Projeto Final: Jogo Air Monkey

Kalil S. Kaliffe[1]

[1]Instituto de Ciências Exatas e Naturais – ICEN UFPA, Belém, PA.

kalilsaldanha@gmail.com

Abstract. *This meta-paper describes how a space ship game was made with Java, in this game the player controls a monkey inside a space ship that can shoot lasers and kill enemies to collect bananas, and the objective of the game is to survive by collecting bananas before the space ship was no health due to enemy hits.*

Resumo. *Este meta-artigo descreve como um jogo de nave espacial foi feito usando Java, neste jogo o jogador controla um macaco dentro de uma nave espacial que pode atirar lasers e matar inimigos para coletar bananas, e o objetivo do jogo é sobreviver coletando bananas antes que a vida da nave acabe devido a ataques dos inimigos.*

1. Introdução

1.1. Justificativa e motivação

A justificativa parte do ponto o qual o software precisa explorar conceitos como classes, herança, polimorfismo e outros, estudados na disciplina de programação 2, os quais vão ser apresentados na secção de requisitos.

Primeiramente, foi desenvolvido um jogo digital como sistema para o projeto final, já que um jogo é um tipo de programa que aceita bem a complexidade de diversas classes e estruturas que foram requisitadas para o projeto final, uma vez o jogo por natureza vai ter que conectar diversas estruturas para funcionar de forma organizada, por exemplo é logico um personagem do jogo herdar atributos, assim como a utilização de estruturas de dados para guardas informações de posição, pontuação e objetos do jogo.

Além disso, a principal motivação para o desenvolvimento de um jogo é, que o jogo possibilita aplicar diversas boas praticas, ao mesmo tempo que é divertido, programar e jogar.

O componente da divisão nesse programa, nos permite aplicar conceitos e praticas da programação, que antes pareciam inúteis, em soluções praticas para problemas de game design ou de jogabilidade.

1.2. Objetivos

O jogo deve ser jogável no computador, através do teclado com ocasionais interações com o mouse e a perspectiva do jogo vai ser Top-Down(de cima para baixo) em 2d.

O jogador vai poder controlar uma nave espacial, que pode se movimentar e atirar, também pode interagir com os inimigos através de colisão da nave com os inimigos que reduz a vida da nave e através de colisões do ataque com os inimigos o qual derrota inimigos.

Além disso o jogador pode pontuar ao coletar bananas, que aparecem ao derrotar inimigos, bananas além de pontuar, regeneram um pouco da vida do jogador.

Quando o jogador falha, ou seja, sua vida chega a zero, o jogo acaba mostrando a pontuação do jogador em bananas coletadas e pergunta ao jogador se ele deseja jogar novamente ou sair do jogo.

2. Modelagem do software

2.1. Elicitação de requisitos funcionais

A partir dos objetivos que o jogo deve realizar, nessa secção é apresentado os requisitos funcionais do software que são esperados para cumprir os objetivos idealizados anteriormente.

Primeiramente, para refletir os objetivos do jogo, o software deve ter uma interface gráfica, e constantemente receber inputs para comandar os elementos da interface, assim foi elaborado os requisitos funcionais da Tabela 1.

Esses requisitos levam descrevem, a descrição dos Requisitos, a Prioridade e o Status, esses dados são apresentados já que na etapa de produção podem ter sido deixados em andamento, já que sua prioridade não era alta.

Tabela 1. Requisitos funcionais do software

Nº	Descrição	Prioridade	Status
F01	Disponibilizar uma nave	Alta	Feito
F02	Disponibilizar inimigos	Alta	Feito

Continua na próxima página

Tabela 1: Requisitos funcionais do software (continuação)

Nº	Descrição	Prioridade	Status
F03	Mostrar a nave e os inimigos do jogo na interface grafica	Alta	Feito
F04	Permitir o usuario controlar a nave, com WASD e Atirar com SPACE	Alta	Feito
F05	Refletir as mudanças do estado do jogo para o usuario	Alta	Feito
F06	Refletir o controle que jogador exerce na nave	Alta	Feito
F07	Permitir o jogador derrubar bananas inimigos	Alta	Feito
F08	Permitir o jogador coletar bananas	Alta	Feito
F09	Indicar a vida atual da nave	Alta	Feito
F10	Fazer a nave perder vida ao colidir com inimigos	Alta	Feito
F11	Fazer a nave recuperar vida ao coletar bananas	Alta	Feito
F12	Indicar a pontuação de bananas coletas	Alta	Feito
F13	Fazer os ataques da nave destruir inimigos	Alta	Feito
F14	Gerar diferentes inimigos	Alta	Feito
F15	Acabar o jogo quando a vida da nave chega a zero	Alta	Feito
F16	Permitir o jogador reiniciar o jogo na tela de derrota	Media	Feito
F17	Mostra desenhos graficos para a nave	Media	Feito
F18	Mostra desenhos graficos para os inimigos	Média	Feito
F19	Retornar efeitos sonoros e musica	Baixa	Em andamento

2.2. Elicitação de requisitos não-funcionais

Os requisitos não-funcionais, apresentam o que deve ser feito para implementar o jogo.

Tabela 2. Requisitos não funcionais do software

Nº	Descrição	Prioridade	Status
F01	Deve ser desenvolvido em linguagem de programação Java	Alta	Feito
F02	Deve ter uma inteface grafica	Alta	Feito
F03	Deve instanciar objetos para serem usados no jogo	Alta	Feito
F04	Deve rederizar na inferface grafica esses objetos	Alta	Feito
F05	Deve ter estados de jogo, Ex: Rodando, no Menu, no Jogo, na Tela de Derrota.	Alta	Feito
F06	Deve estabelescer um keylistener para inputs do usuario	Alta	Feito
F07	Implementa limitadores para os objetos em relação a proporção da tela	Alta	Feito
F8	A interface grafica deve ter uma proporção	Alta	Feito
F9	O programa deve coletar images de Sprites para servir estilizar o jogo	Media	Feito
F10	Implementar velicidade para moivmentar os objetos do jogo	Alta	Feito
F11	Trabalhar com sistema de coordenadas X e Y	Alta	Feito
F12	Tornar a movimentação dos inimigos aleatoria	Media	Feito
F13	Implementar 10 classes	Alta	Feito
F14	Implementar 2 métodos estáticos	Alta	Feito
F15	Implementar 2 variáveis constantes	Alta	Feito
F16	Implementar 2 métodos abstratos	Alta	Feito
F17	Implementar uma interface	Alta	Feito

Continua na proxima página

Tabela 2: Requisitos não funcionais do software(continuação)

Nº	Descrição	Prioridade	Status
F18	Implementar uma estrutura de array simples e um arraylist	Alta	Feito
F19	Implementar um tipo enum	Alta	Feito
F20	Implementar um tratamento de exceção	Alta	Feito
F21	Implementar um relacionamento entre as classes	Alta	Feito
F22	Implementar uma hierarquia de classes	Alta	Feito
F23	Implementar polimorfismo	Alta	Feito
F24	Implementar um padrão de projeto	Alta	Feito
F25	Deve usar uma biblioteca para gerar uma janela	Alta	Feito
F26	Deve usar uma biblioteca para renderizar objetos	Alta	Feito
F27	Controlar ações com bibliotecas de tempo	Alta	Feito
F28	Executar o jogo em loop	Alta	Feito
F29	Executar determinar o ritmo das ações apatir do loop do jogo	Alta	Feito
F30	Optimizar o jogo	Media	Feito
F31	Permitir que jogo rode até o usuario encerrar-lo	Alta	Feito
F32	Implementar bibliotecas de audio	Baixa	Em andamento
F33	Publicar o jogo	Alta	Em andamento
F34	Implementar troca de keybindings	Baixa	Em andamento

Continua na próxima página

2.3. Diagramas UML

A partir, da elicitação dos requisitos do software, são projetados os diagramas UML, os quais vão abstrair os requisitos para aproxima-los ao código que vai ser desenvolvido.

Assim, nessa primeira fase de desenvolvimento, foi realizada a pesquisa de oque o jogo precisava para funcionar e a partir disso foram elaboradas as primeiras classes do jogo, como Game, Player e a abstract GameObject, além disso foi possível começar a elaborar o funcionamento do jogo com classes como Enemy, ID e ActionRules.

Nota-se que na primeira fase de desenvolvimento na Figura 1, o diagrama ainda não possui todas as classes, atributos e métodos para o funcionamento do jogo, pois não é viável planejar o sistema do jogo por completo, já que alguns desafios de game design só vão surgir na segunda fase de desenvolvimento, assim foi previsto o necessário para essa fase.

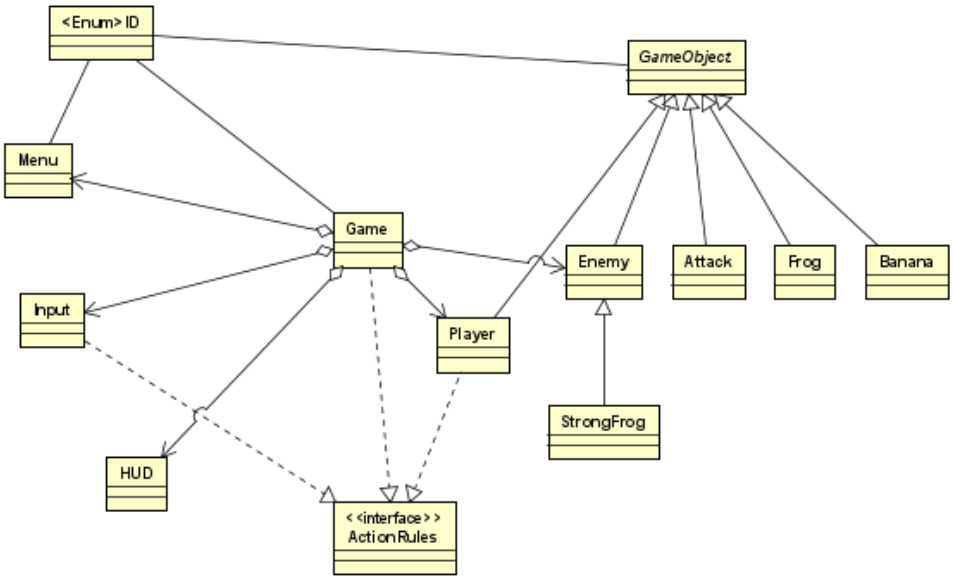

Figura 1. Diagrama de Classes, 1 fase de desenvolvimento

Quando o usuário abre o jogo ele é jogado no menu, então tem que escolher se quer iniciar o jogo ou sair, ao iniciar o jogo o jogador pode controlar a nave, atirar em inimigos e pegar bananas como representado no diagrama da Figura 2.

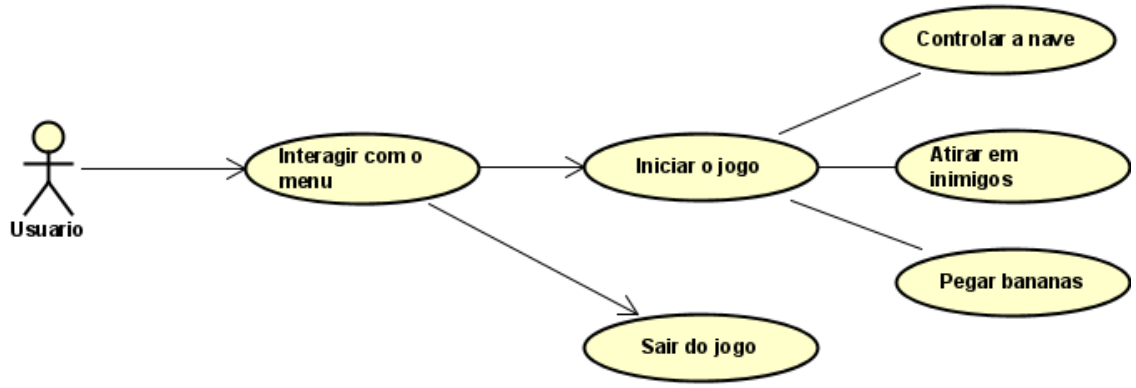

Figure 2. Diagrama de Casos de uso

Agora, na segunda fase de desenvolvimento são estabelecidos os atributos e métodos no diagrama de classes representado na Figura 3.

Lembrando que, foi necessário planejar esses elementos apenas na segunda fase de codificação, já sua implementação tem que ser testada e organiza a partir dos testes.

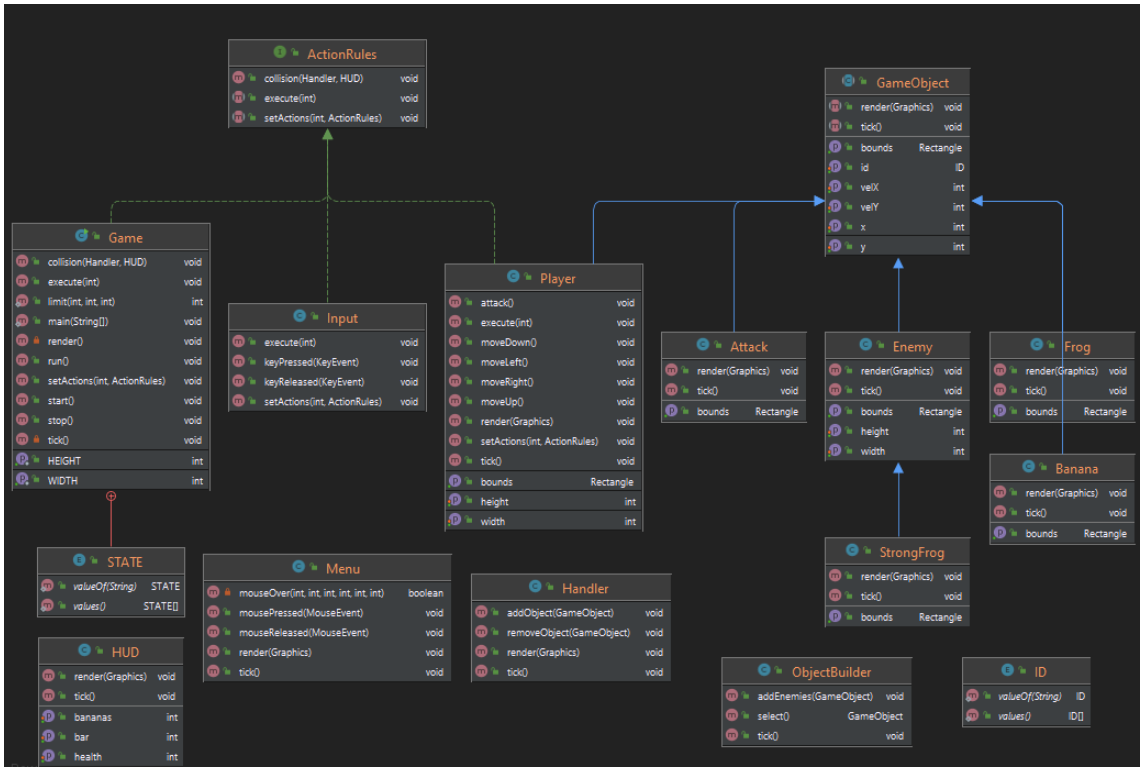

Figure 3. Diagrama de Classes, 2 Fase de desenvolvimento

A classe Game, que serve como uma Main, vai servir como central para instanciar os objetos do jogo, além de rodar o loop principal do jogo, responsável por chamar ticks() que são como o tempo interno do jogo, e por chamar Renders() que renderizam os objetos.

Um desses objetos que deve ser instanciado é um objeto da classe Handler, que contém um vector com os objetos do jogo que implementam a classe abstract GameObject e esses objetos possuem um ID um catalogado pelo Enum, esses GameObjects possuem coordenadas, velocidade, tick e render que são executados pelo loop do Game visto na Figura 4.

O Game também possui seu próprio Enum STATE para coordenar a HUD e Menu, catalogando os states atuais do jogo.

O ObjectBuilder serve para entregar inimigos ao Game.

O input pega os inputs do usuário através de um Keylistener e executam operações nas coordenas do Player.

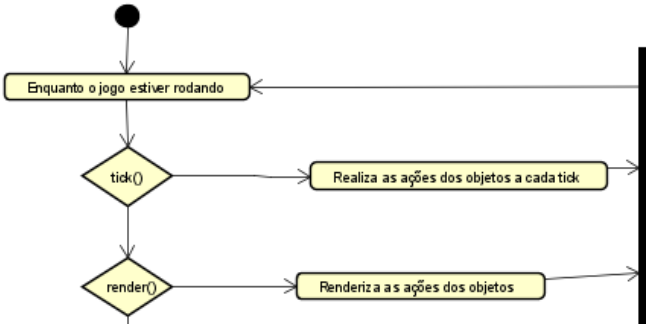

Figure 4. Diagrama de Atividades, Loop do Game

2.4. Ambiente de desenvolvimento

O ambiente de desenvolvimento para Java utilizado foi o IntelliJ IDEA[2] da JetBrains.

Essa IDE, foi utilizada para o projeto, uma que ela facilita realizar os testes do jogo. Além disso, ela ajuda a achar bugs facilmente utilizando sua função de Debug, também, como nesse projeto foram utilizadas bibliotecas, esse ambiente ajudou a otimizar os imports, importando somente o necessário para cada classe.

2.5. Tecnologias utilizadas

Além do ambiente de desenvolvimento, também foram utilizadas, tecnologias como o Astah UML[3] e a ferramenta de UML do Intellij, para construir as UMLs do projeto.

3. Testes Computacionais

Essa secção apresenta os testes computacionais do jogo, o fluxo dos testes vai seguir a rotina de um usuário jogando o jogo normalmente.

Primeiro, ao abrir o jogo o usuário é jogado no Menu Figura 5, que apresenta 3 opções ao usuário, Começar o jogo, Ser informado das regras do jogo e Sair.

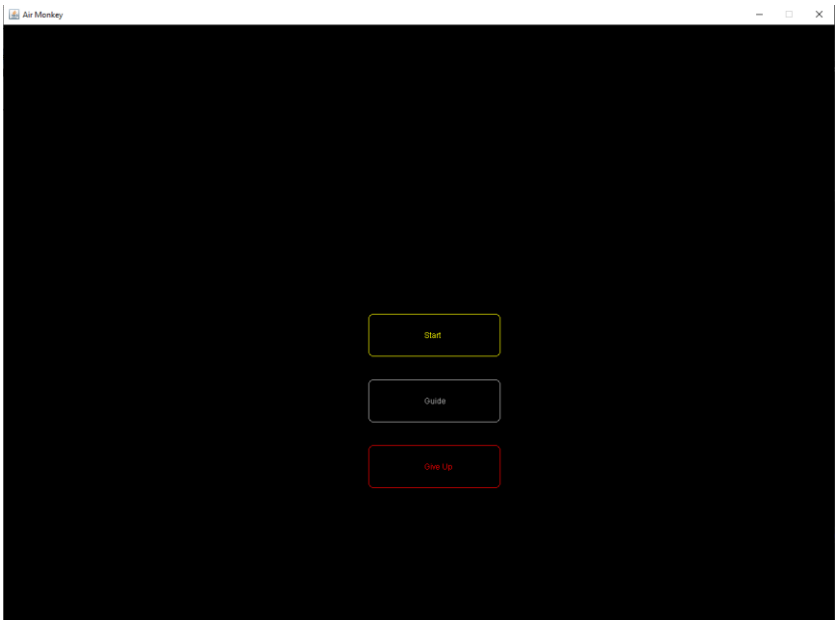

Figure 5. Menu do jogo

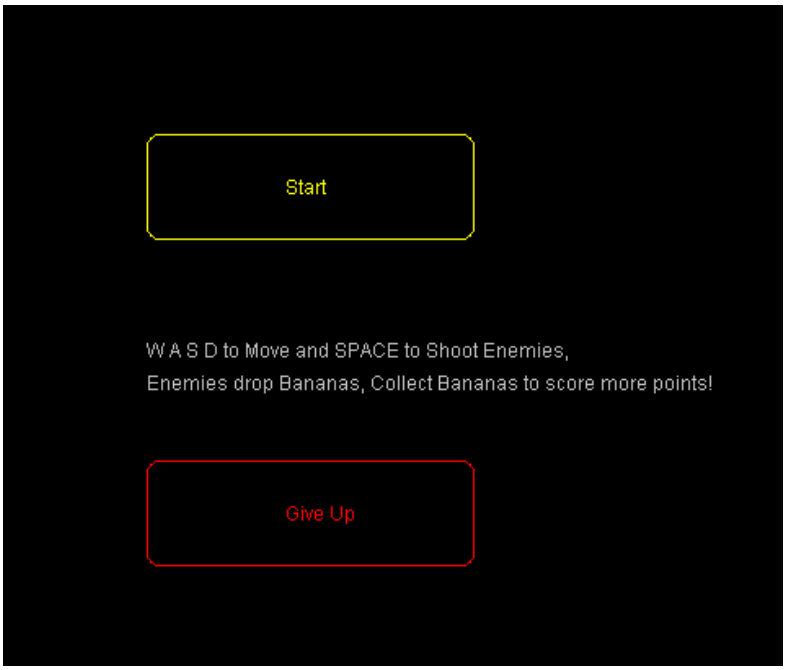

Figure 6. Menu do jogo(Guide apertado)

A Figura 6 mostra as instruções, apresentadas ao usuário quando de aperta Guide.

Quando o jogo é inicia com Start (Figura 7), aparecem a nave, a HUD com a barra de vida e contagem de bananas, também os inimigos começam a aparecer de em posições aleatórias.

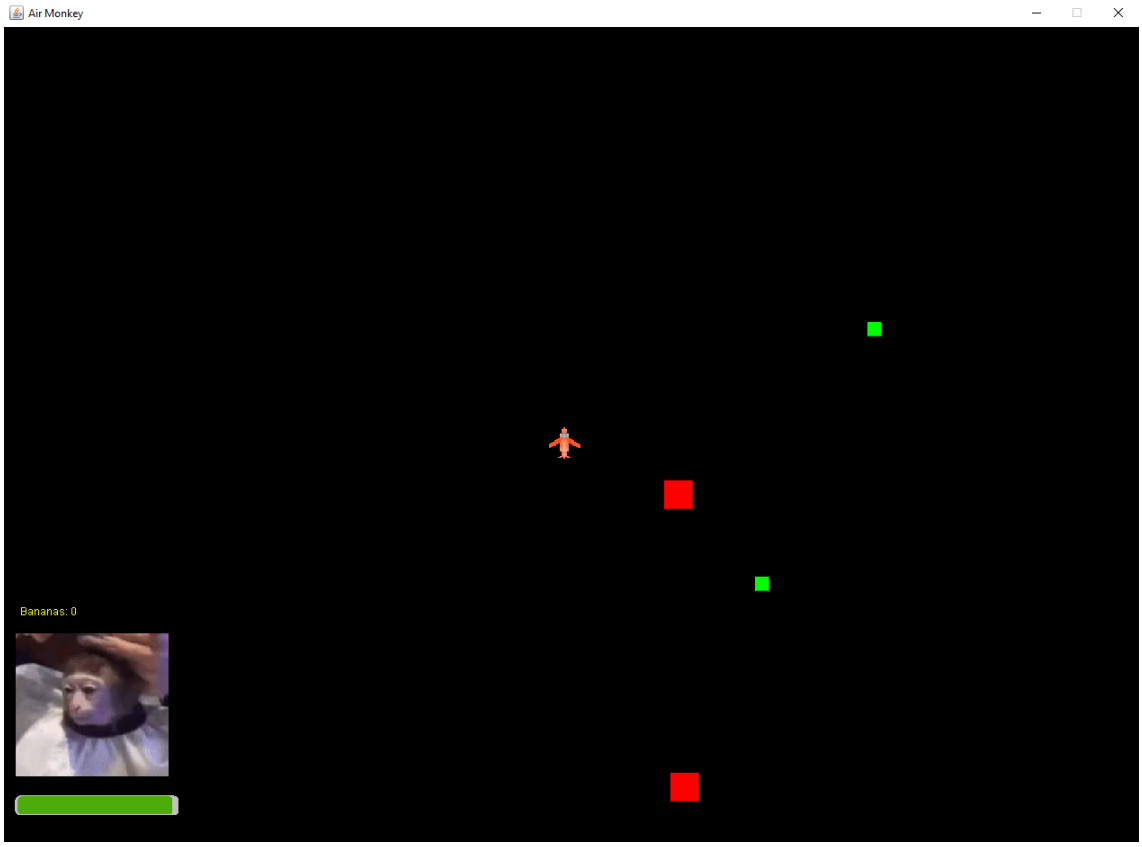

Figure 7. Jogo iniciado

Quando o jogo é iniciado, a nave pode se mover e atirar (Figura 8), quando esse laser da nave entra em contado com um inimigo ele é destruído ao mesmo tempo que deixa uma banana onde morreu.

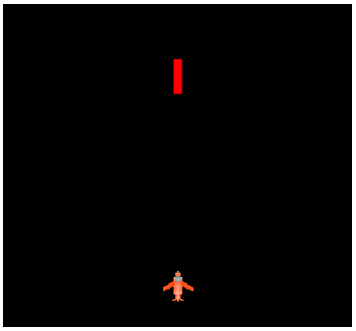

Figure 8. Nave Atirando

A Figura 9 mostra a colisão no tiro com o inimigo que deixa uma banana

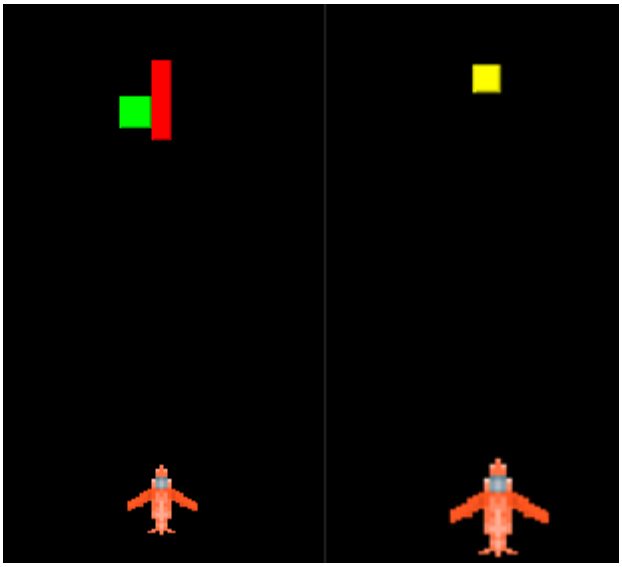

Figure 9. Inimigo Morrendo e deixando uma banana

A colisão é implementada a partir da interface ActionRules, que checa se a fronteira de um objeto interceptou a de outro, o método da fronteira getBounds() é implementado a partir do Abstract GameObject, e ação por colisão é identificada pelo ID do Objeto, logo temos outras colisões como na Figura 10, que mostra a colisão da nave com o inimigo, reduzindo a vida do Player, através da integração da HUD com os dados da colisão.

Figure 10. Macaco perdendo vida

Ao capturar bananas o player regenera um pouco de vida representado na Figura 11, (após a coleta de bananas a partir da Figura 10).

Figure 11. Display de Bananas

Durante um jogo normal (Figura 12) diversos inimigos vão aparecer e jogo diferencia esses objetos com o ID.

Os inimigos vistos na Figura 12 assim como a nave não saem na tela, eles são controlados pelo método estático limit() em Game, que para o Player trava sua coordenada quando está na borda e para o inimigos inverte sua velocidade fazendo ele rebater na parede.

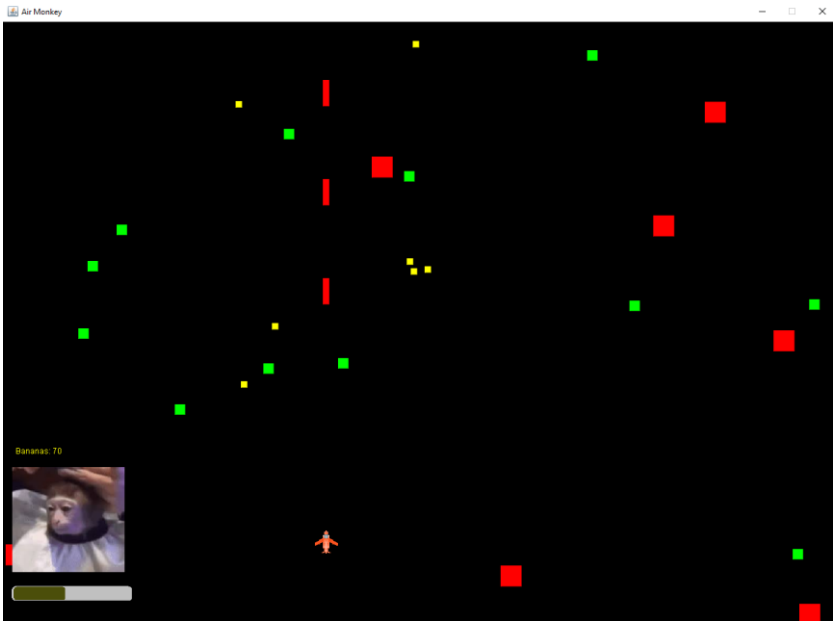

Figure 12. Jogo Sendo Jogado

Se a vida do Player chegar a zero o jogo acaba e retorna a quantidade de bananas coletas, assim como opções para, tentar novamente ou ir para o Menu ou Sair do jogo, Figura 13.

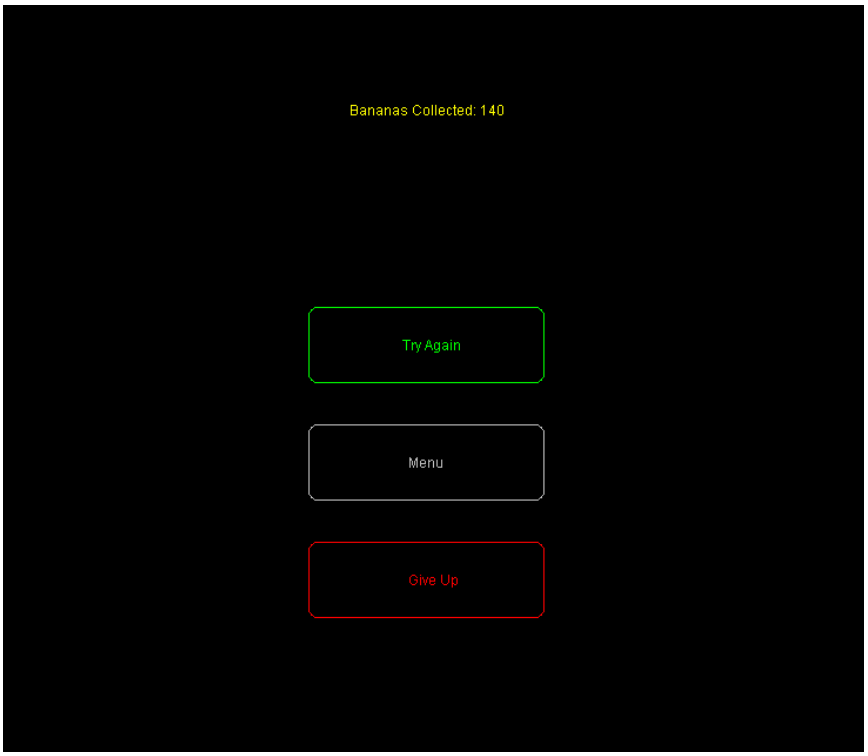

Figure 13. Jogo Encerrado

Se o jogador escolher sair do jogo o jogo simplesmente fecha se ele escolher tentar

novamente, o jogo vai instanciar um new Game() e encerrar a instancia passada.

Se o jogador conseguir sobreviver, ou seja, manter sua vida acima de zero, o jogo continua, porém jogo fica mais difícil a cada momento já que como visto na Figura 12, a quantidade de inimigos aumenta consideravelmente com o tempo, forçando o jogador a desviar e atirar.

Também, o jogo torna mais difícil o jogador só atirar e não desviar já que como pode visto na Figura 14, não podem ser disparados ataques em intervalos de tempo menores 180 milisec, impossibilitando o jogador ficar invencível protegido por seus ataques.

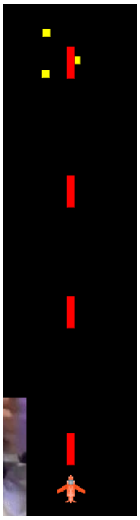

Figure 14. Disparos consecutivos

4. Precificação

Essa secção elabora e explica o preço final no sistema.

O preço final do software vai ser baseado nas tecnologias usadas, no tempo gasto e recursos e nos requisitos elicitados.

Primeiro o projeto foi feito com alguns softwares pagos como: Windows 10 Pro, Intellij IDEA[2] e Astah UML[3], assim o preço inicial do software é de R$ 25 para pagar as licenças utilizadas.

Também, vai ser somado ao preço a conta de energia, usada para alimentar o computador durante a codificação, assim como 7 dias para a codificação, logo é adicionado R$ 50 reais ao preço (preço atual: R$ 75).

Além disso, o programa cumpre os requisitos principais além de estabelecer requisitos adicionais necessários, adicionando R$ 25 ao preço atual (preço atual: R$ 100).

Dessa maneira, ainda considerando futuras atualizações e o potencial do jogo para

multiplicar diversas vezes o investimento inicial, um preço final justo para o software é de R$ 100 de gastos + R$ 100 de comissão, ou seja, um preço final de R$ 200.

5. Conclusão

A partir desse projeto, foi possível, aprender a aplicar conceitos vistos apenas em exemplos na disciplina e aplicar na pratica esse aprendizado no jogo desenvolvido, assim foi possível aprender que algumas práticas e paradigmas que podem parecem fúteis, são extremamente uteis quando aplicados em um sistema que deve funcionar como um organismo, o qual possui órgãos que interagem entre si com harmonia.

Dessa forma, foi possível, aprender a aumentar a produtividade de projeto, quando o projeto é organizado em etapas de desenvolvimento, desde a elicitação de requisitos até a construção das UMLs.

O Air Monkey pode ser acessado nos links do Apêndice.

5.1. Referências

Essa secção contém as referencias citadas nas secções acima.

Ambiente de desenvolvimento: https://www.jetbrains.com/pt-br/idea/[2]

Asta UML: https://astah.net/downloads/[3]

5.2. Apêndice

Essa secção contém os links para código fonte completo.

GitHub: https://github.com/hawkilol/Air-Monkey/tree/main/GameMain

Gist: https://gist.github.com/hawkilol/ff6ba58150feaafcdd6d9c9688f628c7