UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS REITOR JOÃO DAVID FERREIRA LIMA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

ENZO BASSANI
GABRIEL CALGAROTTO BERTAGLIOLI
JOÃO MAI
RAFAEL BEGNINI DE CASTILHOS

Identificação, no código produzido, da utilização dos principais conceitos e técnicas estudadas ao longo do semestre

Os principais conceitos aplicados dentro da programação orientada a objetos foram abstração, encapsulamento, herança e polimorfismo. Também foi levado em consideração os princípios SOLID e utilizado o padrão de projeto Singleton. Todos esses conceitos aplicados nos pilares e também nos postulados SOLID, possuem o propósito de evitar a replicação de código, facilitar a extensibilidade, aumentar a reusabilidade, resultando no aumento da coesão e redução do acoplamento.

Tais técnicas foram tão amplamente utilizadas que fica muito difícil imaginar como o jogo poderia ter sido implementado sem a programação orientada a objeto e seus artifícios.

A arquitetura baseada em estados também foi essencial. Acabamos não implementando-a exatamente conforme o padrão de projeto exemplificado no Refactoring Guru, dado que a utilizamos desde o protótipo de forma quase intuitiva.

No módulo de skills, utilizamos um Effect base abstrato para criação dos outros efeitos, que precisam implementar o método “apply_effect”. A partir dele, criamos os efeitos mais comuns de jogos do gênero, como Dano e Cura.

Além dele, uma segunda classe abstrata chamada LingeringEffect também foi criada, para efeitos que tem aplicação contínua por um período de tempo específico, como as suas especializações PoisonEffect e BuffEffect (mas outros efeitos que tem efeito por turno tradicionalmente também seriam de fácil implementação, como Regeneração ou Queima).

A criação de Skills com efeitos mistos também é bem simplificada, dado que os alvos são definidos nos efeitos, é possível criar Skills que curam e causam dano, curam e fornecem algum benefício, dão dano em ambos os membros do combate, diversas opções de combinações.

No Display, foram criadas classes base para elementos mais simples de exibição, como texto e botões, além de uma interface de Pressable para elementos “pressionáveis”. Os elementos com prefixo menu implementam um “on_pressed” que devolve um próximo estado, enquanto os outros implementam algum outro tipo de ação (usar item, usar skill).

Durante toda a criação do módulo de itens, herança e associação predominaram entre as técnicas utilizadas. Os itens específicos eram derivados da classe-pai abstrata Item que compartilhava alguns atributos base e o método use_item entre eles, embora o método possua diferentes funcionalidades para cada especificação de item.

Utilizamos o padrão de projetos Singleton para a implementação do jogador e outras classes que necessitavam de acesso global, e variáveis de classe para o caso do oponente. Utilizamos da *metaclass* disponível no Python, sobrescrevendo o *magic method* “__call__”.

Como foram divididas as responsabilidades na construção do jogo:

Não foram definidas responsabilidades, cada um implementou as features conforme disponibilidade e afinidade.

João Mai: Skills e Display
Gabriel Calgarotto Bertaglioli: Items, MusicPlayer.
Enzo Bassani: Jogador, Oponente, Animações e estados de combate.
Rafael Castilhos: Salas, Baú e integração dos mesmos com o Jogador.
Todos: Elaboração da lógica de estados, do diagrama de classes e interface gráfica

Principais dificuldades encontradas na construção do projeto, dentro e fora da disciplina

A principal dificuldade encontrada foi a disponibilidade de tempo dos integrantes. Muitas vezes, mesmo sabendo que um determinado código não era o melhor, implementamo-lo mesmo assim, pois planejar e implementar uma versão melhor, porém mais complexa, tomaria um tempo que não tínhamos.

Dentro da disciplina, um ponto que ficou debilitado foram os padrões de projeto, os quais debatemos e tentamos encontrar aplicação no projeto, porém em muitos casos acabou dificultando e preferimos não utilizá-los.

A implementação de menus e interfaces também é dificultosa utilizando o Pygame, dado que nativamente ele não possui ferramentas e APIs para criação de botões, scrolls, etc.

Não conseguimos implementar todos os efeitos sonoros, e variedades de inimigos, assim como a randomização das salas e geração semi-aleatório de itens e skills (que se provaram menos simples do que imaginávamos). Além disso, o Behavior proposto inicialmente para o Opponent também não pode ser implementado, onde ficamos apenas com uma versão rudimentar do que seria o RandomBehavior.

A criação de um final opcional propriamente dito para o jogo também não pode ser implementada, assim como um balanceamento apropriado para o jogo.

A função de save também não foi implementado, quando tentamos utilizar o pickle do Python, mas descobrimos que objetos pygame.Surface não eram serializáveis pela biblioteca, o que dificultou a implementação a ponto de não pensarmos em uma alternativa à tempo.

Florianópolis, 2021.