CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS – CEFET MG

*CAMPUS* CONTAGEM XI

Anna Luisa Andrade da Silva

Lucas Vinicios Santos da Cruz

Victor Gabriel Ribeiro Mariano

**Relatório referente ao projeto de “TP-Batalha Naval”**

Contagem

Abril/2023

Anna Luisa Andrade da Silva - 20212003880

Lucas Vinicios do Santos Cruz - 20212018560

Victor Gabriel Ribeiro Mariano - 20212018542

**Relatório referente ao projeto de “TP-Batalha Naval”**

Relatório técnico apresentado para as disciplinas Laboratório de Linguagem e Técnicas de Programação II e Linguagem e Técnicas de Programação II do Curso de Informática do Centro Federal de Educação Tecnológica de Minas Gerais.

Professor Alisson Rodrigo dos Santos

Contagem

Abril/2023

**AGRADECIMENTOS**

A Deus...

Ao professor Alisson, com toda sua mentoria e paciência para sanar nossas dúvidas, ao colega de classe Átila Melo, que deu um imenso suporte durante todo o desenvolvimento do trabalho, ao canal do *YouTube Gamefromscratch*, com sua explicação sobre a biblioteca e a todos os outros colegas, com o apoio e ajuda.

A todos os nossos familiares que contribuíram com a conclusão desse trabalho, por meio de apoio moral e incentivos.

**RESUMO**

Este trabalho tem como objetivo apresentar, por meio da programação orientada a objetos e com a utilização da biblioteca libGDX, a criação do jogo Batalha Naval. Explorando aspectos da biblioteca no desenvolvimento de aplicações multimédia.

Por conseguinte, os alunos Anna Luisa A. Silva, Lucas Vinicios S. Cruz e Victor Gabriel R. Mariano criaram o jogo *BattleShip*.

**Palavras-chave:** LibGDX, Batalha Naval, *BattleShip*.

**SUMÁRIO**

[1. **INTRODUÇÃO** 7](#_Toc1)

[2. **DESENVOLVIMENTO** 8](#_Toc2)

[2.1. Parte Lógica 8](#_Toc3)

# **INTRODUÇÃO**

Prototipo

Nós foi proposto pelo professor Alisson Rodrigo dos Santos a criação de um jogo denominado batalha naval(ou BattleShip) utilizando a linguagem de programação Java, a biblioteca LibGDX e com o auxílio da IDE (Integrated Development Environment (Ambiente de Desenvolvimento Integrado) Visual Studio code(VsCode). O presente relatório tem como objetivo informar como foi feita a confecção do Jogo em questão e dentro disso expor nossas abordagens, nossas mudanças e nossas dificuldades. Sendo assim esse documento apresentará um relatório detalhado de como o projeto foi confeccionado e vai desde nossas primeiras abordagens até as modificações finais do projeto.

# **DESENVOLVIMENTO**

Logo após, o professor passar as instruções do trabalho, o grupo começou a estudar e debater sobre qual forma seria melhor para o desenvolvimento do trabalho. Uma grande dúvida que surgiu durante o desenvolvimento do projeto foi se separaríamos a parte lógica da parte gráfica, de começo tentamos fazer eles juntos, entretanto, depois de duas tentativas fracassadas, em que travávamos no meio da produção do código, e tinha-se que recomeçar todo o procedimento de novo, decidimos fazer eles separados um do outro.

Na primeira tentativa, não estávamos tão familiarizados com a biblioteca e não tínhamos noção de como elaborar o jogo, depois de muitas conversas, começamos aplicar o fundo e desenvolver todo o código de uma forma geral. Todavia, o programa estava muito confuso e mal desenvolvido, dando um erro em que não conseguíamos fazer a tela do jogo aparecer.

Na segunda tentativa, depois de muita ajuda do professor Alisson e pegando como referência o código que ele apresentou no site GPJECC, elaboramos uma matriz, baseada em *array*, e fomos capazes de pegar a posição do mouse e descobrimos que o motivo do erro da primeira, era um de nossos métodos, porém não conseguímos desenvolvê-lo a ponto de conseguir resolver o erro mantendo a funcionalidade daquele método. Contudo na hora de fazemos a classe Navio, não estávamos encontrando uma forma de colocá-los neste *array*, em que não houvesse fragmentação dos navios, principalmente quando fossem colocados verticalmente. Dessa vez teve um avanço maior do que na primeira vez, mas infelizmente não soubemos como resolver este problema.

Na terceira tentativa, tivemos um suporte do aluno Átila Melo que nos recomendou o canal *Gamefromscratch[1]* e também nos explicou em como desenvolver a parte lógica separada da parte gráfica. A seguir se tem a explicação da parte lógica e gráfica.

## 2.1. Parte Lógica

## A parte lógica dividimos em seis classes: *Game, Main, Player, Ship, Board, Tile.* Cada classe vai ser explicada durante todo este tópico.

## A classe *Tile*, é inspirada nos códigos do nosso colega de classe Átila Melo, estes estão disponíveis no *GitHub[2]* dele. Esta classe tem como variáveis privadas uma *boolean hit* (que será utilizada para verificar se o navio foi atingido), um objeto da classe *Ship* e um objeto da classe Point, a qual foi importada do próprio Java. Ela serve para referir a localização de um ponto, com as coordenadas x e y, mostrada em números inteiros.

## O construtor de *Tile*, tem como parâmetro um objeto de *Point*. Nele é definido todas variáveis. A função *setShip* tem como parâmetro um objeto de *Ship*, ela configura o navio na posição determinada. A função *setHit* tem como parâmetro um *boolean*, caso o ele for falso, não teve um tiro, se for verdadeiro teve o tiro. A *getShip* retorna o navio que está ocupando a casa ou retorna *null* se não houver navio. A *getCurrentState* ela retorna o estado atual da casa, devolve o número dois, caso um navio tenha sido acertado, um para caso tenha sido acertada, mas não haja um navio, zero caso não tenha sido acertada e não haja um navio. A *getPosition* retorna a posição da casa. A *isFull* retorna se a casa tem algum navio, se não tiver retorna *false* e se tiver retorna *true*.

## A classe *Board* tem como objetivo criar o tabuleiro do jogo. Ela tem como variáveis um objeto da classe *Tile*, em forma de matriz, um objeto da classe *Point*, que tem como objetivo contar posições e tem duas constantes públicas que são a largura e a altura do tabuleiro.

## O construtor da classe *Board*, tem como parâmetro um *List* da classe *Ship* — *List* é uma interface que agrupa dados, ela é como uma coleção ordenada de dados — que é a lista de navios a serem inseridos no tabuleiro, dentro dele tem a criação de uma matriz e a criação de um objeto para cada posição da matriz, dentro dele ainda tem a função *placeShips* sendo chamada. A função *getTile* tem como parâmetro um objeto de *Point*, que dá a posição x e y do tabuleiro e ela retorna uma casa na posição indicada. A *getNeighbors* tem como parâmetro um objeto de *Tile* que entrega a casa onde o navio foi posicionado e a função retorna uma lista de casas vizinhas aos navios do tabuleiro. A *placeShips*, que foi chamada no construtor, tem como parâmetro um *List* da classe *Ship* que são os navios que serão inseridos e os coloca no tabuleiro de forma aleatória. O *Set<Tile>*, que foi utilizado é porque, ele é um conjunto de valores sem informações duplicadas, é usado neste código para economia de memória, visto que é inútil saber que dois navios tem o mesmo vizinho, basta que um deles tenha este vizinho para que nenhum navio seja colocado ali. Já o *HashSet* é a classe que guarda estes valores, porém de forma desordenada, diferentemente da classe *TreeSet*, que guarda estes valores de forma ordenada, mas, é mais lenta. O *boolean isTileAvaible*, serve para olhar se a casa está disponível. O *for* que tem na classe serve para colocar os navios no tabuleiro e realiza todos os testes para verificar se é possível colocá-lo naquela posição, se o barco não está fora da tela, se dá para colocar na horizontal ou vertical, configura o barco na posição, testa se já tem um barco, adiciona a casa à lista de casas do navio, adiciona seus vizinhos, testa se a disponibilidade dos vizinhos e por fim configura que o navio foi colocado no tabuleiro.

## A classe *Ship*, define os barcos e configura todas as funções do navio. Nela tem as seguintes variáveis privadas, o inteiro *id*, que servirá para definir qual o tipo do navio, o inteiro *size*, que será o tamanho do navio, um *List* de *Tile*, um *boolean orientation,* para definir qual é a posição do navio e um *boolean isPlaced*, para verificar se o navio foi colocado no tabuleiro. Tem como constantes privadas *SIZE\_PORTA\_AVIOES, SIZE\_ENCOURACADO, SIZE\_CRUZEIRO, SIZE\_DESTROYER*.

## O construtor dessa classe tem como parâmetro o *id*, que servirá para definir qual tipo de navio será criado. Dentro dele tem a criação de um *ArrayList* para definir as casas que serão ocupadas pelo navio e um *switch case*, para definir o tamanho do navio. A função *setOrientation*, tem como parâmetro o *boolean orientation*, que se for *true* navio será na horizontal e se for *false* o navio será na vertical. A *setPlace*, configura se o navio foi ou não colocado no tabuleiro, tem como parâmetro o *boolean isPlaced*, que recebe o estado do navio. O *getOrientation*, retorna a orientação do navio, o *getSize* retorna o tamanho, o *getTiles* retorna em quais casas ele está, o a função *isPlaced* retorna se o navio foi colocado no tabuleiro. A *addTile*, adiciona o navio em uma casa e a *isSunk*, retorna se o navio afundou.

## A classe *Player* guarda a quantidade de bombas que o jogador tem, a pontuação do jogador, seu nome e verifica se ele ganhou. Ela tem como variáveis privadas o inteiro *bombCount, currentScore, maxScore,* a *String name* e o *boolean isWinner*.

## O construtor desta classe tem como parâmetro a variável *bombCount*, que define a quantidade de bombas que o jogador tem. A *addScore*, configura a pontuação do jogador e recebe como parâmetro o score que é a pontuação atual do jogador. A função *setMaxScore*, configura a pontuação máxima do jogador e recebe como parâmetro a variável *maxScore*. O *setName*, configura o nome do jogador e tem como parâmetro a *String name*. O *setState* configura o estado do jogador e recebe o *boolean isWinner* como parâmetro. O *getBombCount* retorna a quantidade de bombas que o jogador possui. O *getScore* retorna a pontuação atual do jogador. O *getMaxScore* retorna a pontuação máxima do jogador. O *getName* retorna o nome do jogador. A *isThereBomb* testa se ainda há bombas para o jogador, se for *true* ainda tem bombas, se for *false* não tem mais. O *isWinner* retorna o estado do jogador, *true* se ele venceu, *false* se ele perdeu e por fim a função *strike* diminui a quantidade de bombas que o jogador possui.

## A classe *Main* serve somente para criar os barcos e adicionar eles no tabuleiro.

**3 CONCLUSÕES E RECOMENDAÇÕES**

Neste trabalho abordamos todos os métodos pedidos na especificação. Tentamos realizar todos os tópicos abordados por um dos títulos Atari, denominado ***TrickShot***. Implementamos os movimentos das quatro (4) bolinhas – incluindo a mestre - e do joystick (que funciona com um taco), checamos as colisões em cada parede do tabuleiro juntamente com as colisões de encapaçamento. E além da imagem colocamos o texto, que consistia na pontuação e no player, e o som que se propaga durante as colisões. Explicamos como funciona o game TrickShot e apresentamos a importância do Code Blocks e da biblioteca de multimedia SDL.

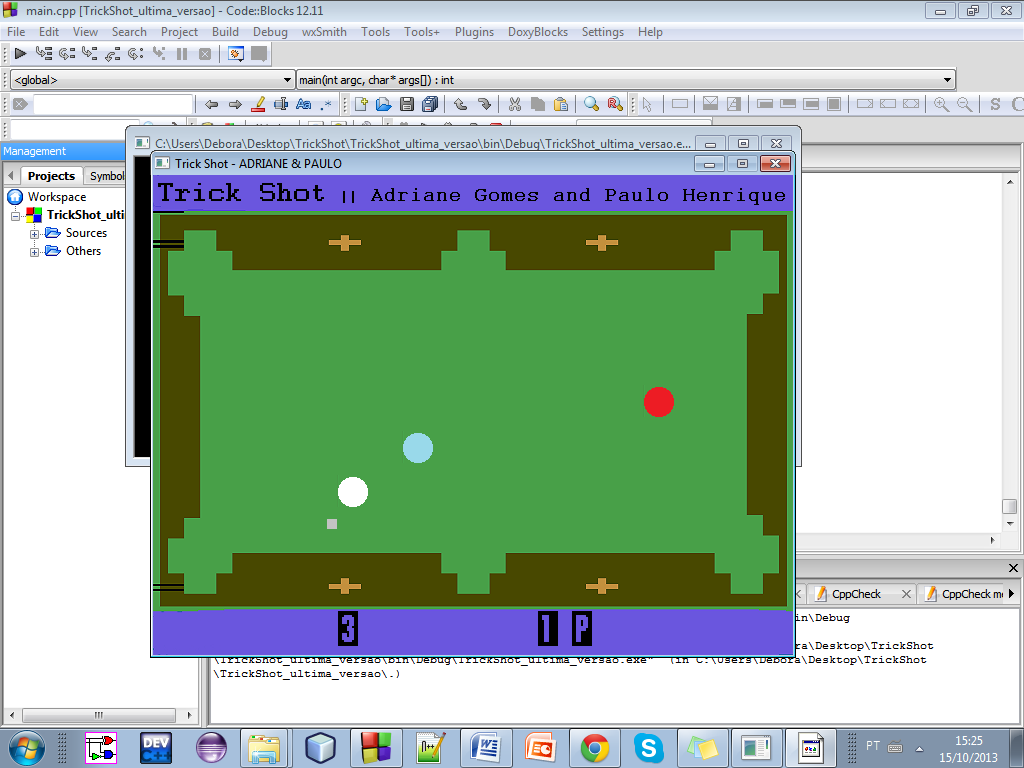
Encontramos dificuldades ao realizar o trabalho, mas com ajuda dos professores e colegas, estas foram se tornando cada vez menores. Foi um trabalho bem complexo, mas que nos ajudou a entender de forma mais clara a programação Orientada a Objetos e, principalmente, a aprofundar nossos conhecimentos com relação à Biblioteca Multimedia SDL. Acreditamos que conseguimos cumprir o objetivo e esperamos ter atendido a todas as especificações.

**APÊNDICE A – *TrickShot*** *para o Atari 2600 pela Imagic IA3000 (1982)*



**APÊNDICE B –** *Tela do Jogo em Funcionamento*

**REFERÊNCIAS**



[1] Lazy Foo’ Productions. Playing Sounds. Disponível em: <http://lazyfoo.net/SDL\_tutorials/lesson11/index.php>. Acessado em:25 de setembro de 2013.

[2] Sir MacStronger. Detecção e tratamento de colisão em um jogo 2D. Disponível em: <<http://www.sirmacstronger.eti.br/jogos2d/conceitos2.htm>>. Acessado em:4 de outubro de 2013.

[3] Background Atari. Disponível em: <http://www.atari.net.br/tela-trick-shot.htm>. Acessado em: 21 de setembro de 2013.

[4] Instalador\_Trick\_Shot.exe. Disponível em: <<http://www.4shared.com/file/X078ClJ7/Instalador_Trick_Shot.html?>>. Acessado em: 15 de outubro de 2013.

**BIBLIOGRAFIA RECOMENDADA**

Atari 2600 Instruções de Arquivo. Este documento obtido a partir da História da Home Video Games Homepage, © 1997-1998 por Greg Possibilidade.

Trick Shot. Disponível em: <http://www.giantbomb.com/trick-shot/3030-12193/>. Acesso em 21 de setembro de 2013

Trick Shot Game for Atari. Disponível em: <http://www.orgamesmic.com/trick-shot-game-atari-2600/>. Acesso em 21 de setembro de 2013.

Trick Shot Game for the Atari 2600 by Imagic. Disponível em: <http://www.atariguide.com/2/292.php>. Acesso em 20 de setembro de 2013.

Trick Shot - Atari 2600 – Imagic Disponível em: <<http://atariage.com/manual_html_page.html?SoftwareLabelID=567> >. Acesso em 20 de setembro de 2013.

Trick Shot. Disponível em: http://www.atarimania.com/game-atari-2600-vcs-trick-shot\_8051.html>. Acessado em 20 de setembro de 2013.