Projeto MC322 - Legends Of Runeterra

Gabriel Vieira Sousa 182891
Giovani Lorenzo Bianchini dos Santos 186397
Pedro Henrique Suguino 187026

Java Boys

Funcionalidades adicionais:

1. Interface gráfica para criação de Deck e seleção de jogadores.
2. 2 novas regiões com 15 cartas cada, e 4 cartas nova para Demacia.
3. Combates que não terminam a rodada quando feitos.
4. Novos efeitos e traços para as novas cartas.

Alterações feitas em cada funcionalidade:

1. Implementado conforme especificado.
2. As 4 cartas aleatórias são as 4 cartas do topo do deck após ele ser embaralhado. Ele é reembaralhado depois das cartas serem trocadas.
3. Implementado conforme especificado.
4. A rodada não acaba até que ambos os jogadores passem seu turno.
5. Implementado conforme especificado.
6. Fizemos com que a diferença de mana entre as cartas fosse no mínimo 0, para que não seja possível ganhar mana trocando seguidores, e que só aconteça a troca se o jogador tiver 6 ou mais seguidores.
7. Implementado conforme especificado.
8. Implementado conforme especificado.
9. O combate não faz com que a rodada acabe. Ela só acaba conforme consta em 4.
10. Implementado conforme especificado.
11. Decidimos não implementar o uso de feitiços durante o combate, já que seria necessário criar reações de feitiço, ou apenas um jogador poderia usar feitiços no combate. Em outras palavras, todos nossos feitiços são lentos (Em LOR, feitiços podem ser lentos, rápidos ou súbitos, e desses apenas lentos não são usados em combate).
12. Implementado conforme especificado.
13. Adicionamos outros traços, conforme visto no UML e explicado no código.
14. Adicionamos outros efeitos, conforme visto no UML e explicado no código. Vale notar que efeitos de ataque ocorrem antes dos atacantes serem bloqueados, e que colocar uma carta na mão foi interpretado como pegar ela do seu deck.
15. Decidimos que a IA não só escolhe aleatoriamente a carta, como também a ação que vai tomar (jogar uma carta, atacar, ou passar o turno.)

16. Tanto a listagem das ações possíveis quanto da situação da mesa é feita pelo terminal, e ao invés de informar para o jogador quais cartas podem ser jogadas, o jogador é permitido de tentar jogar qualquer carta, mas se ele não tiver mana o suficiente para jogá-la a ação é invalidada e ele deve tentar novamente até passar o turno ou jogar uma carta válida, conforme o jogo original.
17. Criamos, utilizando uma interface gráfica interativa, um sistema de criação de deck.
18. Mudamos duas das cartas do deck base:
    a. O Garen não tem mais efeito, e agora tem o traço regeneração., que faz o que seu efeito fazia. Ele evolui quando golpeia duas vezes ao invés de atacar duas vezes.
    b. Como não há feitiços rápidos (Usados durante o combate), Julgamento foi alterado para que um aliado golpeie todos os inimigos ao invés de um aliado atacante golpear todos os inimigos defensores.

Detalhamento das classes:

Nossas classes estão divididas em 4 pacotes principais:

- card: Todas as classes relacionadas à implementação das cartas.
- game: Tudo que existe durante o jogo e é necessário para ele rodar.
- menu: Onde instanciamos as cartas e onde o usuário escolhe seu deck, seu oponente ou cria outro deck.
- runner: pacote do Runner, que inicia o JavaFX e o código do jogo.

Tentamos criar um pacote para todas as classes do JavaFX, mas não conseguimos fazer elas funcionarem a não ser que estivessem diretamente dentro do src. Eventualmente desistimos e mudamos a atenção para partes mais importantes do código.

Dentro de card, temos a classe abstrata Card e suas subclasses Follower, Spell e Champion (subclasse de Follower). Como seguidores e feitiços são jogados de forma diferente, cada subclasse implementa o método abstrato playCard.

Cada campeão tem sua própria classe para permitir maior flexibilidade e um código mais complexo na implementação do campeão.

Ainda no mesmo pacote, temos 3 enumerates: Region, com as 3 regiões utilizadas (Demacia, Noxus e Freljord), Trait, com todos os traços, e Trigger, que tem todas as situações onde um efeito pode ser ativado.

Por fim, temos a classe Effect, que tem um Trigger e um caso. Se o Trigger for ativado, o caso do efeito será procurado em um switch com todos os efeitos possíveis e realizado. Tanto o Trigger quanto o caso são definidos na instanciação.

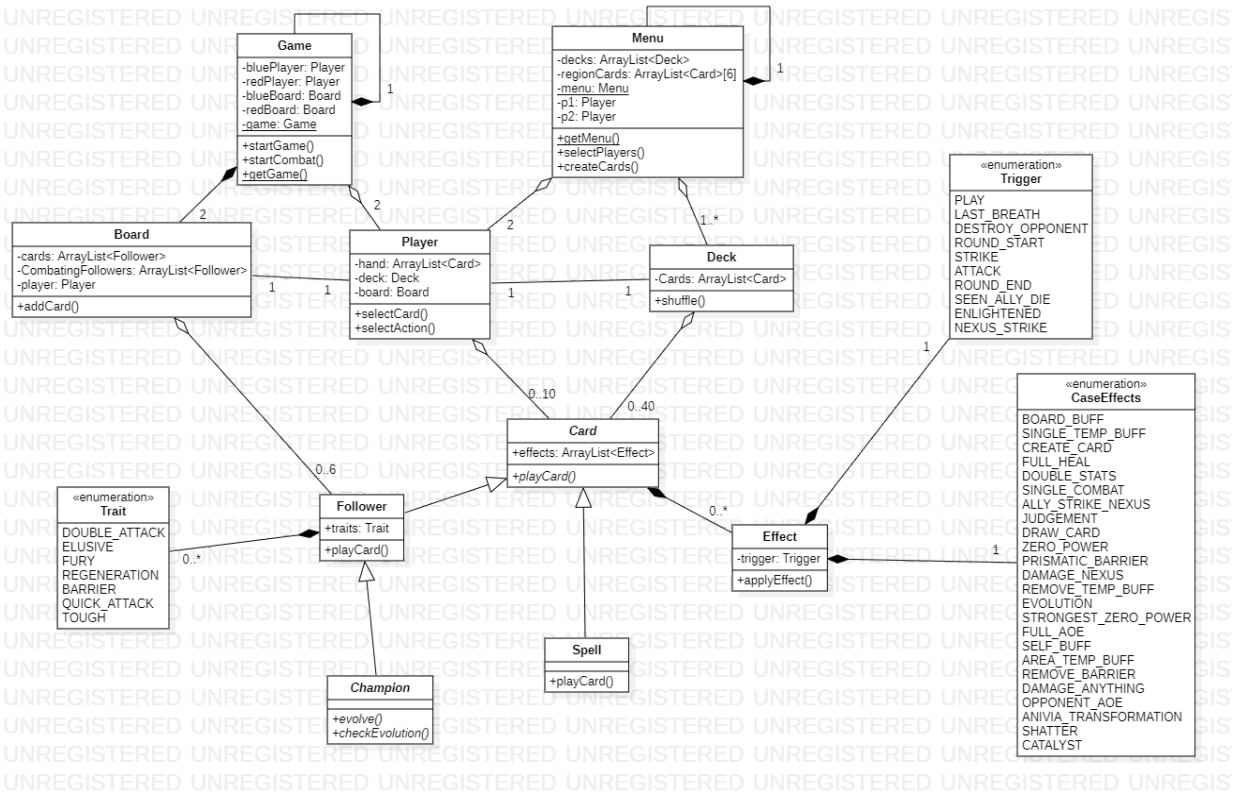
Para o pacote game, temos Player, Board e Deck. Player é composto por Deck e tem uma associação com Board. Deck guarda as cartas e as embaralha, Board guarda os seguidores em jogo, em combate e o turno, e Player guarda a vida, a mana e seleciona cartas e seguidores para lutar.

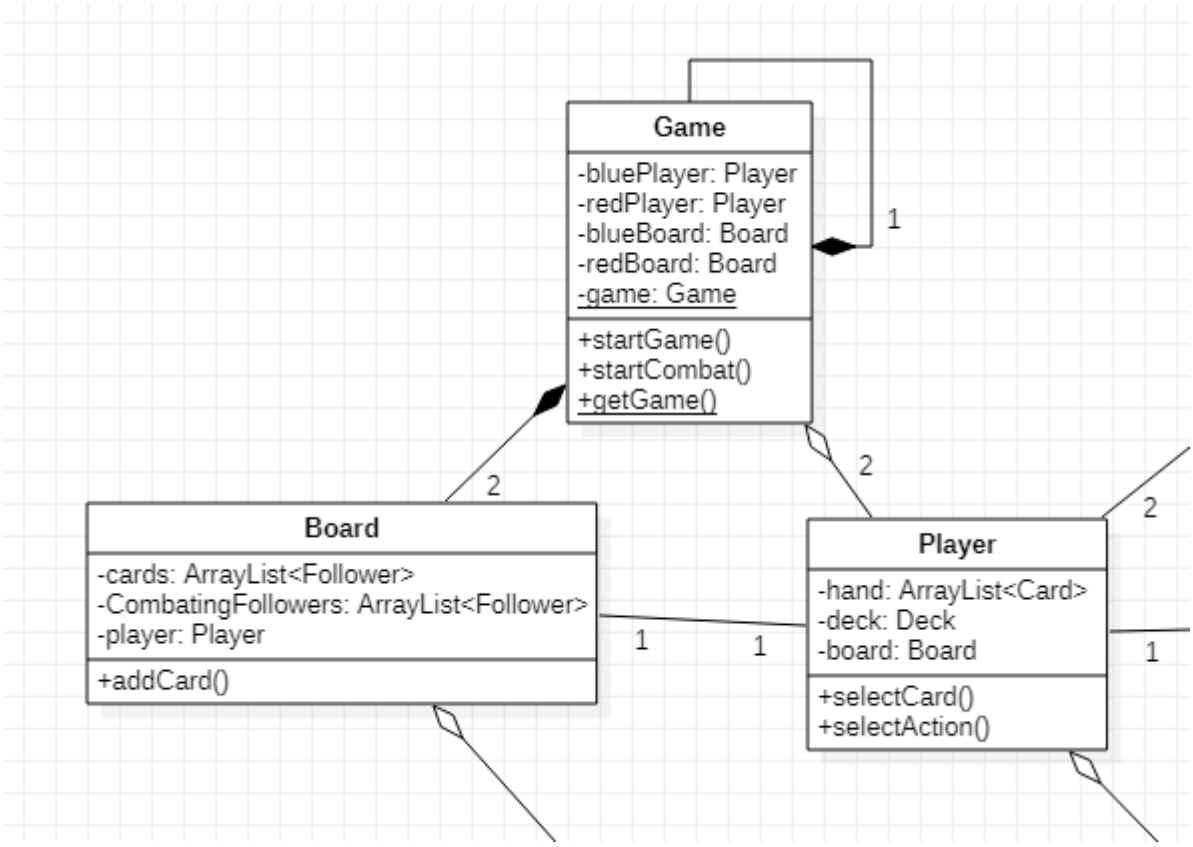
No pacote menu, temos a classe Menu, onde acontece a instanciação, a seleção de player e a criação dos arrays de cartas de cada região(note que Menu é um singleton, para mais fácil acesso dele ao redor do código), e o enumerate TypePlayer, com as 3 combinações de jogadores: PVP, PVIA e IAVIA (P sendo player e IA sendo inteligência artificial).

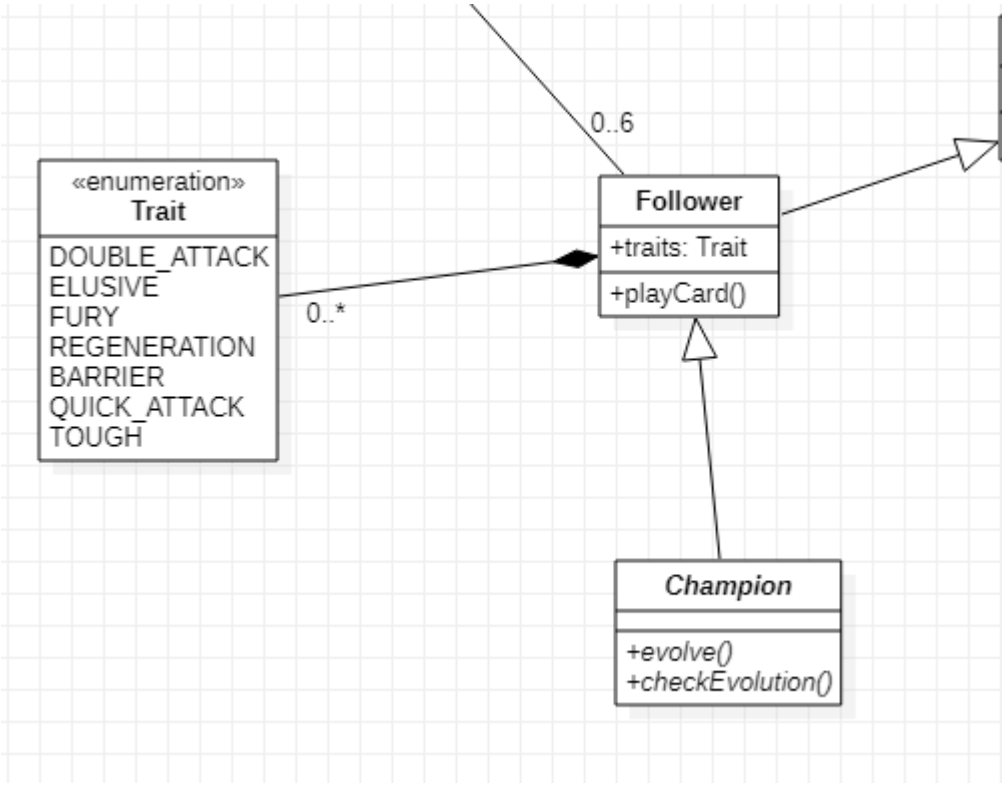
O último pacote runner possui só a Runner, onde iniciamos a interface gráfica pelo JavaFX que posteriormente chama as outras classes.

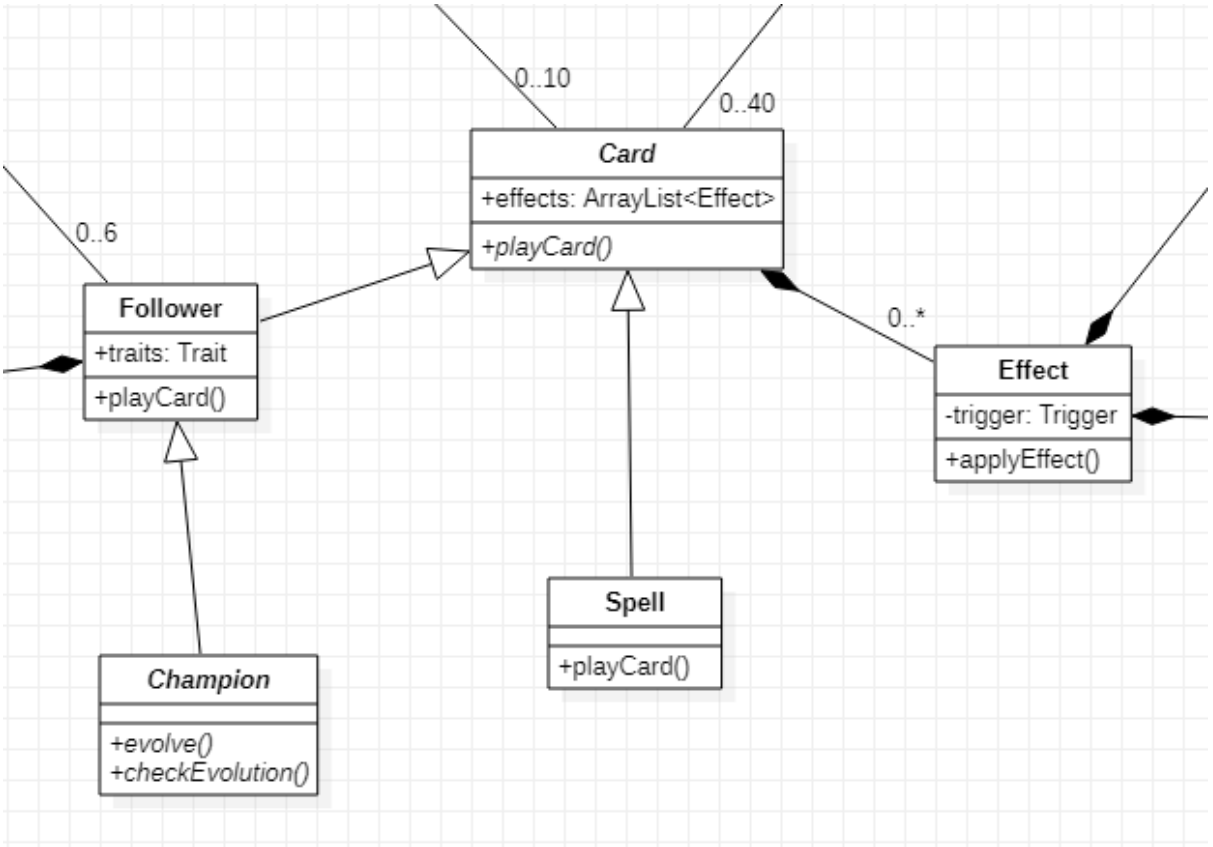
Conforme dito anteriormente, não conseguimos colocar as classes do JavaFX em um pacote pois elas devem estar diretamente na raiz do projeto. Todas elas são controllers que implementam a interface Initializable do JavaFX.

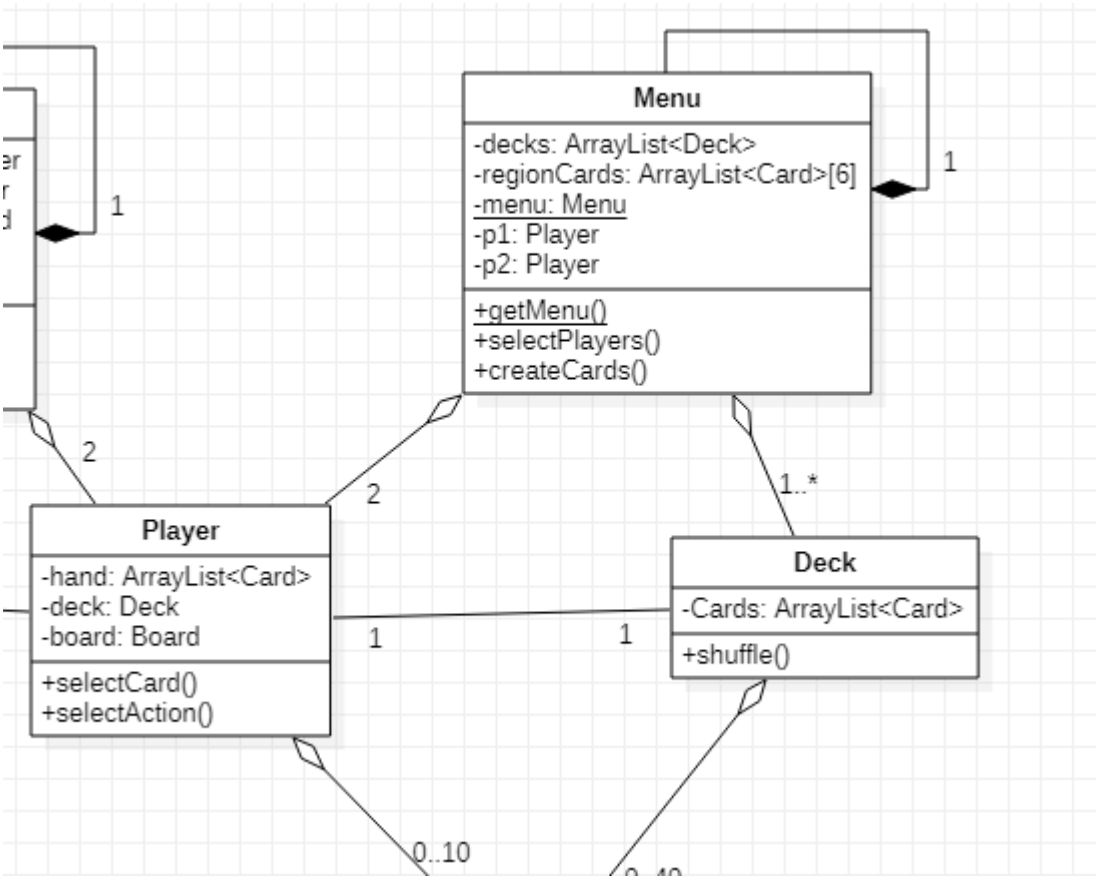
Diagrama UML

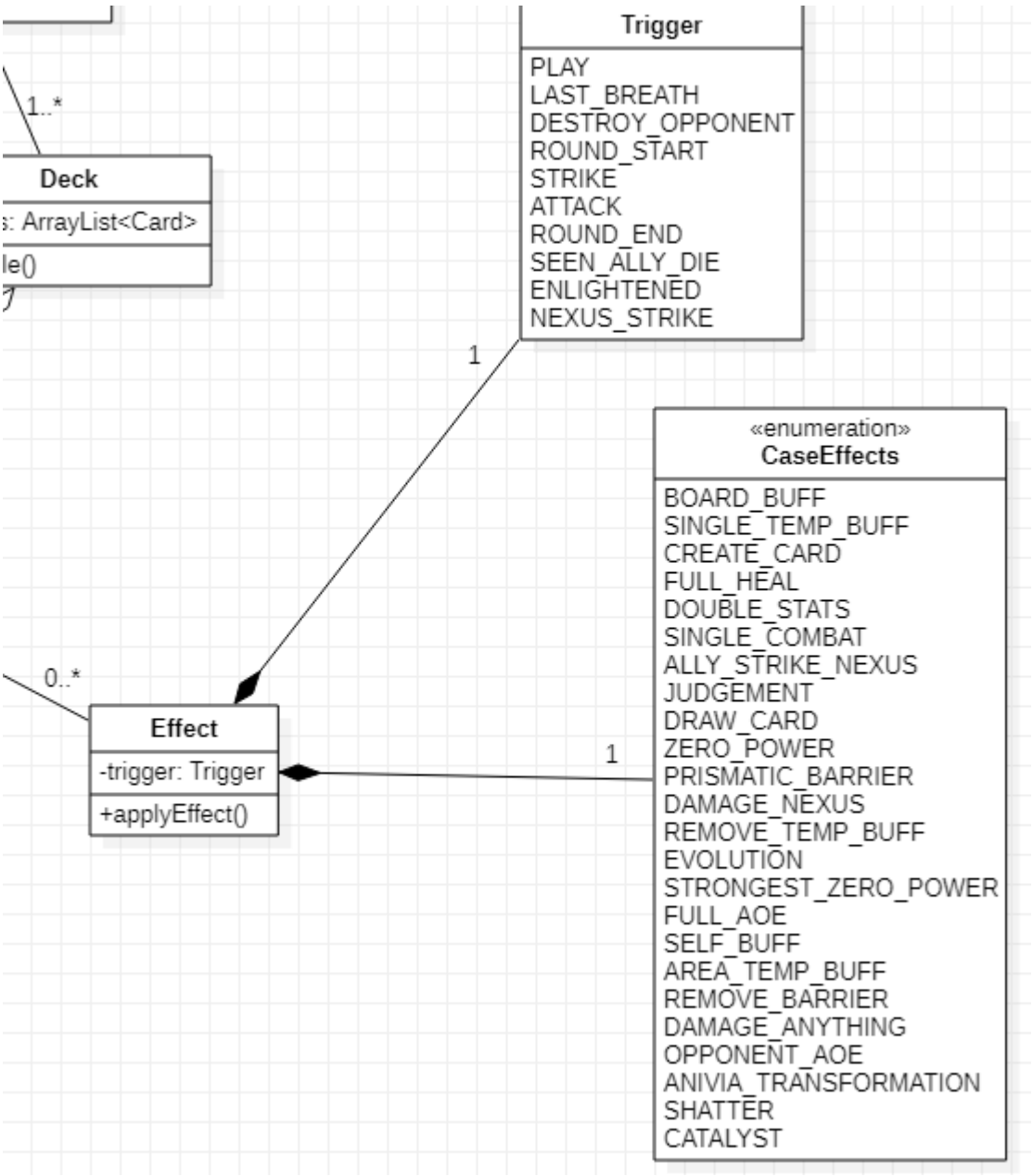





0..6
«enumeration»
Trait
DOUBLE_ATTACK
ELUSIVE
FURY
REGENERATION
BARRIER
QUICK_ATTACK
TOUGH
0..*
Follower
+traits: Trait
+playCard()
Champion
+evolve()
+checkEvolution()


0..10
0..40
Card
+effects: ArrayList<Effect>
+playCard()
0..6
Follower
+traits: Trait
+playCard()
0..*
Effect
-trigger: Trigger
+applyEffect()
Spell
+playCard()
Champion
+evolve()
+checkEvolution()


Menu
-decks: ArrayList<Deck>
-regionCards: ArrayList<Card>[6]
-menu: Menu
-p1: Player
-p2: Player
+getMenu()
+selectPlayers()
+createCards()
1
1
2
2
1..*
Player
-hand: ArrayList<Card>
-deck: Deck
-board: Board
+selectCard()
+selectAction()
Deck
-Cards: ArrayList<Card>
+shuffle()
1
1
0..10



Para a elaboração do UML, decidimos deixar apenas os atributos que tem uma relação com outra classe importante e os métodos que melhor representam cada classe. Cada um dos enumerates está explicado no código.