dados

Mineração de Dados é o processo de automaticamente descobrir informações úteis em grandes repositórios de dados. Técnicas de mineração de dados são utilizadas para vasculhar grandes bases de dados, para encontrar padrões novos e úteis que, de outras formas, continuariam desconhecidos (TAN; STEINBACH; KUMAR, 2006). Esse é um dos sub-processos do *Knowledge Discovery in Databases (KDD)*, que consiste na obtenção de conhecimentos a partir de dados brutos. O processo é ilustrado pela figura 2.3.

A figura mostra a execução de 5 passos, necessários para a obtenção de algum conhecimento previamente desconhecido, em uma base de dados. Essas etapas são:

1. Selecionar os dados a ser analisados *(Selection)*

   Para cada aplicação, devemos escolher as colunas da base de dados que serão relevantes. Esse passo de escolha dos dados a serem utilizados é chamado de **Seleção**.

2. Remover *noises* e decidir estratégias sobre como proceder com os dados em sequência temporal *(Preprocessing)*

   Após a seleção dos dados a serem utilizados, deve-se fazer a limpeza dos mesmos, para que não existam valores indefinidos ou de tipos errados na tabela. Esse passo é chamado

de **Pré-processamento**.

3. Reduzir ou transformar o *dataset (Transformation)*

   O próximo passo consiste em manipular a tabela de forma que possa ser utilizada pelo algoritmo a ser aplicado. Isso pode envolver operações como transposição, normalização, entre outros. A fase de manipulação da tabela é chamada de **Transformação**

4. Encontrar padrões de interesse, aplicar regras de classificação e associação *(Data Mining)*

   Executa-se algum algoritmo que fará a associação, classificação, *clusterização*, etc., dos dados presentes na tabela. Essa é a **Mineração de Dados**

5. Interpretar os dados e gerar um arquivo a ser utilizado no jogo *(Evaluation)*

   Após a obtenção dos resultados da aplicação do algoritmo de Mineração de Dados escolhido, esses resultados devem ser avaliados. Sem isso, não há obtenção de conhecimentos dessa base de dados. Esse passo é chamado **Interpretação**

Das técnicas de Mineração de Dados existentes, foi utilizado o algoritmo *Apriori* para os testes de funcionalidade do jogo.

2.4.1 Apriori

Apriori é o primeiro algoritmo de mineração de regras de associação, que foi pioneiro no uso de poda baseada em suporte, para sistematicamente controlar o crescimento exponencial de itens candidatos (TAN; STEINBACH; KUMAR, 2006). O controle desse crescimento é feito baseado em fatores: suporte e confiança. Suporte é a quantidade de vezes que um determinado item ou *itemset* aparece no *dataset*, e é calculado como:

$$support = \frac{freq(X,Y)}{N}$$

, onde X e Y são dois itens do itemset, e N é o total de itens. Podemos definir um mínimo de ocorrências de um item para definirmos o mesmo como relevante. Confiança é a frequência em que a ocorrência de um dado conjunto A implica na ocorrência de um conjunto B, e é calculado através da fórmula:

$$confidence = \frac{freq(X,Y)}{freq(X)}$$

Além desses dois fatores, também é usada uma medida de performance do modelo utilizado, chamada de *lift*, ou sustentação. Essa medida faz uso da fórmula de suporte, sendo sua fórmula igual a:

$$lift = \frac{support}{support(X) \times support(Y)}$$

Utilizando-se desse algoritmo, itens que aparecem com pouca frequência no *dataset*, e itens que não frequentemente implicam em outros, são descartados, por não apresentarem relevância para o dado problema. Isso possibilita a determinação de um relacionamento entre itens ou *itemsets* dentro do *dataset*.

O processo de poda é ilustrado na figura 2.4, na qual vemos o processo de poda na remoção dos itens com suporte inferior a 3, gerando a tabela final de *itemsets* com apenas um elemento. Na primeira tabela, denominada "Candidate 1-itemsets", os elementos "Cola"e "Eggs", por possuirem suporte inferior a 3, que o exemplo da figura define como mínimo, são removidos do itemset. Em seguida, os itemsets combinados de "Beer+Bread"e "Beer+Milk"são removidos da segunda tabela, de nome "Candidate 2-itemsets". Por fim, o itemset resultante dessa poda é apresentado na tabela "Candidate 3-itemsets", sendo apenas o conjunto "Bread+Diapers+Milk".

2.5 Trabalhos Relacionados

O trabalho de (SYBERFELDT; SYBERFELDT, 2010) apresenta um jogo sério, onde o jogador, atuando como o gerente de produção de uma fábrica de Lego, deve otimizar a produção. Assim como em qualquer jogo, o jogador é recompensado caso atinja o objetivo especificado. Tal objetivo se apresenta como sendo virtualmente impossível para um humano, e a ideia é que o jogador perceba que deve utilizar algum algoritmo de aprendizado de máquina para solucionar o problema. Esse jogo sério faz o uso de um desafio real para exemplificar a necessidade, e promover o estudo de algoritmos de aprendizado. O desafio proposto, porém, difere deste trabalho, principalmente no que se refere ao cenário da aplicação. A fábrica de Lego é um cenário que deve ser montado usando peças mecânicas e eletrônicas, e envolve um grande custo financeiro. Ao contrário disso, um jogo digital pode facilmente ser executado em computadores, e, caso o jogador não possua um, poderá fazer uso de laboratórios informatizados de sua faculdade.

De forma muito similar ao trabalho aqui proposto, o jogo de (SYBERFELDT; SYBERFELDT, 2010) apresenta um ambiente onde é possível tentar encontrar soluções manualmente, porém essas soluções dificilmente atingirão um desempenho considerável. A figura 2.5 mostra uma fotografia do jogo da fábrica de lego, utilizada em tal trabalho.

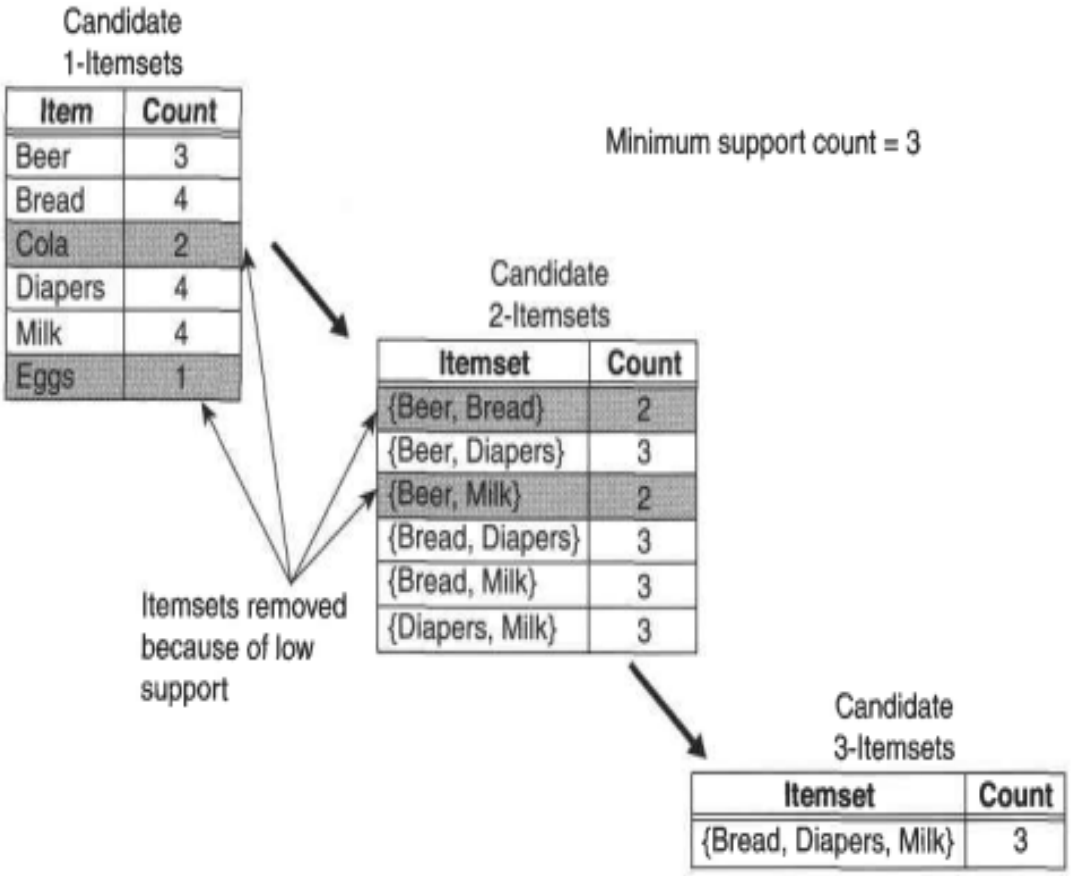


Figura 2.4: Geração de *itemset* frequente com o algoritmo Apriori (TAN; STEINBACH; KUMAR, 2006)

Em um aspecto mais básico, outro trabalho cuja finalidade se assemelha a este proposto, chamado Batalha Matemática (MORAES; COLPANI, 2016), apresenta um jogo em Realidade Aumentada, a ser utilizado para o auxílio na aprendizagem de matemática básica. Apesar de não ser voltado ao ensino de computação, o jogo também faz uso de tecnologias para motivar o aluno, tornando o ensino mais instigante e significativo. A figura 2.6 apresenta uma das fases do jogo Batalha Matemática.

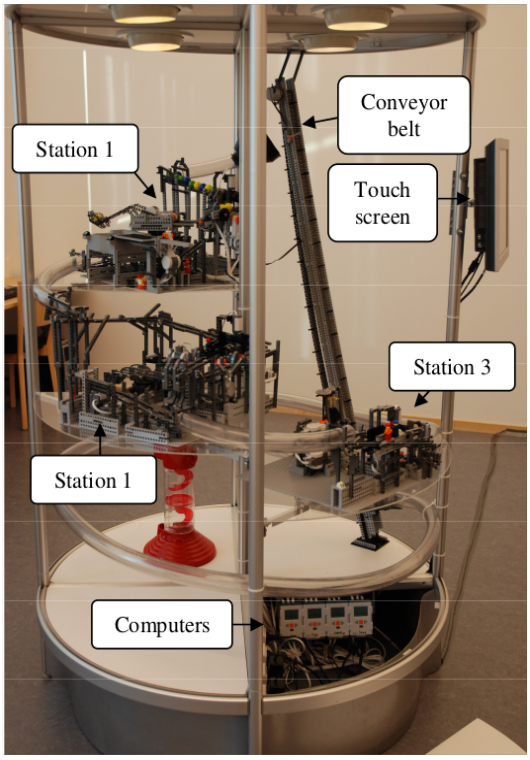


Figura 2.5: Fotografia do jogo de fábrica de lego (SYBERFELDT; SYBERFELDT, 2010)

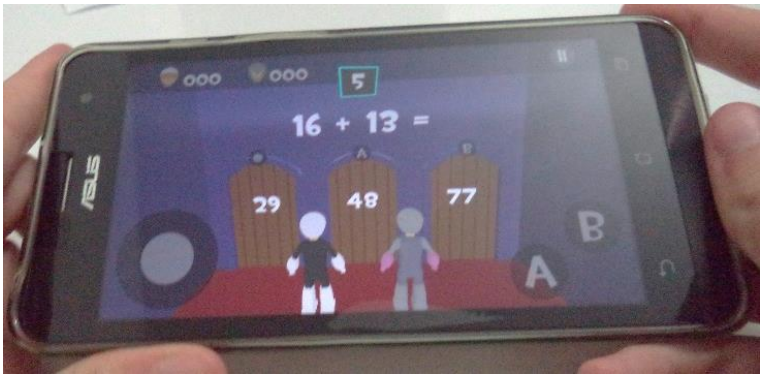


Figura 2.6: Fotografia do jogo Batalha Matemática executando em um smartphone (MORAES; COLPANI, 2016)

3 METODOLOGIA

A proposta do jogo é oferecer um incentivo ao acadêmico, para que ele possa visualizar os resultados e desempenhos de seus algoritmos. O jogo aqui proposto combina alguns elementos das duas ferramentas anteriormente discutidas, tornando mais prazerosa a forma como o acadêmico põe em prática seus estudos.

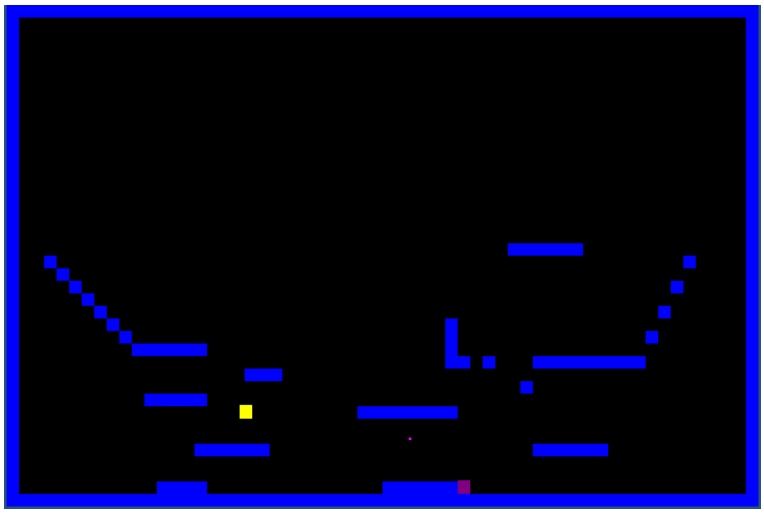


Figura 3.1: Protótipo inicial para o jogo Jump'n Shoot Man, com 2 jogadores

Para esse objetivo, criamos um jogo que exporta dados sobre o estado do mesmo, apresentando ao jogador na forma de um arquivo *CSV*, e que possa receber um arquivo de entrada, com os dados já treinados, para a execução do mesmo. A escolha entre usar ou não um arquivo de entrada é do jogador, e, caso escolha não utilizar, poderá implementar uma sub-rotina de movimentação aleatória do personagem. A figura 3.1 mostra o jogo com 2 personagens, representados pelos quadrados amarelo e roxo. Entre eles, há um projétil (pequeno quadrado roxo), disparado pelo personagem roxo.

A respeito dos dados exportados, foram selecionados os que consideramos mais relevantes para o desenrolar da partida. São eles: posições, orientações, movimentação atual, disparo, e pontuação dos dois jogadores. Posição é composta de dois valores numéricos, representando a localização do jogador em um plano cartesiano bidimensional. Orientação indica o último comando de movimentação horizontal do jogador, para que seja decidida a direção do disparo. Movimentação atual é exportada tanto para fins de análise, como é importada para selecionar a ação baseada na comparação de estados feita pelo jogador. Disparo é um valor *booleano*, indicando se houve ou não disparo naquele momento; Pontuação é cumulativa, resetando apenas ao

reiniciar o jogo. Todos os dados são concatenados ao final de um arquivo, não sendo criado um arquivo separado para cada partida.

3.1 Requisitos

O planejamento deste jogo foi inspirado, principalmente, no modo *"battle"* do jogo *"Super Mario Bros. 3"*, sendo, então, projetado para ser um jogo plataforma 2D. Os jogadores competirão entre si, ou com um *bot*, em um combate de tiro, sendo a pontuação referente à quantidade de disparos que acertaram o oponente. A ideia de usar esse formato de jogo se deve à simplicidade e a facilidade de entendimento das funcionalidades e mecânicas do mesmo.

O jogo foi feito para dois jogadores, mas é possível inserir mais de dois personagens na partida. Caso escolha-se utilizar mais do que dois personagens, os dados da partida não serão salvos, afim de manter a consistência no *dataset*. Esses personagens extras apenas podem ter movimentação aleatória, visto que a comparação de estados no controlador é feita apenas com jogadores 1 e 2.

3.2 Escopo do trabalho

Este trabalho se deu pelo projeto e implementação de um jogo, a ser utilizado para estudo de Mineração de Dados, com foco no algoritmo de associação *"Apriori"*, citado anteriormente. Em geral, algoritmos de Mineração de Dados e Inteligência Artificial que possam associar um conjunto de entradas a um conjunto de saídas devem apresentar um desempenho satisfatório para este trabalho. Algoritmos de clusterização não fazem parte do uso planejado para este trabalho.

O trabalho também não visa ensinar conceitos de Mineração, assim como não pretende ensinar algoritmos. Deve ser usado apenas como ferramenta auxiliar, para que o acadêmico possa visualizar o desempenho da aplicação dos algoritmos de classificação vistos em aulas ou em seus estudos individuais.

3.3 Implementação

A implementação do jogo faz a separação de funcionalidades básicas de cada módulo em uma classe distinta. Espera-se, com isso, que o acadêmico que queira fazer modificações na implementação consiga encontrar facilmente as sub-rotinas a serem modificadas. Algumas

classes foram agrupadas, dentro de diretórios, e não dentro do mesmo arquivo, por semelhança de funcionalidade ou por ser utilizada em apenas um módulo. Cada classe que age diretamente na exibição dos elementos implementa uma sub-rotina de desenho na tela, e uma sub-rotina de atualização.

3.3.1 *Classes*

O jogo pode ser dividido em três partes, apresentadas nas figuras 3.2 e 3.3. Essas partes básicas são: Game, Player e Floor.

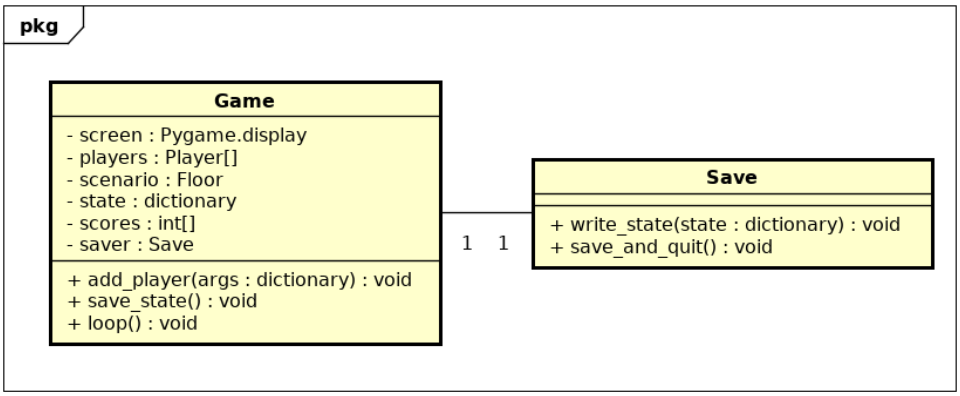


Figura 3.2: Diagrama da classe Game

O cenário consiste de blocos quadrados de dimensão igual a dos personagens, organizado de modo a formar linhas e colunas, representando chão, plataformas e parede. O jogador possui sub-rotinas de mecânica de movimentação, ciclos de vida, disparos e uma movimentação aleatória. Por fim, o controlador de estados é responsável por fazer as comparações entre os estados importados do arquivo fornecido pelo jogador, e os estados do jogo a cada momento. Nessa sessão serão explicados os módulos do jogo, com suas respectivas funcionalidades.

3.3.1.1 A classe *Game*

Game define a classe que inicializa, configura e executa o jogo. O uso básico dessa classe é: Inicializar a biblioteca *PyGame*, instanciar o cenário, instanciar os jogadores, configurar os jogadores, e invocar o *loop* principal, no qual, a cada 1/60 de segundo, ou um *frame*, são invocados os métodos de atualização dos personagens, e redesenho da tela. Essa classe também é responsável pr invocar o método de gravar estado do jogo, que acontece a cada 1/4 de segundo, ou seja, a cada 15 *frames*. Chamamos de *frame* o desenho completo da tela a cada

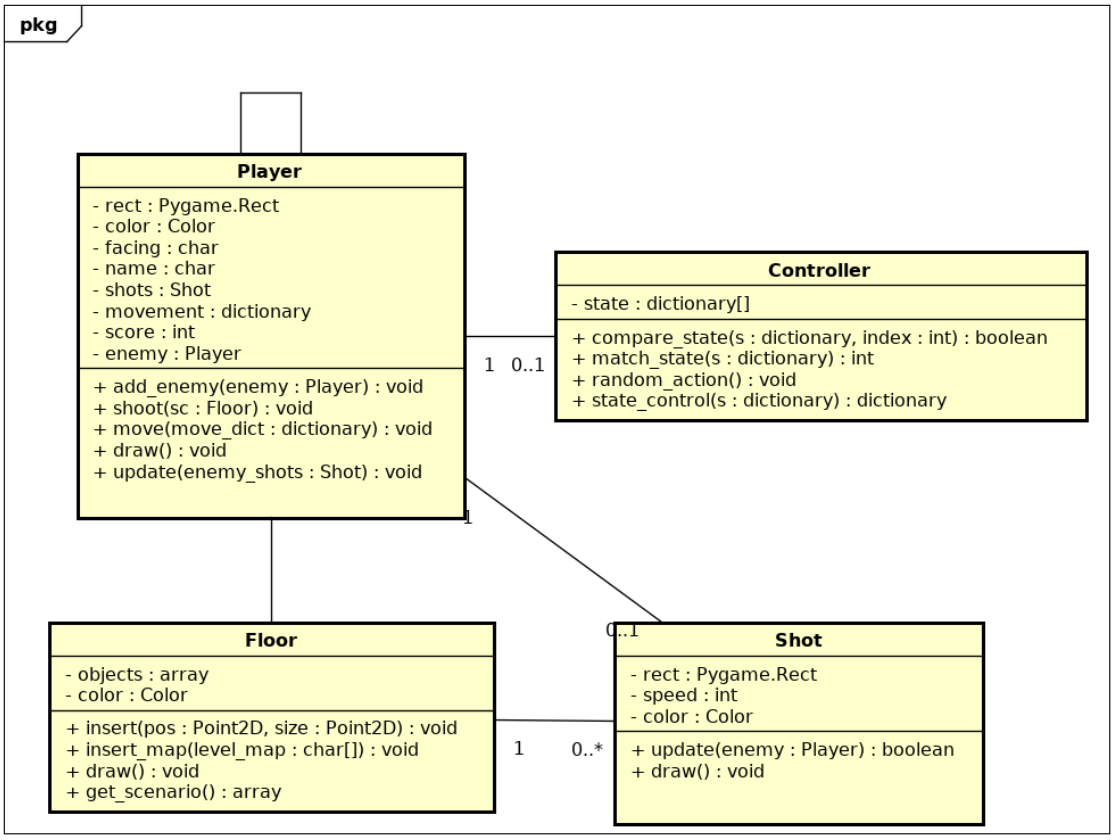


Figura 3.3: Diagrama das classes Player, Floor, Controller e Shot

iteração.

3.3.1.2 A classe *Player*

A classe *Player* abstrai o conceito do personagem, com atributos de posição, movimentação e pontuação. O atributo *screen* é inicializado na classe *Game*, e atribuido a todos os objetos da classe *Player*, para facilitar as chamadas dos métodos de desenho da bibliteca gráfica. O atributo *movement* define um dicionário de movimento, cujos índices são definidos na variável global *fieldnames*, presente na classe *Save*, que será discutida posteriormente. A seguir, um exemplo de dicionário de movimento:

```
{
"horizontal":"right",
"jump": "up,
"shoot": 1,
}
```

O atributo mais importante da classe *Player* é chamado *rect*. Esse atributo é uma instância da classe *pygame.Rect*, contendo os atributos específicos de posição e tamanho, entre outros

não utilizados no desenvolvimento desse jogo. Além disso, toda a detecção de colisão no jogo é feita com o método *pygame.Rect.colliderect*, da mesma classe (SWEIGART, 2012).

3.3.1.3 A classe *Floor*

Floor define o cenário, e possui apenas dois atributos: *objects*, que contém um *array* de objetos do tipo *Pygame.Rect*, e *color*, que define a cor base para o desenho dos elementos de cenário. A classe não conta com um médoto de atualização, visto que o cenário deve se manter imóvel, mas ainda possui um método de redesenho, visto que a cada *frame* a tela toda deverá ser redesenhada.

Além disso, a classe conta com dois métodos de definição de cenário: o primeiro é *"insert"*, onde o jogador poderá inserir manualmente elementos de cenário, que, por padrão, terão tamanho de 10x10 pixels, para ficar de acordo com a grade quadrada do jogo; o segundo método é *insert_map*, com o qual pode ser utilizado um *array* de caracteres para definir o mapa. O mapa padrão está no arquivo *baseScenario.py*.

3.3.1.4 A classe *Controller*

A classe *Controller* é utilizada para a movimentação do personagem baseada em um arquivo importado pelo jogador. Nela constam métodos de comparação de estados, que, se utilizado, fará comparações a cada troca de estado do jogo com todos os estados definidos pelo jogador no arquivo *csv*. O único atributo dessa classe é um *array* de dicionários, com os mesmos índices descritos em *fieldnames*. A leitura do arquivo é feita com a classe *csv.DictReader*, que faz o *parsing* das linhas de texto em elementos do tipo dicionário.

Caso o jogador decida implementar um método de movimento aleatório, para quando o comparador de estados não retorne uma igualdade, ele pode implementá-lo nessa classe, dentro do método *random_action*. Caso isso não seja implementado, retornará um valor Nulo, o que será interpretado pela classe *Player* como não havendo correspondência do estado atual com os estados informados, e utilizará o método implementado em *Player*.

3.3.1.5 A classe *Save*

Como as classes *Controller* e *Save* são as únicas que fazem leitura e escrita em arquivos externos ao programa, foram colocadas no mesmo módulo, chamado *state*. A classe *Save* é

responsável por salvar cada estado do jogo em uma formatação indicada na sua variável global, *fieldnames*, que é usada pra escrever o cabeçalho do arquivo *csv*, caso não exista esse arquivo. No caso de existir, cabeçalho não será re-escrito. *Save* possui apenas dois atributos: *f*, que é um ponteiro pra um arquivo, e *writer*, que é uma instância da classe *csv.DictWriter*, responsável por fazer o *parsing* do dicionário do estado do jogo, para salvar como uma linha de texto. Abaixo, um exemplo de elemento de dicionário de estado:

```
{
"p1_x": 20, "p1_y": 12, "p1_facing":"right",
"p1_horizontal":"right", "p1_vertical":"up",
"p1_shoot":0, "p1_score":0,
"p2_x": 24, "p2_y": 7, "p2_facing":"left",
"p2_horizontal":"center", "p2_vertical":"not",
"p2_shoot":1, "p2_score":3,
}
```

O formato, indicado na classe *Save*, consiste em um arquivo com os mesmos atributos do dicionário. Ao salvar o estado em uma linha na tabela, os dados representam o atributo como consta no momento do jogo. No caso da importação, os campos "p1_horizontal", "p1_vertical", e "p1_shoot"representam a ação que deve ser tomada, caso o estado presente do jogo seja semelhante ao estado da mesma linha.

Logo, uma linha no arquivo de saída cujos valores são

```
20,12,right,right,up,0,0,24,7,left,center,not,1,3
```

representa um estado do jogo, no qual o jogador 1 se encontra na posição 20,12, virado para a direita, se movendo para a direita, pulando, não atirando, e com pontuação 0. A segunda metade da linha representa os valores do jogador 2. Essa mesma linha, no arquivo de entrada, representaria um estado a ser comparado, no qual os jogadores estão nas posições 20,12 e 24,7, e selecionaria a ação de não atirar.

4 TESTES COM ALGORITMOS DE MINERAÇÃO DE DADOS

4.1 Aplicação no jogo

O processo de usar o jogo como ferramenta de auxílio se dá na seguinte ordem:

1. Execução do jogo, com ou sem um *dataset* de entrada.

   Ao executar o jogo, os dados exportados serão salvos em um arquivo de saída. Independentemente de o jogador ter usado dados de entrada, o jogo sempre exportará dados.

2. Mineração dos Dados exportados.

   Esse é o ponto principal do jogo. Os dados devem ser analisados pelos jogadores, preferencialmente automatizando com algoritmos, que devem ser selecionados, pré-processados, transformados, minerados e interpretados.

3. Formulação de um arquivo de entrada.

   O jogador deverá usar os dados já minerados do processo anterior para formular um arquivo *csv* a ser usado como entrada de dados do jogo. Esse arquivo conterá estados a serem comparados, e ações a serem tomadas.

4. Re-execução do jogo, com a entrada do arquivo formulado.

   Nesse momento, o jogador visualizará os resultados e o desempenho da forma como ele fez a aplicação dos seus conhecimentos.

A figura 4.1 ilustra o processo apresentado acima.

Os dados exportados após a execução do jogo são salvos em um arquivo chamado "output.csv", e são apresentados como na tabela 4.1. Para facilitar a visualização, foram omitidos os valores de p2, e o conjunto de dados foi reduzido.

Para fazer uso desses dados, foi implementado um algoritmo em R, utilizando a biblioteca *arules*. Tal biblioteca provê funções de tratamento, manipulação, análise e representação de dados e padrões. Além disso, conta com uma implementação em C do algoritmo Apriori.

Seguindo os passos exemplificados na figura 2.3, começamos por fazer a seleção dos dados. Neste caso, selecionamos apenas os dados de posição dos jogadores e pontuação.

```
raw_data = read.csv("output.csv")
selected_columns = c("p1_x", "p1_y", "p1_score", "p2_x", "p2_y")
```

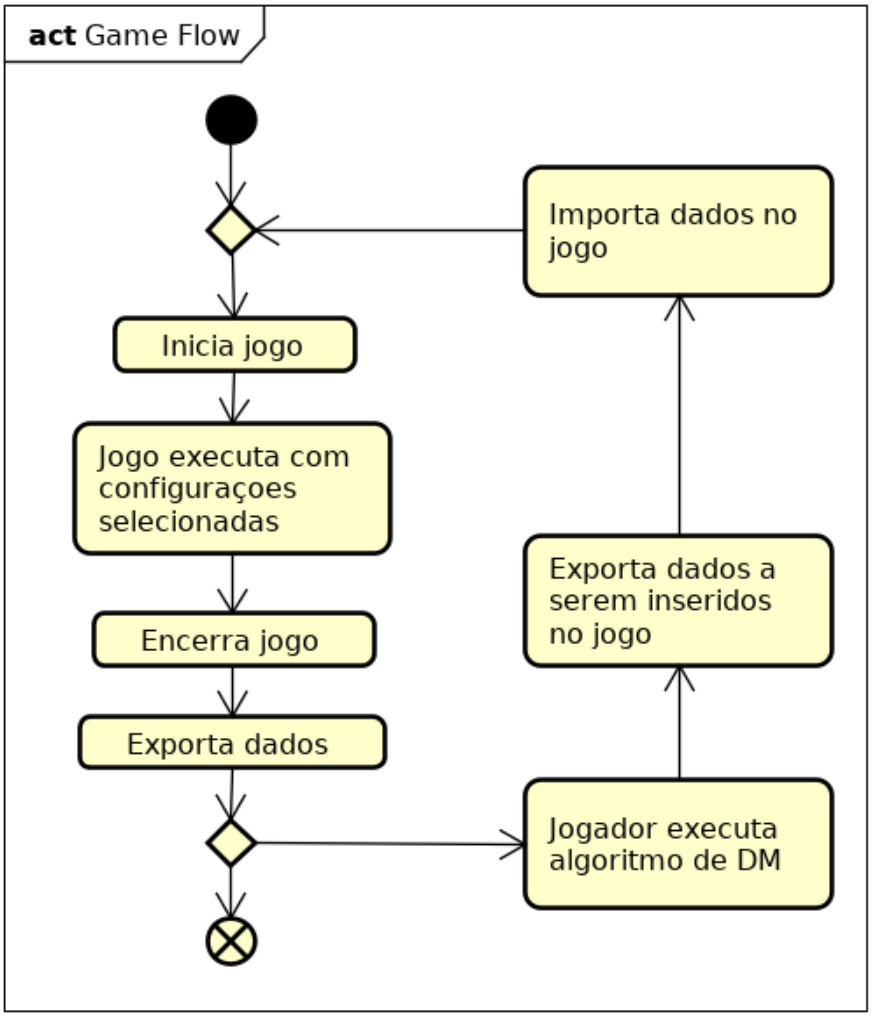


Figura 4.1: Processo de jogo e re-jogo

p1_x	p1_y	p1_facing	p1_horizontal	p1_vertical	p1_shoot	p1_score
5	11	right	center	not	0	0
8	23	right	center	not	0	0
11	26	right	center	not	0	0
14	26	right	center	not	0	0
17	27	right	right	not	0	0
20	34	right	right	not	0	0
23	38	right	right	not	0	0
26	38	right	right	not	0	0
29	38	right	center	not	1	0
25	38	right	center	not	1	0

Tabela 4.1: Tabela de dados gerados pela execução do jogo

```
selected_data = raw_data[, selected_columns]
```

Em seguida, definimos uma estratégia para proceder sobre os dados. A estratégia imple-

mentada consiste em transformar o valor de p1_score para refletir apenas se houve ou não houve acerto. Caso houver, será atribuido o valor 1, e caso não houver, 0. Essa estratégia dispensa o uso da coluna p1_shoot, tendo em vista que o incremento na pontuação só poderá ocorrer caso o tiro tenha sido disparado. Além disso, para considerar o tempo desde que o tiro foi disparado até o momento em que acertou o oponente, causando o aumento na pontuação, alteramos os valores de "p1_score"nas 3 linhas anteriores a cada ocorrência do valor "1".

```
for (i in nrow(selected_data):2){
   if(selected_data$p1_score[i] > selected_data$p1_score[i-1]){
      selected_data$p1_score[i] <- 1
   }else{
      selected_data$p1_score[i] <- 0
   }
}

for (i in nrow(selected_data)){
   if(selected_data$p1_score == 1){
      selected_data$p1_score[i-3] <- 1
      selected_data$p1_score[i-2] <- 1
      selected_data$p1_score[i-1] <- 1
   }
}
```

Após isso, selecionamos apenas as linhas onde houve pontuação, para encontrar as regras que inferem essas ocorrências. As linhas onde não houve ocorrência de pontuação são irrelevantes.

```
was_score = selected_data[,"p1_score"] == 1
selected_data = selected_data[was_score,]
```

Por fim, podemos gerar as regras de associação, com o uso da biblioteca "arules", e fazer a análise dos resultados.

```
require("arules")

rules = apriori(selected_data, parameter = list(conf=0.05, supp=0.1,
   target = "rules"))
inspect(rules)
```

A confiança e o suporte foram propositalmente escolhidos como valores baixos, pois, durante a execução do jogo com comportamento aleatório, a ocorrência de pontuação é baixa demais para gerar um dataset confiável. Obtemos, então, a tabela 4.2 de regras de associação. Assim como na tabela 4.1, algumas linhas e colunas foram omitidas para poder encaixar a tabela dentro da página.

lhs	rhs	support	confidence	lift	count
{}	{p2_y=33}	0.1	0.1	1	9
{}	{p2_x=1}	0.1	0.1	1	9
{}	{p1_y=34}	0.111111111111111	0.111111111111111	1	10
{}	{p2_y=32}	0.122222222222222	0.122222222222222	1	11
{}	{p1_y=37}	0.155555555555556	0.155555555555556	1	14
{}	{p2_x=58}	0.211111111111111	0.211111111111111	1	19
{}	{p1_y=38}	0.533333333333333	0.533333333333333	1	48
{}	{p2_y=38}	0.588888888888889	0.588888888888889	1	53
{}	{p1_score=1}	1	1	1	90
{p2_y=33}	{p1_score=1}	0.1	1	1	9
{p1_score=1}	{p2_y=33}	0.1	0.1	1	9
{p2_x=1}	{p1_score=1}	0.1	1	1	9
{p1_score=1}	{p2_x=1}	0.1	0.1	1	9
{p1_y=34}	{p1_score=1}	0.111111111111111	1	1	10
{p1_score=1}	{p1_y=34}	0.111111111111111	0.111111111111111	1	10
{p2_y=32}	{p1_score=1}	0.122222222222222	1	1	11
{p1_score=1}	{p2_y=32}	0.122222222222222	0.122222222222222	1	11
{p1_y=37}	{p1_score=1}	0.155555555555556	1	1	14
{p1_score=1}	{p1_y=37}	0.155555555555556	0.155555555555556	1	14

Tabela 4.2: Regras de associação geradas pelo algoritmo apriori. Essas regras são meramente ilustrativas da execução do algoritmo. Regras mais significativas poderiam incluir a proximidade entre os dois jogadores

Após obtermos as regras de associação, podemos descartar as regras nas quais o valor de RHS(*"Right Hand Side"*) é diferente de "p1_score=1", e as regras nas quais o valor de LHS(*"Left Hand Side"*) é nulo. Podemos, assim, construir um *dataset* baseado nas regras onde o jogador p1 obteve sucesso, para aplicar na próxima execução do jogo.

```
df = data.frame(inspect(rules))
pscore = df$rhs == "{p1_score=1}"
notnull = df$lhs != "{}"
relevant = df[pscore & notnull, ]
```

A imagem 4.2 mostra a re-execução do jogo, com o jogador 1 utilizando o conjunto de regras gerado a partir da execução do algoritmo apriori sobre o *dataset* exportado na primeira execução. Os testes foram executados com o mesmo conjunto de dados, e com duração de dois minutos. Esse tempo foi escolhido arbitrariamente, apenas para garantir a consistência das partidas. A tabela 4.3 apresenta os resultados de 20 execuções do jogo, e mostra que o jogador 1 foi vitorioso em 14 das 20 partidas. Além disso, a média de pontos obtidas após o processo do KDD foi maior em aproximadamente 40%.

A tabela 4.4 mostra os resultados de 20 execuções, utilizando a pontuação média do

jogador 1, arredondada para cima, como critério de parada. Podemos ver que o tempo para atingir a média é inferior ao tempo definido na série anterior de testes, de dois minutos. A quantidade de vitórias do jogador 1 permaneceu a mesma, com 14 jogos ganhos de 20. Como nesse tipo de execução é impossível haver empate, o número de vitórias do jogador 2, controlado por um algoritmo aleatório, subiu de 4 para 6. Os valores apresentados nas tabelas mostram que o jogo pode ser útil para a visualização da melhora no desempenho do jogador, ao aplicar técnicas de mineração de dados para o controle dos movimentos.

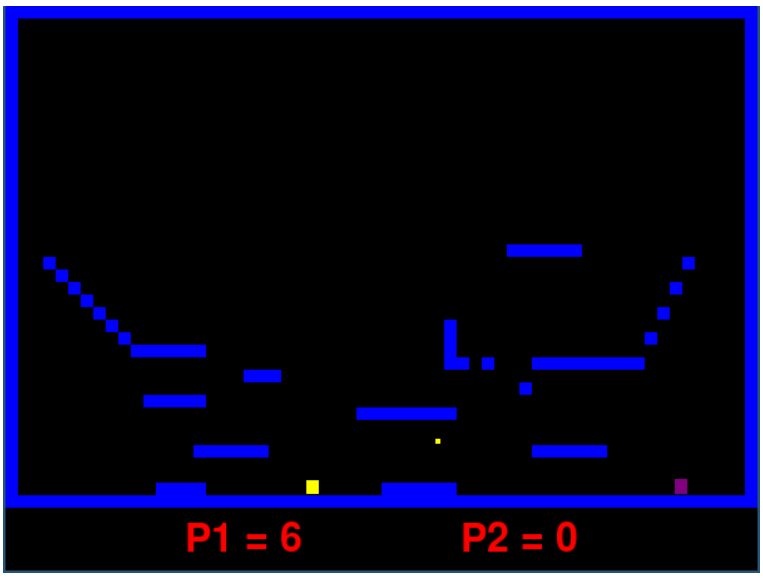


Figura 4.2: Execução do jogo, utilizando um *dataset* como arquivo de entrada para os movimentos do jogador

#	p1_score	p2_score
1	6	0
2	6	3
3	6	4
4	3	7
5	5	2
6	4	5
7	2	1
8	3	6
9	4	2
10	5	5
11	8	3
12	3	1
13	6	9
14	5	0
15	4	1
16	2	1
17	8	4
18	2	2
19	5	3
20	7	4
media	4.7	3.15

Tabela 4.3: Resultados da execução do jogo, com duração de dois minutos

#	tempo	p1_score	p2_score
1	2:12.103	5	4
2	1:42.434	5	1
3	0:57.996	5	0
4	1:12.479	5	2
5	1:48.687	5	4
6	1:21.407	5	3
7	0:43.664	5	0
8	1:9.140	2	5
9	1:4.512	5	2
10	0:55.985	5	0
11	2:47.593	3	5
12	1:56.874	2	5
13	1:43.503	5	1
14	1:27.873	1	5
15	1:23.471	3	5
16	0:47.946	5	0
17	0:51.857	5	0
18	1:20.451	5	2
19	0:55.930	5	2
20	1:54.089	2	5
media	1:24.899	-	-

Tabela 4.4: Resultados da execução do jogo, com limite de 5 pontos

5 CONCLUSÃO

O problema abordado nesse trabalho foi a necessidade de criação de uma ferramenta, que pudesse servir tanto como motivadora, quanto como auxiliadora nos estudos de mineração de dados.

O resultado final foi um jogo customizável, no qual cada jogador pode definir quais são os obstáculos e plataformas. Assim, cada configuração do jogo pode resultar em uma execução diferente. Como esperado, o jogo demonstrou melhora no desempenho do jogador, ao comparar a primeira execução, a qual utilizava decisões puramente aleatórias, com a execução baseada no *dataset*. Essa execução final utiliza uma combinação de comportamento aleatório com decisões baseadas em regras. Apesar dessa melhora no desempenho, ainda é possível que tal melhora seja apenas coincidental, visto que é possível que, em dois momentos distintos, a execução aleatória possa ter desempenhos completamente diferentes.

Para que haja a garantia de que essa melhora do desempenho seja consequência da execução de um algoritmo, é necessário fazer uma extensa série de testes, com execução do algoritmo apriori com outras configurações. É possível, também, realizar tais testes com o uso de outras categorias de algoritmos, como mencionado no capítulo 2. Afim de eliminar a aleatoriedade da re-execução, também será necessário re-executar o jogo durante periodos prolongados, assim como refazer várias vezes essas execuções.

A eficácia do jogo como ferramenta para auxiliar no aprendizado não foi testada, visto que não era escopo deste trabalho. Essa avaliação pode ser feita futuramente, como um trabalho separado.

REFERÊNCIAS

DEMPSEY, J.; RASMUSSEN, K.; LUCASSEN, B. **The instructional gaming literature**: implications and 99 sources. 1996. 1–63p. v.Technical.

FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. Advances in Knowledge Discovery and Data Mining. In: FAYYAD, U. M. et al. (Ed.). . Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996. p.1–34.

KAGGLE. **Kaggle**: your home for data science. Acessada em 30/04/2019, https://www.kaggle.com/.

MORAES, I. G. de; COLPANI, R. Desenvolvimento de um Serious Game com Realidade Aumentada para auxiliar no processo de ensino-aprendizagem de Matemática Básica. , [S.l.], 2016.

ROBOCODE. **Robocode Home**. Acessada em 30/04/2019, http://robocode.sourceforge.net/.

SWEIGART, A. **Making Games with Python & Pygame**. 1.ed. [S.l.: s.n.], 2012. 329–335p. n.1.

SYBERFELDT, A.; SYBERFELDT, S. A serious game for understanding artificial intelligence in production optimization. , [S.l.], 2010.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Pang-Ning Tan - Introduction to data mining**. [S.l.: s.n.], 2006. 792p.

TAROUCO, L. M. et al. Jogos educacionais. **Revista Novas Tecnologias na Educação (RENOTE)**, [S.l.], v.2, n.1, p.1–7, 2004.

ANEXOS

ANEXO A – Código fonte do simulador

O código fonte apresenta alguns trechos importantes da implementação do jogo, deixando de lado o que se refere à configuração do mesmo, e serve apenas para um melhor entendimento do trabalho por parte do leitor. Caso queira ler o código completo, poderá solicitar ao autor, através do email listado.

```
##### player/player.py #####

class Player(object):

    """
    Player is the base character in the game.
    """
    def __init__(self, pos=(10, 10), size=(10, 10)):
        """
        Creates a new "Player" instance.
        :param pos: a tuple containing the x,y coordinates of the
            top-left part of the rectangle.
        :param size: the x,y size in pixels.
        """
        self.rect = pygame.Rect(pos, size)
        self.color = RED
        self.vSpeed = 0
        self.falling = True
        self.facing = "left"
        self.screen = None
        self.name = ""
        self.controller = None
        self.shots = []
        self.cooldown = 0
        self.alive = True
        self.respawn_time = 0
        self.movement = {}
        self.random_timeout = 0
        self.score = 0
        self.state_controller = None
        self.state = None
        self.enemy = None
        self.anim_state = -3

    def move(self, move_dict, sc):
        colliders = sc.get_scenario()
        if "horizontal" in move_dict:
            if move_dict['horizontal'] == 'left':
                self.move_single_axis(-1, 0, colliders)
                self.facing = "left"
            else:
```

```
            self.move_single_axis(1, 0, colliders)
            self.facing = "right"
         self.anim_state += 0.2
         if self.anim_state > ANIM_STATES:
            self.anim_state = -3
      if 'jump' in move_dict:
         if move_dict['jump'] == 'down':
            self.jump(1)
         elif move_dict['jump'] == 'up':
            self.jump(-1)
      if "shoot" in move_dict:
         if move_dict['shoot'] == 1:
            self.shoot(colliders)


##### scenario/floor.py #####

class Floor(object):

   def __init__(self, screen=None):
      self.objects = []
      self.color = BLUE
      self.screen = screen

   def insert(self, pos, size=(GRID_ROW, GRID_COLUMN)):
      self.objects.append(pygame.Rect(pos, size))

   def insert_map(self, level_map=scenario):
      for ci, i in enumerate(level_map):
         for cj, j in enumerate(i):
            if j is '#':
               self.insert(pos=(cj*GRID_ROW, ci*GRID_COLUMN))

   def get_scenario(self):
      return self.objects

##### game.py #####

class Game(object):

   def __init__(self):
      pygame.init()
      pygame.display.set_caption("Jump'n Shoot Man")
      self.screen = pygame.display.set_mode((600, 400))
      self.run = True
      self.clock = pygame.time.Clock()
      self.players = []
      self.scenario = Floor(self.screen)
      self.saver = Save()
      self.state = {}
      self.scores = [0,0]
```

```
def save_state(self):
    if len(self.players) is not 2:
        print("Should have 2 players set. Not saving state.")
        return

    players_pos = [self.players[i].rect for i in
        range(len(self.players))]
    players_facing = [self.players[i].facing for i in
        range(len(self.players))]
    players_move = [self.players[i].movement for i in
        range(len(self.players))]
    players_score = [self.players[i].score for i in
        range(len(self.players))]
    self.state = {
        "p1_x": players_pos[0].x//GRID_COLUMN,
        "p1_y": players_pos[0].y//GRID_ROW,
        "p1_facing": players_facing[0],
        "p1_horizontal": players_move[0]["horizontal"],
        "p1_vertical": players_move[0]["jump"],
        "p1_shoot": players_move[0]["shoot"] or 0,
        "p1_score": players_score[0],
        "p2_x": players_pos[1].x//GRID_COLUMN,
        "p2_y": players_pos[1].y//GRID_ROW,
        "p2_facing": players_facing[1],
        "p2_horizontal":players_move[1]["horizontal"],
        "p2_vertical": players_move[1]["jump"],
        "p2_shoot": players_move[1]["shoot"] or 0,
        "p2_score": players_score[1],
    }
    self.saver.write_state(self.state)

def loop(self):
    loops = 0
    while self.run:
        for e in pygame.event.get():
            if e.type == pygame.QUIT:
                self.run = False
                self.saver.save_and_quit()
                return
            if e.type == pygame.KEYDOWN and e.key == pygame.K_ESCAPE:
                self.run = False
                self.saver.save_and_quit()
                return
        self.clock.tick(60)

        self.screen.fill(BLACK)
        self.scenario.draw()

        for i, player in enumerate(self.players):
            self.scores[i] = player.score
```

```
            player.update(self.scenario)
            player.set_state(self.state)
            player.draw()


         pygame.display.flip()
         loops = loops + 1
         if loops is SAVE_FRAME:
            loops = 0
            self.save_state()


##### state/stateLoader.py #####

class Controller(object):

   def __init__(self):
      self.state = []

   def load_states(self, state_file):
      f = open(state_file, 'r')
      fieldnames = ['p1_x', 'p1_y', 'p1_facing',
                'p1_horizontal', 'p1_vertical', 'p1_shoot',
                   'p1_score',
                'p2_x', 'p2_y', 'p2_facing',
                'p2_horizontal', 'p2_vertical', 'p2_shoot',
                   'p2_score']
      load_file = csv.DictReader(f, fieldnames=fieldnames)
      for i in load_file:
         self.state.append(i)

      f.close()

      try:
         int(self.state[0]['p1_x'])
      except:
         print("removing first row")
         del(self.state[0])

   def compare_state(self, s, _index):
      # These will be the comparing states. Everything else is
         command.
      comparable_states = ['p1_x', 'p1_y', 'p2_x', 'p2_y',
         'p2_facing', 'p2_shoot']
      if s == {}:
         return False
      for i in comparable_states[:4]:
         t = type(s[i])
         err = abs(s[i] - t(self.state[_index][i]))
         if(err > MAX_ERROR):
            return False
```

```
            for i in comparable_states[4:]:
                if s[i] != self.state[_index][i]:
                    return False
            return True

        def state_control(self, s):
            _eq = self.match_state(s)
            if _eq is -1:
                return self.random_action()
            print("You got a match! Not the one you wanted, though...\n")
            move = {'horizontal': self.state[_eq]['p1_horizontal'],
                    'jump': self.state[_eq]['p1_vertical'],
                    'shoot': self.state[_eq]['p1_shoot']}
            return move


#### state/stateSaver.py #####

class Save(object):

    def __init__(self, filename='output.csv'):
        exists = filename in os.listdir()
        self.f = open(filename, 'a+',
            buffering=arbitrary_saving_moment)
        self.writer = csv.DictWriter(self.f, fieldnames=fieldnames)
        if not exists:
            self.writer.writeheader()

    def write_state(self, state):
        self.writer.writerow(state)

    def save_and_quit(self):
        self.f.close()
```