Linguagem LEVEL

Desenvolvida por João Pedro Alves Miguel

Motivação

A linguagem Level foi desenvolvida com o objetivo de proporcionar uma forma estruturada e acessível de criar jogos de RPG em ambiente de terminal. Diferente das linguagens tradicionais, que exigem o uso de frameworks gráficos complexos ou engines completas, a Level simplifica o processo de desenvolvimento, permitindo que o foco seja a narrativa e as mecânicas essenciais do jogo. Com ela, desenvolvedores podem criar mundos interativos, personagens, batalhas e eventos diretamente no terminal, de forma textual e descritiva.

Características Principais

A **Level** é uma linguagem de programação **de alto nível**, **fortemente tipada** e **interpretada em tempo real** pela **GameVM** — sua máquina virtual dedicada.

Ela foi desenhada com foco em **clareza**, **imersão narrativa** e **simplicidade estrutural**, permitindo que desenvolvedores criem mecânicas de jogo e histórias interativas sem depender de interfaces gráficas complexas.

Suas características combinam elementos de linguagens modernas com uma sintaxe inspirada em storytelling:

• Sintaxe narrativa e expressiva

Um dos pilares da Level é sua **sintaxe descritiva**, que aproxima o código de uma **linguagem natural**. Comandos como say(), attack(), move() e use(), refletem ações típicas de jogos de RPG e podem ser lidos quase como frases narrativas, tornando o código **auto explicativo** e **fácil de acompanhar**. Isso permite que o desenvolvedor foque mais na **história** e **na emoção da cena** do que em detalhes técnicos da linguagem.

• Estrutura modular e organizada

A Level oferece uma estrutura modular semelhante às linguagens modernas, com suporte a:

- **Funções (func)** — para encapsular lógicas de jogo, comportamentos de personagens e eventos narrativos;
- **Entidades (entity)** — que representam objetos de jogo, como NPCs, inimigos, itens ou o próprio jogador;
- **Blocos de controle** — como if, else, until e during, que permitem implementar lógica condicional e loops.

Cada elemento é independente e pode ser reutilizado, o que facilita a **organização do código**, a **modularidade** e o **reuso de mecânicas** em diferentes contextos.

• Tipagem explícita

A linguagem utiliza **tipagem explícita e estática**, exigindo que o tipo de cada variável seja declarado no momento de sua criação. Isso aumenta a legibilidade do código e reduz erros de execução.

Tipos disponíveis:

- **text:** armazena cadeias de caracteres (diálogos, nomes, mensagens);
- **number:** representa valores inteiros (vida, dano, ouro, tempo);
- **boolean:** valores lógicos (verdadeiro/falso), úteis em condições e controles de fluxo;
- **array:** estruturas de dados indexadas, usadas para armazenar coleções de itens, inventários, atributos e listas de entidades.

Curiosidades

A Level foi inspirada na ideia de unir narrativa e desenvolvimento na área de jogos. Sua sintaxe foi desenhada para ser intuitiva até mesmo para quem nunca programou, permitindo que escritores e designers de jogos criem histórias interativas sem precisar conhecer engines complexas. Além disso, a máquina virtual GameVM, oferece uma camada de abstração simples que transforma as instruções em ações narrativas. Por exemplo, um comando attack(enemy_hp, 20) diminui a vida de um inimigo e pode gerar mensagens automáticas no terminal.

Exemplos de Uso

A seguir, um pequeno exemplo de código em Level que demonstra uma batalha simples entre o jogador e um inimigo:

```
player_name: text = "Ragnar";
health: number = 100;
enemy_hp: number = 50;
player_damage: number = 20;
func main() {
	say("Um inimigo apareceu!");
	attack(enemy_hp, player_damage);
	if (enemy_hp <= 0) {
		say("O inimigo foi derrotado!");
		gather("potion");
		use("potion");
	} else {
		say("O inimigo ainda resiste!");
	}

	return;

}

main();
```

Exemplo mais complexo de uso:

```
player_name: text = "Ragnar";
health: number = 80;
max_health: number = 120;
player_damage: number = 30;

potion_heal: number = 50;
inventory: array = ["faca"];

enemy_orc_name: text = "Orc Guerreiro";
enemy_orc_hp: number = 60;
enemy_orc_damage: number = 25;

enemy_bat_name: text = "Morcego";
enemy_bat_hp: number = 18;
enemy_bat_damage: number = 8;

func announce_status() {
    say("----- Status -----");
    say("Player:");
    say(player_name);
    say("HP:");
    say(health);
    say("Inventário (tamanho):");
    say(len(inventory));
    if (len(inventory) > 0) {
        say("Itens:");
        i: number = 0;
        until (i < len(inventory)) {
            say(inventory[i]);
            i = i + 1;
        }
    } else {
        say("Nenhum item no inventário.");
    }

    say("Inimigos:");
    say(enemy_orc_name);
    say(enemy_orc_hp);
    say(enemy_bat_name);
    say(enemy_bat_hp);
    return;
}
```

```
func encounter_orc() {
    say("Um Orc bloqueia seu caminho!");
    say("Orc ataca primeiro!");
    health = health - enemy_orc_damage;
    if (health <= 0) {
        say("Você foi derrotado pela emboscada do Orc...");
        return; // fim se jogador morreu
    }
    say("Você sobreviveu ao golpe. Vida atual:");
    say(health);

    // Batalha: jogador ataca até orc ou jogador morrer
    until (enemy_orc_hp > 0) {
        say("Você desfere um golpe no Orc...");
        attack(enemy_orc_hp, player_damage);  // golpe grande do jogador
        if (enemy_orc_hp <= 0) {
            say("Orc derrotado!");
        } else {
            // Orc contra-ataca (somente se ainda estiver vivo)
            say("Orc contra-ataca!");
            health = health - enemy_orc_damage;
            if (health <= 0) {
                say("O Orc te derrotou...");
                return;
            }
            say("Vida após contra-ataque:");
            say(health);
        }
    }

    say("Você vasculha o corpo do Orc e encontra uma potion.");
    gather("potion");
    return;
}

func encounter_bat() {
    say("Um morcego aparece nas sombras!");
    say("Morcego tenta acertar...");
    health = health - enemy_bat_damage;
    if (health <= 0) {
        say("O morcego foi fatal... você caiu.");
        return;
    }
    say("Você é atingido levemente. Vida atual:");
    say(health);

    until (enemy_bat_hp > 0) {
```

```
        say("Você ataca o morcego!");
        attack(enemy_bat_hp, player_damage);
        if (enemy_bat_hp <= 0) {
            say("Morcego derrotado!");
        } else {
            say("Morcego bate de novo...");
            health = health - enemy_bat_damage;
            if (health <= 0) {
                say("O morcego te derrubou...");
                return;
            }
        }
    }

    // possível loot do morcego
    say("Nada além de penas e um pequeno objeto brilhante...");
    gather("small_trinket");
    return;
}

func main() {
    say("Aventura: Duas Criaturas");
    announce_status();

    // Encontro com o Orc
    encounter_orc();
    // verificar se jogador morreu
    if (health <= 0) {
        say("Cena encerrada: você foi derrotado.");
        return;
    }

    // Depois do orc, encontro com morcego
    encounter_bat();
    if (health <= 0) {
        say("Cena encerrada: você foi derrotado pelo morcego.");
        return;
    }

    say("Você venceu ambos os inimigos (ou sobreviveu à luta)!");
    say("Inventário final (itens):");
    say(len(inventory));
    announce_status();

    if (health < max_health) {
        say("Se houver potion, vamos usar para recuperar vida...");
        use("potion");
        say("Vida depois de possível uso de potion:");
```

```
        say(health);
    }

    say("Fim da cena. Parabéns, aventureiro!");
    return;
}

main();
```

Esses exemplos mostram a simplicidade da linguagem, mas também as diversas ferramentas interessantes que podem ser utilizadas, para criar uma história, e tirar do papel rascunhos de narrativas e jogos.