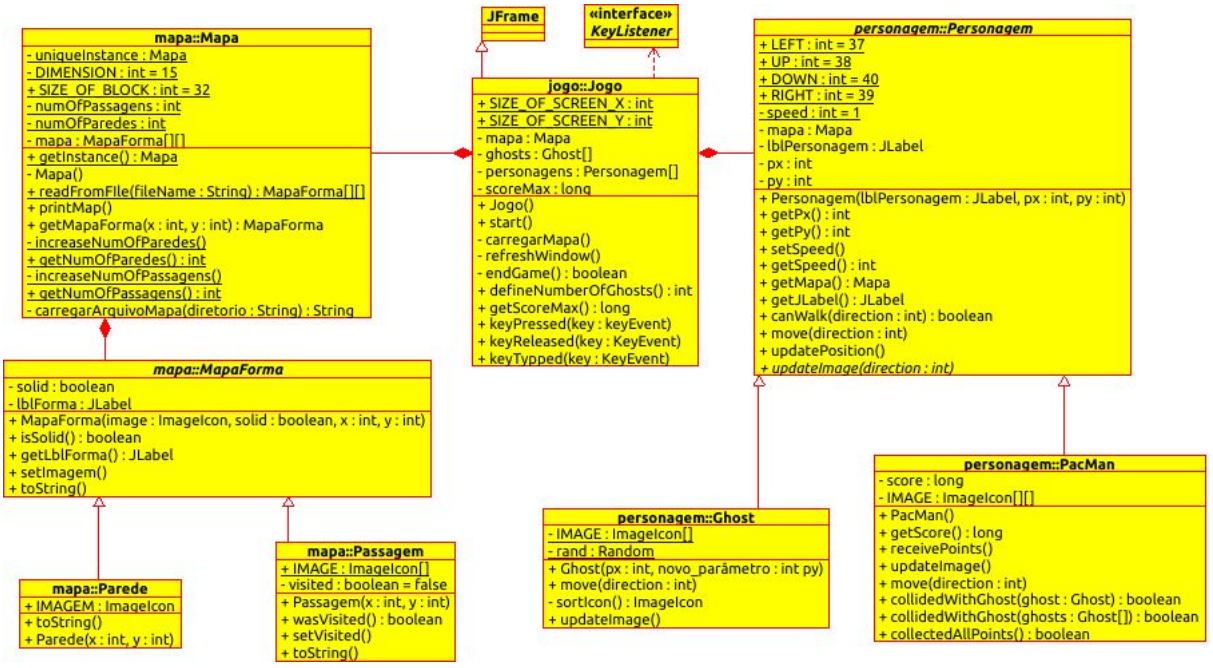
Nome: Carlos Alberto Maniusis Junior RA: 11081816
Nome: Victor Lourenço Borges RA: 11013216

Introdução

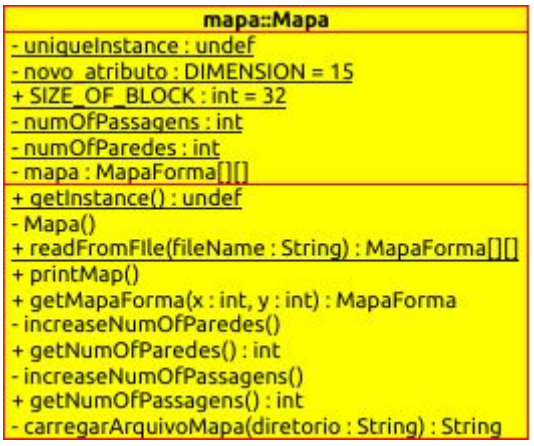
Este projeto é inspirado no jogo Pac-Man, desenvolvido pela Namco. Escolhemos este tema para que pudéssemos aplicar e reforçar os conceitos de programação orientada a objetos aprendido ao longo do curso, além de adquirir novos conhecimentos, como a utilização de uma interface gráfica.

Descrição das Classes

Diagrama de Classes

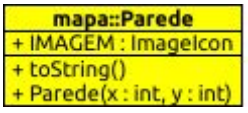


Classe **Mapa**: essa classe é um Singleton, ou seja, existe apenas uma instância dessa classe em todo o projeto, ela armazena uma matriz de dimensão *DIMENSION* do tipo MapaForma que representa o mapa do jogo, além disso, a classe também possui alguns métodos para a leitura do mapa a partir de um arquivo e métodos para o gerenciamento dos dados que foram lidos a partir de um arquivo.

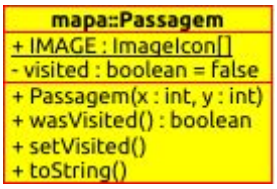


Classe **MapaForma**: É uma classe abstrata, ou seja, não pode ser instanciada, serve para representar um tipo genérico de elemento de mapa, possui propriedades comuns a todos os blocos que compõem o mapa do jogo, como por exemplo, o ícone que aparecerá na tela do jogo e sua permeabilidade.

Classe **Parede**: É um caso particular de um elemento de mapa, portanto, herda todas as propriedades comuns aos elementos de mapa (MapaForma) mas é sólido (ou seja, não pode ser atravessado) e tem um ícone diferente dos outros elementos de mapa que não são paredes.

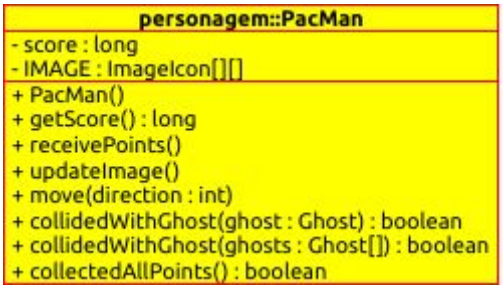


Classe **Passagem:** Assim como a classe **Parede**, é um caso particular de um elemento de mapa e, portanto, herda todas as características de MapaForma, mas possui um ícone diferente dos elementos de mapa que não são paredes, além disso, é possível passar pela Passagem, então ela não é sólida e possui um atributo que diz se ela já foi visitada ou não.

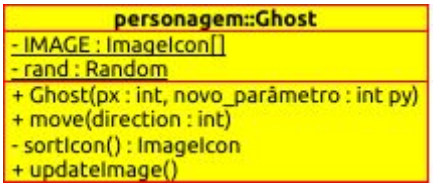


Classe **Personagem**: É uma classe abstrata que agrupa todas as características comuns aos personagens do jogo, como por exemplo, a posição, o ícone que será apresentado na tela, e a sua velocidade de movimento. Há também métodos que são comuns a todos os personagens, como por exemplo movimentar-se e verificar se podem se movimentar.

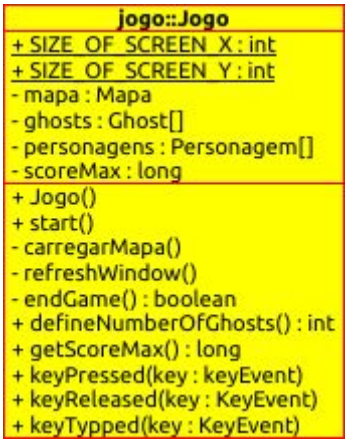
Classe **PacMan**: É um personagem, portanto, herda todas as características que um personagem tem, além disso, possui métodos para ver quantos pontos ele já conseguiu obter, se está numa situação de colisão com algum inimigo.



Classe **Ghost**: Assim como **PacMan**, possui todas as características de um **Personagem**, contudo, possui propriedades particulares, como por exemplo, seu ícone e sua forma de se movimentar.



Classe **Jogo:** Tem como o papel fundamental fazer o controle do jogo, é uma classe que extende **JFrame** e implementa **KeyListener**, possui um vetor de personagens e cada vez que uma tecla é pressionada, ele realiza o movimento dos personagens na tela, e verifica se o jogo deve acabar ou continuar.



Conceitos de Programação Orientada a Objetos Utilizados

Foram utilizados **construtores**, para garantir que as classes fossem instanciadas com os parâmetros necessários. **Gets**, **sets**, e **encapsulamento** foram utilizados para manter a coesão do programa. Foi também implementado o conceito de **herança** nos pacotes mapa, personagem e jogo. Em jogo, a classe Jogo estende 'JFrame', e implementa a **interface** 'KeyListener'. No pacote mapa, há a superclasse abstrata 'MapaForma', que dá características aos objetos (parede e passagem) do mapa, como imagem e permeabilidade. No pacote personagem, as classes 'PacMan' e 'Ghost' herdam atributos e métodos (como posição, velocidade, movimento) da superclasse abstrata 'Personagem'. O conceito de herança foi utilizado para aproveitar as características comuns entre as classes, não se fazendo necessário a cópia excessiva de código. As superclasses foram criadas como **abstratas** pois apenas passam características para as subclasses, não podendo ser instanciadas. Junto à herança, foi usado o conceito de **polimorfismo**, onde foi necessário a sobrescrita de alguns métodos, de acordo com a especificidade da subclasse. Na classe 'Jogo', por exemplo, foi criado um vetor do tipo 'Personagem', que irá conter objetos tanto do tipo 'Ghost' como do 'PacMan'. O método virtual 'move' irá ser chamado para todos os objetos do vetor, mas sua implementação será diferente, de acordo com a instância. Há também uma relação de **composição** entre algumas classes: as classes 'Mapa' e 'Personagem' dependem da classe 'Jogo' para existir, assim como a classe 'MapaForma' depende de 'Mapa'. Foi também utilizado o padrão de projeto **Singleton** na classe 'Mapa', para garantir que seja instanciada apenas uma vez, e, assim, as classes trabalhem com o mesmo mapa.

Participação de Cada Integrante do Grupo

Nos reunimos inicialmente para aprender mais sobre a interface gráfica e como realizar uma interação humano-computador a partir do teclado e também, para fazer uma modelagem

inicial do jogo, pensando sempre em como fazer de uma maneira que maximize a reutilização de código. Victor ficou responsável por fazer uma versão inicial do programa, que era funcional, mas que ainda necessitava de um aprimoramento com o uso de conceitos de POO para uma maior coesão.Também ficou responsável pela questão 3 no relatório. Carlos implementou o método responsável por ler o mapa no jogo, e aprimorou a versão inicial do programa, melhorando a coesão e a estrutura do projeto. Também realizou a questão 2 no relatório, e a implementação da UML.