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 visa 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 6
2. DESENVOLVIMENTO 7
2.1. Parte Lógica 8
2.2. Parte gráfica 11
3. CONCLUSÕES E RECOMENDAÇÕES 12
4. APÊNDICE A – Compilador imprimindo as posições dos navios 13
5. REFERÊNCIAS 14
6. BIBLIOGRAFIA RECOMENDADA 15

1. INTRODUÇÃO

Nos foi proposto pelo professor Alisson Rodrigo dos Santos, para a disciplina de Linguagens e Técnicas de Programação II a criação do jogo Batalha Naval utilizando a linguagem de programação Java, em conjunto com a biblioteca de ambiente gráfico LibGDX. O presente relatório tem como finalidade apresentar as nossas ideias, técnicas, abordagens e dificuldades durante o desenvolvimento do jogo.

Os objetivos do trabalho proposto eram o primeiro contato e aprendizado básico da biblioteca LibGDX, desenvolvimento da lógica de programação no paradigma de Programação Orientada a Objetos (POO), base da linguagem de programação Java, desenvolvimento das habilidades de pesquisa, leitura de documentação de bibliotecas e classes na internet e adaptação a novas técnicas.

As especificações da construção do jogo eram: um tabuleiro 10x10, utilização de imagens e sons para melhorar a interação, os barcos seriam posicionados aleatoriamente, com as restrições de que um barco não pode encostar no outro e sobre a quantidade de barcos, o jogador deve clicar sobre a tela para jogar e o resultado deve ser sinalizado no tabuleiro, além da possibilidade de jogar novamente sem reiniciar o jogo.

2. 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.

A classe *Game* é onde todo o jogo opera, nela todas as classes criadas, entram e funcionam uma interligada a outra. Tem como variáveis privadas as seguintes, um objeto da *Board*, um *List* de *Ship*, um objeto de *Player* e os seguintes inteiros, *scoreDefaultValue*, *scoreMultiplier, temporaryScore.*

No seu construtor tem como parâmetro um *List* de *Ship* e a dificuldade do jogo, dentro dela tem um *switch case* que configura a dificuldade do jogo e define a quantidade de bomba consoante a dificuldade. O *getMultiplier* retorna o multiplicador, o *getTable* retorna o tabuleiro, o *getShips* retorna a lista de navios, o *getBombCount* retorna a quantidade de bombas do jogador. O *strike* permite que o jogador jogue um torpedo, caso ainda haja bombas para ele joga. O *increaseMultiplier* é a função que aumenta o multiplicador de pontuação. A *increaseScore* aumenta a quantidade de pontos ganhos. O *resetMultiplier*, reinicia o multiplicador de pontuação. O *resetTempScore* reinicia a pontuação temporária – a pontuação temporária permite ficar somando a pontuação e depois multiplicar pelo multiplicador, exemplo, se o jogador acertou cinco vezes seguidas na hora de multiplicar não seria dez vezes 5, mas sim a soma de toda a pontuação do jogador vezes a quantidade de vezes seguidas que ele acertou. A função *addScore*, configura a pontuação atual do jogador, recebe como parâmetro a variável inteira *scoreMultiplier*. A *hasShip*, confere se todos os navios foram afundados, retorna *true*, se todos os navios foram afundados e retorna *false* se ainda há navios no tabuleiro. O *isGameOver*, verifica se o jogo acabou, retorna *true* se ele finalizou e retorna *false* se ele ainda não acabou. A função *isWInner*, retorna *true* se o jogador ganhou e retorna *false* se ele perdeu. Por fim, a função *bombsToScore*, converte todas as bombas restantes em pontos para o jogador, ela vai ser chamada somente quando o jogo ser finalizado com a vitória do jogador.

A parte lógica foi pensada para ser interligada com a parte gráfica, se ela não funcionar, a parte lógica não consegue ser utilizada por completo.

2.2. Parte gráfica

A parte gráfica estava sendo testada pelo código do aluno Átila Melo, para sabermos como funcionava, todavia, tivemos vários problemas de tal forma que não conseguimos nem elaborar de outra forma, nem fazer da forma que estávamos planejando, pois o raciocínio que seguíamos era parecido com o do trabalho dele.

Nela decidimos fazer o tabuleiro por meio de *Table* da *libGDX*. Ele é uma variação do *TableLayout* desenvolvida pelo mesmo autor para a biblioteca *libGDX*. Tanto *Table* quanto *TableLayout* fornecem funcionalidades similares, porém, enquanto *Table* é destinado ao uso em *libGDX*, o *TableLayout* é voltado para outras ferramentas de interface do usuário, tais como *Swing* e *Android*.

Ele utiliza uma tabela lógica para determinar o tamanho e posição dos seus filhos, assim como as tabelas HTML. Para os *widgets* de layout em *scene2d.ui*, é recomendado o uso de tabelas, já que oferecem mais praticidade e poder do que a simples definição manual do tamanho e posição dos *widgets*. Uma vantagem dos layouts baseados em tabelas é que eles não dependem de posicionamento absoluto, possibilitando uma adaptação automática a diferentes resoluções de tela e tamanhos de *widgets*. Justamente por causa dessa vantagem, que decidimos utilizar ele para construir o tabuleiro. Apesar dele ser utilizado geralmente para fazer inserção de texto pelo jogador, decidimos que poderíamos elaborar ele para o tabuleiro, pois ele tem a possibilidade de ser um *bottom*, o que facilitaria na hora de pegar a posição do mouse.

Para pegar as imagens, sons, musicas, decidimos usar o *AssetManager*, que é uma das funcionalidades da biblioteca. Decidimos utilizar ele, pois, é uma maneira mais eficiente de puxar os *assets*, e ocupa menos memória do que ficar puxando textura por textura usando *sprites*. Entretanto, quando fomos testar o código com ele, o compilador não conseguia achar a pasta *assets*, tentamos resolver de outras formas, usando outras funcionalidades dele e até mesmo tentamos seguir outra lógica de raciocínio sem o uso desse recurso, mas mesmo assim não conseguimos fazer com que o compilador achasse a pasta *assets*.

Dando esse erro no código, não pudemos encontrar a solução a tempo da data limite do trabalho, fazendo com que o nosso jogo não tivesse a parte gráfica e assim não funcionasse, já que a parte lógica é completamente dependente da parte gráfica.

Desta forma, para não ficarmos sem, ter algo para mostrar, decidimos criar na parte lógica a classe *Main*, que imprimia o estado dos barcos, será mostrado no apêndice A, a impressão dos dados.

3. CONCLUSÕES E RECOMENDAÇÕES

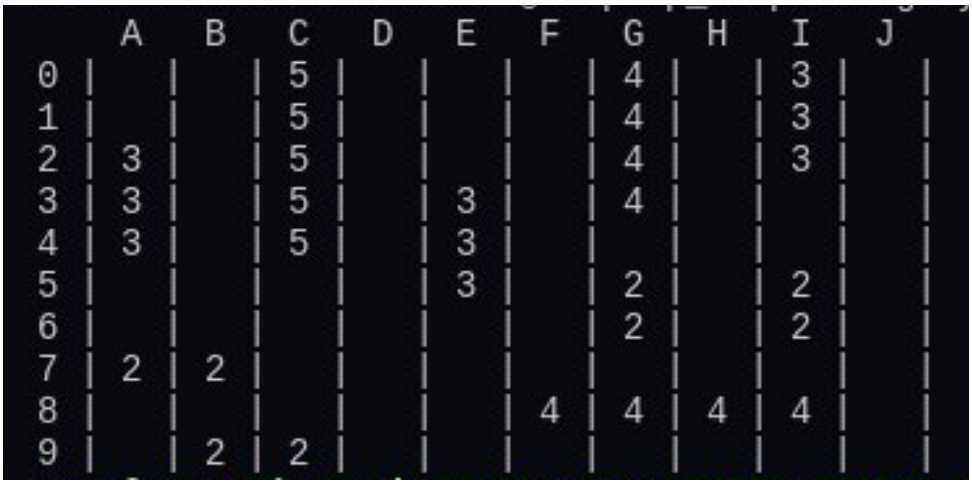
Neste trabalho abordamos todas as especificações propostas. Tentamos realizar todos os tópicos que nos foram especificados, porém, por falta de organização de tempo por parte dos alunos integrantes, além da grande dificuldade em entender a biblioteca, os objetivos não foram atingidos com sucesso.

As dificuldades encontradas ao realizar o trabalho foram muitas, desde a instalação e uso da biblioteca, até o funcionamento da mesma. Não conseguimos compreender completamente as funções da biblioteca nem as técnicas utilizadas para desenvolver um jogo, visto que nunca havíamos tido contato com este tipo de abordagem nos outros anos do curso. Mesmo a implementação da lógica nos tomou muito tempo e, mesmo que tenhamos conseguido desenvolver a parte lógica do jogo, ela foi bastante inspirada nos códigos de outro aluno (Átila Melo), que tentou nos guiar no desenvolvimento do trabalho durante a reta final até a data de entrega.

A implementação da biblioteca gráfica na lógica do jogo foi um completo fracasso. Não conseguimos, de forma alguma, executar alguma versão das três tentativas realizadas para o desenvolvimento do jogo. A parte lógica, como dito anteriormente, foi bastante inspirada no código do colega de classe Átila Melo, porém, não fomos capazes de compreender as funcionalidades da biblioteca LibGDX para podermos entregar um trabalho completo.

Sobre as especificações, foram cumpridas as seguintes especificações: criação de um tabuleiro 10x10 (parte lógica), escolha de imagens e sons para serem utilizados, um barco não é gerado por cima de outros, os tamanhos dos barcos foram atendidos, o nível de dificuldade foi implementado (parte lógica).

4. APÊNDICE A – Compilador imprimindo as posições dos navios



	A	B	C	D	E	F	G	H	I	J
0			5				4		3	
1			5				4		3	
2	3		5				4		3	
3	3		5		3		4			
4	3		5		3					
5					3		2		2	
6							2		2	
7	2	2								
8						4	4	4	4	
9		2	2							

5. REFERÊNCIAS

[1] GAMEFROMSCRATCH. **LibGDX Video Tutorial Series - YouTube**. www.youtube.com. Disponível em: <https://www.youtube.com/playlist?list=PLS9MbmO_ssyCZ9Tjfay2tOQoaOVoG59Iy>. Acesso em: 3 abr. 2023.

[2] MELO, Átila. **atilamelo/batalha_naval**. GitHub. Disponível em: <https://github.com/atilamelo/batalha_naval>. Acesso em: 3 abr. 2023.

6. BIBLIOGRAFIA RECOMENDADA

DCAMPOS. **dcampos/Batalha-Naval**. GitHub. Disponível em: <https://github.com/dcampos/Batalha-Naval>. Acesso em: 3 abr. 2023.

LIBGDX. **Wiki**. libGDX. Disponível em: <https://libgdx.com/wiki/>. Acesso em: 3 abr. 2023.

Jogo: Batalha Naval em Java. Java Progressivo. Disponível em: <https://www.javaprogressivo.net/2012/09/jogo-batalha-naval-em-java.html>. Acesso em: 3 abr. 2023.