Fatal Error

Desenvolvedores:
Emanuel Juliano Morais Silva, emanueljulianoms@gmail.com
Marcos Vinícios Pacheco, vinipacheco08@hotmail.com

Introdução:

O jogo Fatal Error pode ser encontrado na Íntegra em: https://github.com/EmanuelHappy/Intregration_AI_TopDownShooter, sendo este o nosso projeto prático na disciplina de programação da Universidade Federal de Minas Gerais, criado com o auxílio do software GitKraken, uma ferramenta de visualização de versionamento git.

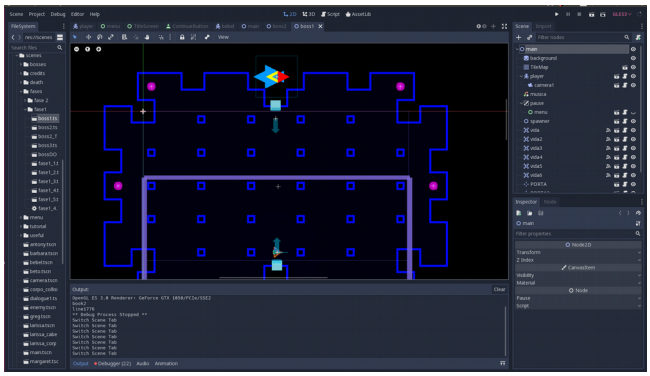
* Para a devida visualização do código, principalmente das cenas é necessário o download do Godot: https://godotengine.org/download/linux

Devido ao tamanho do código e descentralização das funções, a documentação do jogo foi dividida nas seguintes etapas:

1 – Engine e Scripts Globais
2 – Interface de Usuário
3 – Diálogos
4 – Gameplay

Engine:

O jogo utiliza a game engine Godot, uma ferramenta open source que usufrui da linguagem GDScript (linguagem integrada de alto nível, dinamicamente tipada e similar ao Python), nesse sentido, diversos objetos e funções já são pré-definidos para facilitar o desenvolvimento.
Diante disso, alguns conceitos devem ser antes compreendidos para entender de forma clara o código:



A interface do Godot é bem intuitiva, permitindo ao desenvolvedor acesso direto aos arquivos e aos nodes, assim como editores de áudio, vídeo, output dos códigos e uma tela na qual pode-se arrastar e reposicionar seus objeto.
Por tratar-se de uma Engine, a maioria dos objetos do Godot possuem representações e funções visuais.

-Nodes funcionam como objetos, nos quais os atributos de uma node pai são passados para seus filhos, o que auxilia na organização das cenas.
-Cenas são Conjuntos de Nodes, contendo sempre um Node raiz e funcionar como Node filha para outras cenas, basicamente o editor do Godot é um editor de Cenas.
-Códigos em Godot sempre terminam em .gd, sendo na maioria das vezes associados aos Nodes para executar uma ação a partir de suas funções pré definidas (tal como trocar de fase ao apertar um botão).

Esses são apenas conceitos básicos, toda a documentação do Godot, assim como tutoriais podem ser encontrados em https://godotengine.org/ ou em https://github.com/godotengine/godot.

Assim que o jogo inicia, alguns scripts são executados, como o roteiro dos diálogos (cobertos de forma mais detalhada na sessão de diálogos) e o script “game”, que controla grande parte do código e onde se encontram as funções e variáveis globais:

save_game() e load_game(): As funções de save e load usufruem de um dicionário global para armazenar informações do jogador, tais como número de mortes, nível que ele se encontra, último inimigo encontrado, tempo de vida, entre outras. Após a aquisição desses dados, sempre que o jogador morre ou retorna ao menu, abre-se um arquivo .json onde esse dicionário será armazenado para, ao abrir novamente o jogo, manter o progresso.

Continue(): Possibilita ao jogador continuar na mesma fase após morrer, assim como coordena o fluxo do jogo.

get_camera(), get_player() e get_main(): Funções de uso maior durante a gameplay, servem para sinalizar posições chaves na tela e identificar a cena atual.

Interface de Usuário:

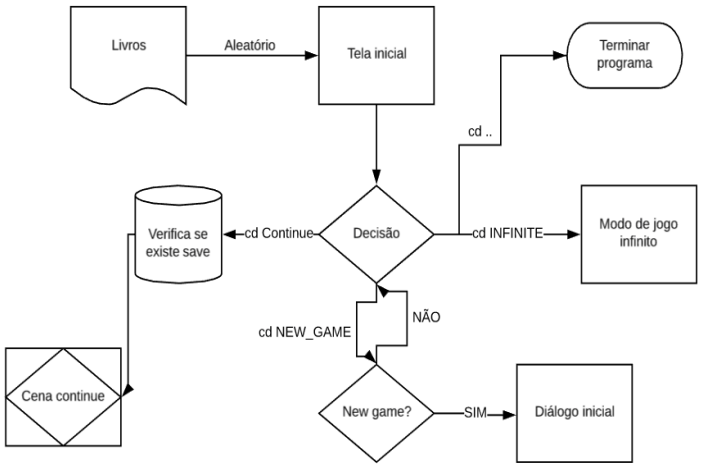
Primeira Cena do jogo, nem todos os detalhes acerca de seu funcionamento serão cobertos nessa documentação, tais como funcionamento dos botões, ou quais foram os objetos utilizados, questão que se aplica a todas as demais cenas, o objetivo dessa documentação é explicar de forma clara o funcionamento e fluxo do jogo.

A interface de usuário compõe o conjunto das seguintes pastas:

- title_screen: Contém as fontes utilizadas ao longo do jogo e a tela inicial com suas funcionalidades, tais como:
  - Gerar um texto aleatório na parte esquerda, possibilitado pelas funções random_book() e random_line() do script Story.gd;
  - Esboça o último nível jogado e quantas vezes o jogador morreu, assim como os desenvolvedores, algoritmo encontrado no _ready() de titlescreen.gd;
  - Apresenta botões que redirecionam para outras cenas, tais como Continue (não liberado enquanto o jogador não passar pelo tutorial), New_Game e Infinite, que redirecionam para suas respectivas cenas encontradas em game, suas cenas e funções são encontradas na pasta buttons.

- game: Contém as cenas de seleção de New_game (novo jogo), Infinite (modo de jogo unicamente para treinamento, sem progresso na história) e Continue (cena que permite ao jogador selecionar qualquer nível já explorado e jogá-lo novamente)

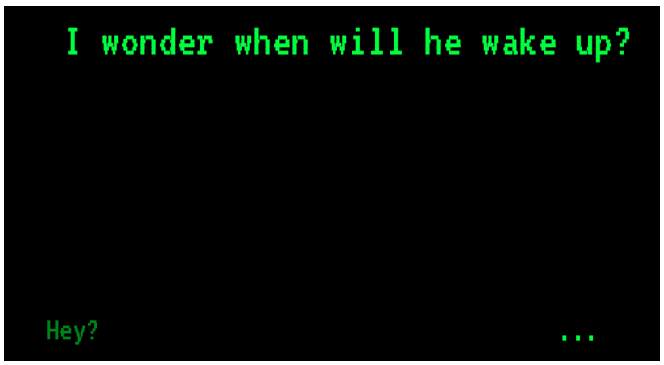
- books: Conjunto de arquivos .txt utilizados para gerar um texto aleatório na tela inicial.



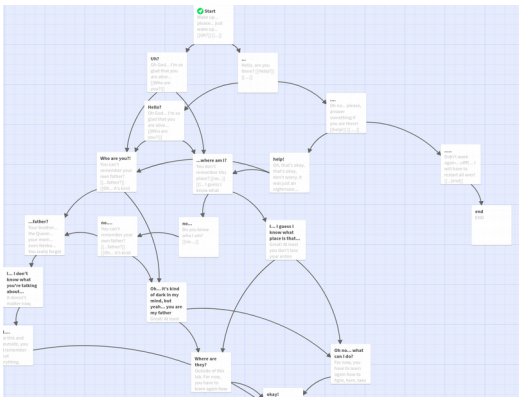
Modelo representativo da Tela inicial.

Vale ressaltar a cena death.tscn, encontrada dentro da pasta scenes, ela é de fundamental importância por, além de demonstrar todas as informações do save do jogador, também serve de link entre diálogos e gameplay, possibilitando, por meio da função global continue(), que o jogador retorne da mesma fase após morrer, ou sair do jogo caso queira.

Diálogos:



Tela padrão de diálogo



Ferramenta Twine

Os diálogos do jogo foram feitos utilizando https://twinery.org/. Twine é uma ferramenta open-source de criação de histórias interativas que, com o plugin twison, é capaz de exportar essas histórias para um arquivo json.

Após armazenarmos esses arquivos em variáveis globais iniciadas assim que o jogo inicia, utilizamos uma cena padrão de diálogo, para expressar seu conteúdo. Dessa forma, geramos os diálogos principais que serão expostos pelas cenas dialogue1.tscn, death_tutorial.tscn e level.tscn.

Diálogos menores, compostos de uma frase acerca do nível que o jogador se encontra e que aparecem sempre que ele morre são geradas de forma diferente, por meio de listas globais criamos diálogos pequenos e que se adequam a cada parte do jogo, sendo executados pelas cenas little_deaths.tscn e small_deaths.tscn.

Todas essas cenas são encontradas na pasta dialogues ou na pasta tutorial. Por se tratarem de diálogos, todas seguem o modelo padrão expresso na imagem:

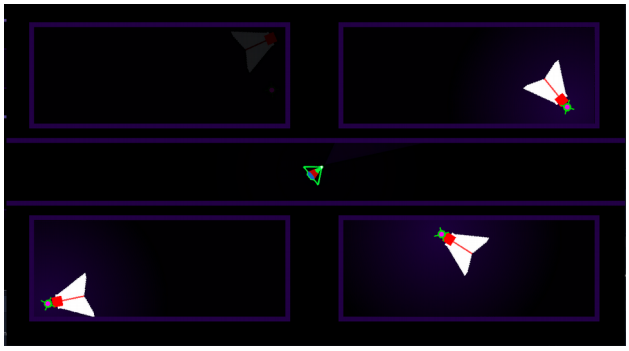
Um texto central, executando letra por letra através de um timer e que, a partir da cor da letra, fonte e áudio, é possível distinguir entre qual personagem está falando. Essa distinção ocorre por meio das chaves do json, ou por meio da função check_character(page) que checa quem está falando em cada página através de tags.

Uma ou duas opções na parte de baixo, que representam as opções de resposta do jogador. A parte interessante é que, graças a essa estratégia, é possível desenvolver múltiplos finais e distintos diálogos, permitindo uma maior dinamicidade ao jogo.

Além disso, alguns diálogos possuem efeitos sonoros, de cor e de fonte para expressar a ocorrência de eventos ou emoções.

As cenas de diálogos são sempre chamadas assim que o jogador morre, ou passa para outra fase, sendo assim, a uma intrínseca relação entre o sistema de save, principalmente com a variavel global var level e a função Continue(), assim como com a mecânica das portas do sistema de gameplay. Após o término do diálogo, o jogador pode ser redirecionado tanto para a tela de morte, quanto para o próximo nível, dependendo da cena que ele se encontra e da função check_level().

Gameplay:



O sistema de gameplay é talvez a parte mais complexa do jogo, constituído por 10 fases, intercaladas por diálogos, contendo 3 boss fights, 13 inimigos distintos e únicos que interagem entre si e com o jogador, além de cenas extras para criar desafios nas fases, efeitos sonoros, sprites e habilidades especiais.

O sistema de gameplay representa a quase totalidade das seguintes pastas:

Samples:

A pasta samples corresponde às músicas e aos efeitos sonoros do jogo, sendo, em sua maioria, sons adquiridos do site https://freesound.org/, comunidade voltada principalmente para o desenvolvimento de jogos e que disponibiliza forma gratuita e livre sons para ajudar outros desenvolvedores.
Todos os sons do jogo são adicionados como filho de algum Node, seja através do editor, ou por script, funcionando como sinalizador de eventos, mas sem possuir funções próprias devido à facilidade que as funções próprias do Godot oferecem para manipulá-los.
Vale ressaltar que o Godot já possui um editor de áudio próprio que, apesar de simples, auxilia na modificação do áudio durante a execução do jogo.

Sprites:

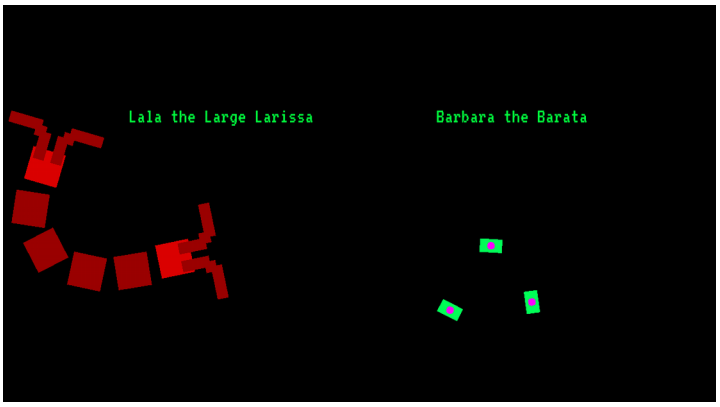
A pasta sprites corresponde à arte, contendo, por padrão, sprites próprios da engine. Contudo, para assegurar a originalidade do jogo, todos os sprites utilizados são de nossa autoria.
A adição de sprites ao jogo ocorre pelo editor, sem a necessidade de códigos próprios para isso.

Scenes:

A pasta scenes contém todas as cenas de gameplay do jogo, desde níveis a inimigos e cenas auxiliares para ajudar na criação de fases:

useful:
A pasta useful foi criada após a percepção de que utilizávamos nodes similares no desenvolvimento das fazes, sendo assim, a criação dessas cenas serviu para automatizar e agilizar esse trabalho, por meio de uma função do Godot chamada export, que permite que uma variável seja modificada pelo editor.
Os elementos desta pasta são:

- ativador.tscn: ativa um array de inimigos definido pelo editor após o jogador passar por sua área;
- trap.tscn e flecha.tscn: atira, com um delay escolhido pelo editor, flechas em linha reta que causam dano;
- lento.tscn: reduz a velocidade de quem passar por sua área, sendo essa redução definida pelo editor;
- morte.tscn: área vermelha que pode ser ativada para se mover, uma vez que o jogador encosta nele, ele morre imediatamente;
- parece.tscn: permite que o jogador atravesse sua área apenas uma vez, impedindo-o de voltar;
- TitleTest.tscb: paredes que formam o cenário, impedindo objetos com colisão de atravessarem sua área;
- PORTA.tscn: permite a transição entre fases e área, ela pode tanto modificar a posição do jogador e da câmera quanto transitar de cena, normalmente ela redireciona o jogador para uma cena de diálogo;
- shock.tscn: laser ativável de tempos em tempos (também definido pelo editor) que causa dano ao jogador;
- spawner.tscn e spawnador.tscn: geram de forma aleatória inimigos pelo mapa, a área de efeito, o delay e os tipos de inimigos também foram exportados para serem definidos pelo editor.
- vida.tscn: recupera a vida do jogador.
- câmera: pode estar acoplada ou não ao jogador, de modo a acompanhá-lo, tornar o cenário escuro, ou mudar o campo de visão.



Sprites e musicas representam um trabalho artístico garantem originalidade ao jogo, propiciando uma experiência única aos jogadores.

Bosses e inimigos:
Os bosses e os inimigos são aqueles que fornecem diversão ao jogo, possuindo cada um seu próprio código para permitir que se movimentem de forma única, além disso, cada um possui sons e sprites distintos, são eles:

- antony.tscn: Formiga que anda para direções aleatórias, porém, ao encontrar mais 4 amigos, inicia um ataque ao jogador.
- barbara.tscn: Barata inofensiva e anda em direções aleatórias. Ao morrer, fornece vida.
- bebel.tscn: Abelha que, ao avistar o jogador, anda em linha reta em sua direção, caso o acerte, seu ferrão fica preso e ela morre, causando dano, caso contrário, voa até o limite da câmera e tenta acertar o jogador mais uma vez.
- beto.tscn: Besouro que possui mais vida que os outros, mas anda de forma vagarosa até o jogador.
- greg.tscn: Grilo inofensivo, move em uma direção aleatória de forma rápida por pouco tempo, após isso, fica parado esperando para se mover de novo. Ao morrer, fornece vida.
- larissa.tscn: Lacraia que possui distintas cenas que a formam, cabeça e corpo. Ao avistar o jogador, move em sua direção, porém, caso seja iluminada pela lanterna, tenta fugir pelas sombras e alcançar o jogador por outro lado.
- margaret.tscn: Mariposa inofensiva quando não existe fonte de luz, caso exista, a persegue de forma rápida. Essa fonte pode ser tanto a lanterna do jogador, quando o vagner.
- monica.tscn: Mosca que voa ao redor do jogador formando uma órbita que reduz com o tempo.
- vagner.tscn: Vagalume inofensivo que pisca de tempos em tempos, move em direções aleatórias e, ao morrer, fornece vida.
- arnaldo.tscn: Formiga com asas, voa na direção do jogador em linha reta.
- bortolome.tscn: Rainha Formiga, Boss que cria Arnaldos e Anthonys enquanto se esconde no escuro.
- centipeta.tscn: Centopeia gigante, Boss similar a uma larissa, mas com mais cenas em seu corpo e possui distintas fases durante sua batalha.
- vanessa.tscn: Vespa, Boss que aparenta ser uma mistura de Bebel com Monica, voa em linha reta e em órbita, alternando entre ambos.

Player:
O player representa aquilo que o jogador controla, possuindo diversas cenas filhas e cenas pré-carregadas devido à sua complexidade de ações, são elas:

- mover: o jogador pode se movimentar pelo cenário por meio das teclas w a s d
- atirar: com o cursor do mouse o jogador pode mirar para onde ele quer atirar, os tiros são disparados pelo botão esquerdo do mouse, possuindo um delay entre eles.
- tiro.tscn: cena pré carregada que segue em linha reta na direção do mouse, causa dano ao atingir inimigos.
- iluminar: o jogador possui uma lanterna que, ao interagir com mapas escuros, ilumina inimigos ao redor, ela é ativada com o botão direito do mouse.
- dash: ao apertar espaço, o jogador anda rapidamente em uma direção por pouco tempo, mas possui um delay para ser usado.

Error:
Dentro da pasta death, existe a cena error.tscn, sempre que o jogador morre ela aparece, como se fosse uma transição entre a morte do jogador e o próximo diálogo.

Credits:
Os créditos funcionam como uma fase, possuindo inclusive o mesmo tipo de node usado para criar fases padrões, porém, mistura elementos das partes de diálogo com gameplay, servindo unicamente para demonstrar o trabalho dos desenvolvedores.

Menu:
A pasta menu possui uma cena de pause, com a qual, ao ser adicionada nas fases, permite que o jogador pause todo o jogo. Duas opções são apresentadas então: salvar e retornar à tela inicial ou continuar a jogar.

Scripts:
Os scripts possuem os códigos associados a cada uma das cenas descritas, com suas funções, variáveis e alguns comentários.

Fases:
A pasta fases contém 9 das 10 fases do jogo, sendo aquelas que apresentam história.

Graças à criação de cenas auxiliares, conseguimos automatizar a criação dos níveis de modo que o uso de scripts se tornou dispensável ao final.

Desse modo, o jogo se mostra extremamente escalável, possibilitando que a criação das fases se dê unicamente pela imaginação dos desenvolvedores, sem a necessidade de se preocupar com a escrita de eventos, a não ser que algo mais específico e refinado seja necessário. Tudo isso é decorrente das fortes bases desenvolvidas ao longo das primeiras semanas do projeto.