Conceitos e técnicas utilizados

Ao longo do desenvolvimento do projeto, foram utilizados diversos dos conceitos aprendidos durante o semestre da disciplina. Seguem alguns exemplos:

- **Padrão de projeto Singleton e Heranças**

```python
from abc import ABC

class Singleton(ABC):
    __instance = None

    def __new__(cls, *args):
        if cls.__instance is None:
            cls.__instance = object.__new__(cls, *args)
        return cls.__instance
```

Classe abstrata Singleton

```python
class Constantes(Singleton):
    __screenSize = (1200, 600)
    __defaultSize = (80, 80)
    __slotCounter = 0
    __charCounter = 0
```

Classe que herda da abstrata Singleton

- **Padrão de projeto MVC**

```python
class BatalhaModel:
    def __init__(self) -> None:

        self.__skillsAliados = Habilidades().skillsAliados
        self.__skillsInimigos = Habilidades().skillsInimigos

        self.__personagemSelecionado = None
```

Classe Model

```
class BatalhaView:
    def __init__(self,
                 aliados: pygame.sprite.Group,
                 inimigos: pygame.sprite.Group,
                 posicoes: list) -> None:
        self.__aliados = aliados
        self.__inimigos = inimigos
        self.__slots = [SkillSlot(i) for i in posicoes]
        self.__spritesSlots = pygame.sprite.Group(self.__slots)
        self.__habilidades = pygame.sprite.Group()
```

Classe View

```
class Controller:
    def __init__(self) -> None:
        self.__tela = Tela()
        self.__saveJogo = JogoDAO()
        self.__savePersonagens = PersonagemDAO()
        self.__batalhaModel = BatalhaModel()
```

Classe Controller

- **Interface gráfica (GUI)**
  A implementação de interface gráfica com o usuário foi feita inicialmente usando PySimpleGUI, mas posteriormente refeita usando Pygame

```
class Menu():
    def __init__(self):
        self.__botoes = [Botao('play', 600, 400)]
        self.__image = pygame.image.load(os.path.join('assets','orc.png'))
                                                    .convert()
        self.__image = pygame.transform.scale(self.__image,(1200, 600))
        self.__rect = self.__image.get_rect()
        self.__centro_largura,self.__centro_altura = (1200, 600)
        self.__rect.center =
(self.__centro_largura/2,self.__centro_altura/2)
```

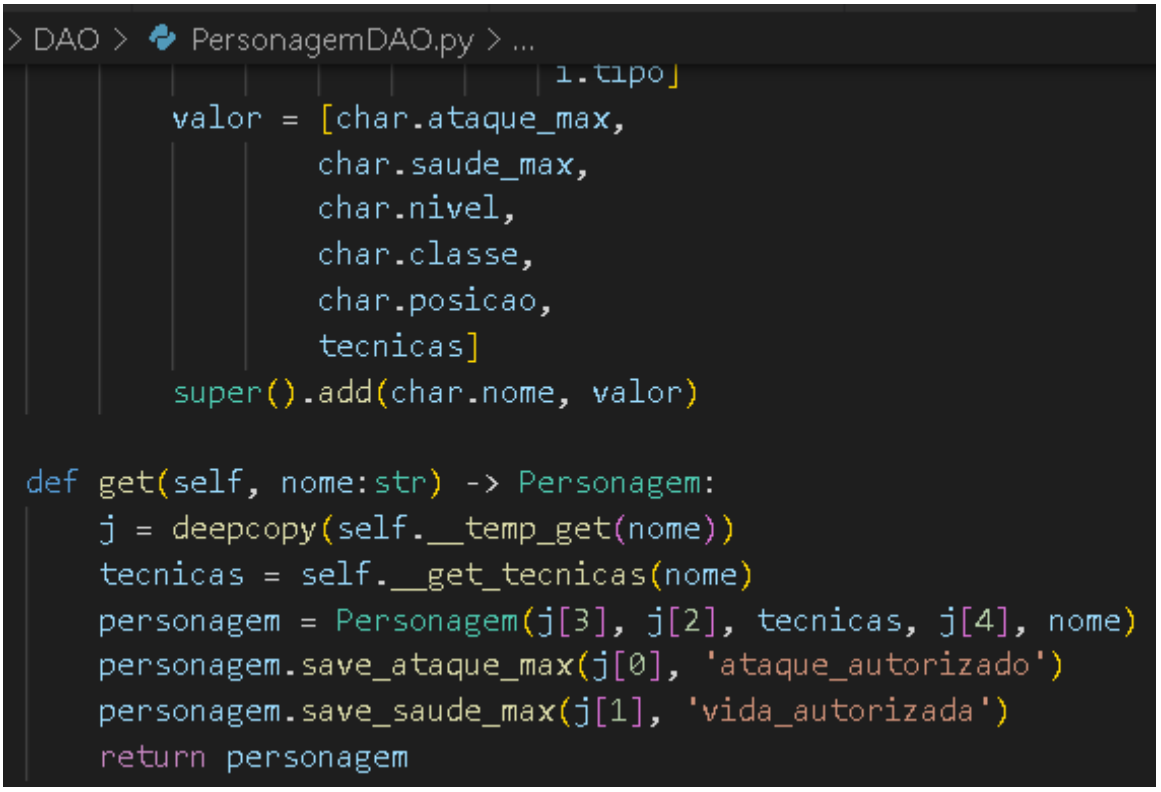
- **Persistência (DAO)**

```
class DAO(ABC):
    def __init__(self, datasource=''):
        self.__datasource = datasource
        self.__cache = {}
        try:
            self.__load()
        except FileNotFoundError:
            self.__dump()
```

Classe Abstrata DAO, herdada por classes específicas

```
from DAO.DAO import DAO

class JogoDAO(DAO):
    def __init__(self, datasource='Nivel.pkl'):
        super().__init__(datasource)
        self.add(0)

    def add(self, nivel:int):
        if isinstance(nivel, int):
            return super().add('fase', nivel)

    def get(self) -> int:
        return super().get('fase')

    def remove(self):
        return super().remove('fase')
```

Classe JogoDAO, que herda da abstrata DAO



Classe PersonagemDAO

Quanto a salvar os personagens, como pickle não permite salvar atributos do pygame, a classe PersonagemDAO salva apenas os atributos, mas a função get e get_all instanciam o personagem novamente antes de retornar

- **Construção de diagramas UML e planejamento do projeto**

  Apesar de que o grupo teve bastante dificuldades com o planejamento do projeto e acabou tendo que improvisar, isso acabou por trazer à tona a importância do planejamento do projeto prévio ao desenvolvimento, pois este se torna muito mais fácil quando existem elementos visuais para se basear durante o desenvolvimento, e também ajuda a seguir um único padrão, sem desviar bruscamente do planejado.

Separação de responsabilidades

- **Tális**

  Desde o início, ficou responsável por desenvolver a interface de batalha, uma das principais do jogo, incluindo posicionamento de personagens, animações na tela, exibição de habilidades, etc.

  Criou as classes relacionadas a Batalha e a Sprites (Sprite, Projétil, SkillSlot), para facilitar os efeitos visuais na tela.

- **Léo**
  Desenvolveu tudo relacionado ao menu e ao mapa, garantindo a fácil extensibilidade e adição de novas funções

- **Maykon**
  Desenvolveu e implementou a persistência
  Organizou a classe Singleton (que deveria ser abstrata), acrescentando a herdeira Constantes -- Que reúne os valores indispensáveis do jogo, permitindo futuras extensões mais facilmente e evitando retrabalho
  Modularizou classes que estavam no mesmo arquivo, separou god-classes, limpou o código do arquivo main, colaborando no desacoplamento, compartimentalização e separação MVC em geral.
  Desenvolveu o 'caminho' (os cenários constantes associados ao nível do Mapa) pelo qual os personagens passarão, bem associou inimigos a eles.

Principais dificuldades

Encontramos diversas dificuldades ao longo do semestre relacionadas ao desenvolvimento do projeto, mas a principal com certeza foi com relação ao planejamento prévio do projeto. Apesar de termos criado um diagrama UML no início do desenvolvimento, este acabou sendo "descartado" pois era necessário fazer diversas alterações para que condizesse com o que estava sendo desenvolvido, além de que o prazo de entrega do MVP acabou por apressar um pouco o desenvolvimento, o que foi se tornando uma bola de neve, e nos vimos nos últimos dias do prazo sem um diagrama do projeto e com diversos problemas para resolver.
A utilização do pygame também foi consideravelmente desafiadora, e tivemos certa dificuldade em dividir as classes de maneira correta para seguir os padrões SOLID, que apesar de serem claros, requerem prática para serem aplicados frequentemente de maneira correta.