UNIVERSITÀ DEGLI STUDI DI PERUGIA
Dipartimento di Matematica e Informatica

Magic the project

Angeletti Leonardo
Belfiori Davide
Castellani Alessandro
Luchini Chiara
Mecarelli Marco
Moretti Gabriele
Polticchia Mattia
Proietti Filippo
Proietti Gabriele
Vagnarelli Federico

Anno Accademico 2021/2022

MAGIC THE GATHERING

Indice

Introduzione

Questo documento contiene la descrizione delle modifiche apportate al progetto *Magic*, in particolare verranno discussi:

- Implementazione del mulligan di tipo *London*
- Implementazione delle Evasion Ability
- Modifiche ai mazzi di gioco

1 Mulligan

Il mulligan è un processo opzionale mediante il quale qualsiasi giocatore può tentare di pescare una mano di apertura migliore prima di iniziare il gioco. I giocatori che vogliono prendere un mulligan devono annunciare le loro intenzioni in ordine di turno, quindi ogni giocatore che ha annunciato di voler prendere un mulligan ne prende uno. Questo processo viene ripetuto fino a quando tutti i giocatori decidono di non prenderne più. Vi sono varie tipologie di mulligan.

1.1 Mulligan originale

La prima regola del mulligan di Magic ("tutte terre/nessuna terra") consentiva ad un giocatore che aveva pescato zero o sette terre nella mano iniziale di rivelare quella mano al proprio avversario e rimescolarla, pescando una mano sostitutiva di sette carte. Ogni giocatore può farlo solo una volta per partita.

1.2 Paris Mulligan

La regola del mulligan di Parigi è stata introdotta al Sealed Deck Pro Tour del 1997 a Los Angeles. Un giocatore insoddisfatto della propria mano, per qualsiasi motivo e senza dover rivelare quella mano, può rimettere la propria mano nel proprio grimorio per avere l'opportunità di pescarne una nuova con una carta in meno, dopo aver mescolato. La scelta di prendere un mulligan è stata fatta dopo aver determinato il giocatore iniziale, ma prima di qualsiasi altra azione. I giocatori possono prendere più

mulligan, fino a quando non sono soddisfatti della loro nuova mano, o se rimangono con una mano di zero carte.

1.3 Vancouver Mulligan

Il mulligan di Vancouver ha sostituito il mulligan di Parigi nel gioco sanzionato. Per eseguire un mulligan di Vancouver, il giocatore rimette la mano nel proprio grimorio, quindi pesca una mano con una carta in meno. Una volta che tutti i giocatori mantengono le loro mani iniziali, ogni giocatore con meno carte della propria mano iniziale può fare Scry. Scry consente ad un giocatore di guardare un certo numero di carte dalla cima del proprio grimorio e metterle in fondo al grimorio o nuovamente in cima in qualsiasi ordine.

1.4 London Mulligan

Introdotto ufficialmente insieme al rilascio del Core Set 2020. Ogni giocatore rimescola le carte in mano nel proprio grimorio, pesca una nuova mano di carte pari alla dimensione della sua mano iniziale ogni volta che fa mulligan. Una volta che il giocatore è soddisfatto, mette una carta in fondo al suo grimorio per ogni volta che ha fatto mulligan. Non c'è nessun Scry.

La fase di inizializzazione del mulligan è comune ad ogni tipologia e prevede che ogni giocatore abbia pescato la prima mano.

Successivamente, a partire dallo *Starting Player*, comincia il giro di richieste così che ogni giocatore possa decidere se effettuare o meno il mulligan.

A tutti coloro che hanno confermato la scelta viene permesso di pescare la nuova mano.

Il processo di richieste e pescaggio viene gestito dalla regola *"103.4 part 3b (London Mulligan)"* descritta di seguito:

```
rule "103.4 part 3b (London Mulligan)"
when
    $g : Game(stage == Game.STARTING_STAGE,
        mulliganType == Game.LONDON_MULLIGAN,
        mulliganFlag == true,
        mulliganPhase == true);
```

Le condizioni di attivazione prevedono che:

- Il tipo di mulligan selezionato sia London
- L'inoltro delle richieste di mulligan è attivo (`mulliganFlag == true`)
- Il gioco si trova effettivamente nella fase di mulligan (`mulliganPhase == true`)

```
then
  // Se almeno 1 giocatore è nella lista di priorità ...
  if ($g.mulliganPriorityOrder.size() > 0) {
    // ... se esiste un giocatore a cui deve essere chiesto di fare mulligan ...
    if($g.mulliganPriorityMarker.hasNext()) {
      // ... ottengo il riferimento al giocatore
      $g.mulliganPriorityMarker.next();
      p = $g.mulliganPriorityMarker.getObject();
      // ... se ha superato il limite di mulligan possibili ...
      if(p.mulliganCounter == p.startingHandSize) {
        // ... rimuovo il giocatore dalla lista di priorità
        ...
      }else{
        // altrimenti inoltro la richiesta di mulligan
        ...
        // BLOCCO l'inoltro delle richieste agli altri giocatori
        $g.mulliganFlag = false;
      }
    }else{
      // Tutti i giocatori che hanno confermato la scelta
      // effettuano il mulligan
      for(Player player : $g.mulliganPriorityOrder) {
        player.takeMulligan($g.mulliganType);
      }
      // Resetto il puntatore sulla lista di priorità così da poter
```

```
      // ricominciare il giro di richieste ai giocatori che
      // hanno confermato il mulligan
      $g.mulliganPriorityMarker.toHead();
    }
  }else{
    $g.mulliganPhase = false; // Termina la fase di mulligan
    $g.mulliganFlag = false; // Termina l'inoltro delle richieste
    $g.discardTurn = true; // Inizia la fase di scarto
  }
  update($g);
end
```

L'ordine con cui vengono inoltrate le richieste di mulligan è gestito da una coda di priorità (`mulliganPriorityMarker`) dalla quale gli stessi giocatori possono essere rimossi in caso di risposta negativa.

Una volta inoltrata una richiesta di mulligan, occorre attendere la risposta del giocatore, nel mentre è importante che nessun altra richiesta venga inoltrata, per questo motivo la classe **Game** mette a disposizione il campo `mulliganFlag` che consente di interrompere e riprendere il giro di richieste.

Le risposte sono gestite da due regole separate (regole `"103.4 part 3b answerHandlerYes"` e `"103.4 part 3b answerHandlerNo"`) una per il caso affermativo ed una per quello negativo.

Entrambe reimpostano il `mulliganFlag` al valore `true`, allo stesso tempo, nel caso di risposta negativa, il giocatore viene rimosso dalla coda di priorità.

Una volta terminato il giro di richieste, a tutti i giocatori che hanno deciso di effettuare il mulligan viene permesso di pescare la nuova mano.

Quando la coda di priorità è vuota la fase di mulligan può terminare e cominciare quella di scarto.

Tutti i giocatori che devono scartare, possono farlo contemporaneamente, pertanto è sufficiente utilizzare una regola che si attiva per ogni giocatore che abbia effettuato almeno un mulligan.

```
rule "103.4 part 3c mulligan LONDON"
when
  // il gioco è nella fase di scarto
  $g: Game(stage == Game.STARTING_STAGE,
    mulliganType == Game.LONDON_MULLIGAN,
    mulliganFlag == false,
    mulliganPhase == false,
    discardTurn == true)
  // esiste un giocatore che non stà scartando e deve farlo
  $p: Player(!discarding, h: hand, shs: startingHandSize, mc: mulliganCounter)
  // un giocatore deve scartare quando il numero di carte nella sua mano
  // è maggiore del numero di carte iniziale meno il numero di mulligan effettuati
  eval(h.size() > shs - mc)
then
  System.out.println("103.4 part 3c mulligan London -> Player "+ $p.getNickname() );
  // invio la scelta della prossima carta da scartare
  MakeChoice choice = new MakeChoice();
  choice.idChoice = 11700;
  choice.choiceText = "Scegli quale carta scartare";
  for(int i = 0; i < $p.hand.size(); i++) {
    choice.addOption(i, $p.hand.get(i).getNameAsString());
  }
  GameEngine.sendToNode(choice, Game.CHOICE, Game.ONE_OF_CHOICE, $p.id);
  // imposto il giocatore nella fase di scarto
  $p.discarding = true;
  update($p)
end
```

Ad uno stesso giocatore vengono inviate tante richieste di scarto quante sono le volte che ha effettuato mulligan.

Tra una richiesta e la successiva occorre attendere che venga scelta la carta da scartare, per questo motivo la calsse **Player** è dotata del campo `discarding` il quale indica proprio che il giocatore sta scartando una carta.

Quando tutti i giocatori hanno terminato la fase di scarto il gioco può proseguire.

```
rule "103.4 part 3c mulligan LONDON discard end"
when
  // il gioco è nella fase di scarto
  $g: Game(stage == Game.STARTING_STAGE,
    mulliganType == Game.LONDON_MULLIGAN,
    mulliganFlag == false,
    mulliganPhase == false,
    discardTurn == true)
    // nessun giocatore deve scartare
  forall (
    $p: Player(!discarding, shs: startingHandSize, mc: mulliganCounter,
      h: hand, h.size() == shs - mc)
  )
then
  // Fine della fase di scarto
  $g.discardTurn = false;
  // Terminazione del Mulligan
  $g.finishedEndPhaseOfMulligan = true;
  update($g)
end
```

2 Evasion Ability

2.1 Definizione

Una **Evasion Ability** è un'abilità che limita il modo in cui una creatura (in difesa) possa bloccare una creatura attaccante.

Per **Abilità** si intende il testo presente su un' oggetto che spiega cosa fa e cosa può fare.

Per **Creatura** si intende un tipo di carta permanente. Ogni creatura dispone di un sottotipo che è correlato sia al tipo che al tipo di tribù della carta.

Per **Blocco** si intende la volontà di mandare una creatura in combattimento in maniera difensiva.

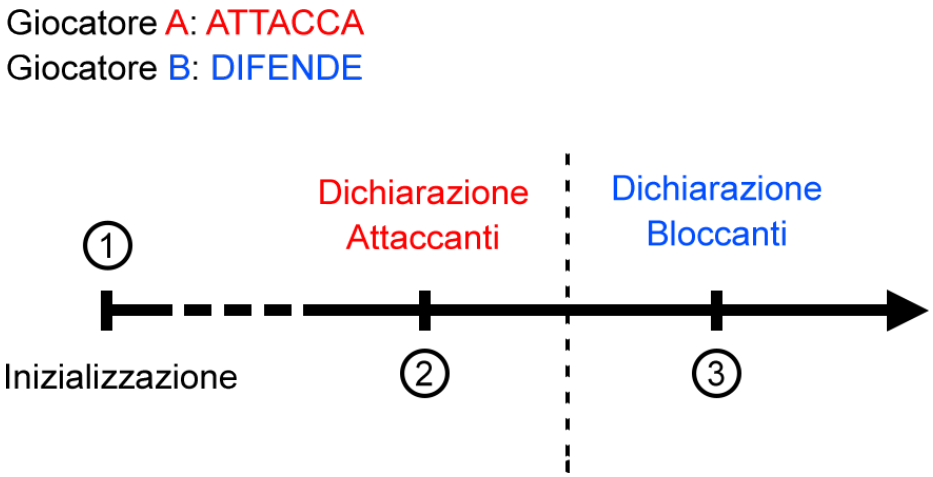
Invece per *Creatura attaccante* si intende una creatura che è stata dichiarata parte di un attacco (legale) durante la fase di combattimento oche è stata messa sul campo di battaglia per attaccare. Rimane una creatura attaccante finché non viene rimossa dal combattimento o la fase finisce (in base a quale viene prima).

2.2 Processo di attivazione

Il processo di attivazione delle Evasion Ability tiene conto di 3 macro fasi che si ripetono durante il gioco:

1. Inizializzazione
2. Dichiarazione delle creature attaccanti
3. Dichiarazione delle creature difensori

Consideriamo ad esempio il caso di due giocatori, A e B, il primo attacca ed il secondo difende.



2.2.1 Inizializzazione

Ogni Evasion Ability è caratterizzata da una regola che la definisce come tale, ad esempio:

702.28a. Shadow is an evasion ability.

Durante la fase di inizializzazione tali regole vengono attivate al fine di ricercare all'interno della **libreria** di ogni giocatore tutte le creature che posseggono una o più Evasion Ability ed impostare tutti i flag necessari così da poterle utilizzare successivamente.

L'implementazione di queste regole segue uno schema comune a tutte le Evasion Ability, nel caso di *Shadow* si presenta come segue:

```
rule "702.28a"
  when
    $g : Game(stage == Game.STARTING_STAGE)
    $p : Player($id : id,
         $nickname: nickname,
         $deck: deck;
         $lib: library, library.size() > 0)
    $c : Card() from $lib
    $la: LinkedList() from $c.getKeywordAbilities()
    $a : Ability(keyword_ability==false && keyword_text.equals("shadow"))
          from $la
 then
         $a.setKeyword_ability(true);
         $a.setEvasion_ability(true);
         update($p)
end
```

Le condizioni di attivazione prevedono che il gioco si trovi nello **STARTING_STAGE**, che la libreria di un certo giocatore sia stata riempita e che, all'interno di questa, sia presente una carta con abilità `"shadow"` che non sia stata già processata.

Se tutto ciò risulta verificato, la regola riconosce l'abilità come Evasion Ability impostando un flag booleano.

2.2.2 Dichiarazione degli attaccanti

Il giocatore **A** dichiara quali, tra le creature che egli controlla, attaccheranno l'avversario.

Queste vengono memorizzate all'interno di un campo apposito della classe **Game**, chiamato `attackingCreatures` (regole *508.1a choice* e *508.1a answerHandler*).

2.2.3 Dichiarazione dei bloccanti

Il giocatore **B** dichiara quali, tra le creature che egli controlla, bloccheranno gli attacchi.

Queste vengono a loro volta memorizzate all'interno di un campo della classe **Game**, chiamato `blockingCreatures` (regole *509.1a choice* e *509.1a answerHandler*).

A questo punto entrano in gioco le Evasion Ability, in particolare è stato scelto di separare il controllo sulla validità dei blocchi in due passaggi.

Il motivo di tale decisione può essere compreso analizzando la regola relativa all'abilità *"Shadow"* (regola 702.28b), la quale può essere intesa come l'unione di 2 sotto-regole:

Parte 1: *A creature with shadow can't be blocked by creatures without shadow, and ...*

Parte 2: *... a creature without shadow can't be blocked by creatures with shadow.*

La prima parte afferma che una creatura che possiede l'abilità *"Shadow"* non può essere bloccata da una creatura che al contrario non la possiede, mentre la seconda impedisce ad una creatura con *"Shadow"* di bloccare un attaccante che invece ne è sprovvisto.

Quest'ultima affermazione consente di effettuare una distinzione iniziale sulle creature che una certa carta, dichiarata come bloccante, può effettivamente bloccare.

Il primo controllo consiste proprio nell'attivazione delle regole che eseguono una discriminazione a priori sugli attaccanti.

Ognuna di queste può agire su un campo specifico della classe **Permanent** chiamato `attackersICanBlock`, tale attributo viene inizializzato come una lista contenente tutte le creature attaccanti nel momento in cui lo stesso permanente viene scelto come difensore. (regola *509.1a answerHandler*)

Nel caso di *"Shadow"* la regola si presenta come segue:

```
rule "702.28b part 2"
when
  $g:Game(stage == Game.GAME_STAGE,
    stepCheckEvasionAbility.listReference.size() > 0,
    $sce: stepCheckEvasionAbility,
    $blockers: blockingCreatures.listReference,
    $attackers: attackingCreatures.listReference)
  eval($g.currentStep.getObject().name == "declare blockers")
  eval($g.stepCheckEvasionAbility.getObject() == "702.28b part 2")
then
  // ricerca di tutti i BLOCCANTI CON Shadow
  LinkedList blockersWithShadow = new LinkedList();
  for (Permanent blocker : $blockers) {
    if (blocker.checkKeywordAbility("shadow")){
      blockersWithShadow.add(blocker);
    }
  }
  // ricerca di tutti gli ATTACCANTI SENZA Shadow
  LinkedList attackersWithoutShadow = new LinkedList();
  for (Permanent attacker : $attackers) {
    if (! attacker.checkKeywordAbility("shadow")) {
      attackersWithoutShadow.add(attacker);
    }
  }
  // per ogni bloccante con Shadow ...
  for (Permanent blocker : blockersWithShadow){
```

```
    // ... rimuovo, dalla lista degli attaccanti che
    // possono essere bloccati, tutti quelli senza shadow
    blocker.attackersICanBlock.removeAll(attackersWithoutShadow);
  }
  $sce.next();
  update($g)
end
```

Le condizioni di attivazione richiedono ovviamente che il gioco sia nella fase di dichiarazione dei bloccati:

```
    eval($g.currentStep.getObject().name == "declare blockers")
```

L'attivazione delle Evasion Ability viene gestita in modo sequenziale, a questo scopo è stato aggiunto un campo nella classe **Game**, chiamato `stepCheckEvasionAbility`.

Tale attributo consiste in una lista scorrevole per mezzo di un puntatore, l'elemento puntato da quest'ultimo determina l'abilità da attivare, per questo motivo la seconda condizione di attivazione della regola precedente consiste in:

```
    eval($g.stepCheckEvasionAbility.getObject() == "702.28b part 2")
```

L'avvio e la terminazione della sequenza delle abilità è gestita da 2 ulteriori regole:

```
rule "canBlock"
when
        $g:Game(stage == Game.GAME_STAGE,
                $de : defendingPlayers,
                $g.blockingCreatures.getSize()>0,
                stepTimeFrame == Game.BEGIN_TIME_FRAME,
                $sce: stepCheckEvasionAbility.listReference)
        eval($g.currentStep.getObject().name == "declare blockers")
        eval($g.stepDeclareBlockers.getObject() == "canBlock")
```

```
        eval($sce.size() == 0)
        $p: Player($id : id) from $de.object
then
        // Inizializzazione della lista
        $sce.add("702.28b part 2"); // Shadow
        // ...
        $sce.add("canBlock end");
        // Avvio della sequenza
        $g.stepCheckEvasionAbility.next();
        update($g);
end

rule "canBlock end"
when
        $g:Game(stage == Game.GAME_STAGE,
                stepCheckEvasionAbility.listReference.size() > 0,
                $sce: stepCheckEvasionAbility.listReference)
        eval($g.currentStep.getObject().name == "declare blockers")
        eval($g.stepCheckEvasionAbility.getObject() == "canBlock end")
then
        // Reset stepCheckEvasionAbility
        $g.resetStepCheckEvasionAbility();
        // Prosecuzione della dichiarazione dei bloccanti
        $g.stepDeclareBlockers.next();
        update($g);
end
```

La prima (`canBlock`) si occupa dell'inizializzazione della lista di Evasion Ability e dell'avvio della sequenza di controlli.

Ogni controllo, al termine della propria esecuzione, sposta in avanti il puntatore al controllo successivo.

Quando questo punta all'ultimo elemento della lista, ovvero `"canBlock end"`, la regola corrispondente si attiva, resetta la lista `stepCheckEvasionAbility` e prosegue la fase di dichiarazione dei bloccanti.

Al momento *"Shadow"* è l'unica abilità che presenta questa particolare composizione, tuttavia, in previsione del fatto che altre regole con la stessa caratteristica possano essere aggiunte in futuro, è stato deciso di mantenere la divisione del controllo in due fasi.

Una volta terminato il filtraggio preliminare, il giocatore **B** sceglie, per ogni bloccante dichiarato, quale attaccante deve bloccare.

Ovviamente la scelta viene consentita solo tra le creature rimaste nella lista `attackersICanBlock`, qualora questa lista dovesse essere vuota, il permanente in questione non può più essere utilizzato come bloccante (regole *509.1a second choice* e *509.1a second answerHandler*).

Una volta che sono note tutte le associazioni attaccante-bloccante, è possibile verificare se effettivamente i blocchi dichiarati siano validi, ovvero attivare la prima parte della regola *"Shadow"* e tutte le Evasion Ability rimanenti.

Esattamente come nella fase precedente, anche in questo caso le abilità vengono attivate in sequenza, sfruttando la stessa lista a scorrimento `stepCheckEvasionAbility`.

Perciò sono state aggiunte 2 regole che gestiscono l'inizio e la terminazione di tale successione.

```
rule "isBlockValid"
when
        $g:Game(stage == Game.GAME_STAGE,
            $de : defendingPlayers,
            stepTimeFrame == Game.BEGIN_TIME_FRAME,
            $sce: stepCheckEvasionAbility.listReference)
        eval($g.currentStep.getObject().name == "declare blockers")
        eval($g.stepDeclareBlockers.getObject() == "isBlockValid")
        eval($sce.size() == 0)
        $p: Player($id : id) from $de.object
then
```

```
        // Reset del flag di validità dei blocchi
        $g.blockValid = true;
        if($g.blockingCreatures.getSize() == 0) {
      // se non ci sono più bloccanti,
      // non è necessario effettuare i controlli
            $g.stepDeclareBlockers.next();
        } else {
      $sce.add("702.9b"); // Flying
      $sce.add("702.13b"); // Intimidate
      $sce.add("702.14c"); // Landwalk
      $sce.add("702.28b part 1"); // Shadow
      $sce.add("702.31b"); // Horsemanship
      $sce.add("702.36b"); // Fear
      $sce.add("702.111b"); // Menace
      $sce.add("702.118b"); // Skulk
      // ...
      $sce.add("isBlockValid end");
      // Avvio della sequenza
      $g.stepCheckEvasionAbility.next();
        }
        update($g);
end

rule "isBlockValid end"
when
        $g:Game(stage == Game.GAME_STAGE,
                stepCheckEvasionAbility.listReference.size() > 0,
                $sce: stepCheckEvasionAbility.listReference)
        eval($g.currentStep.getObject().name == "declare blockers")
        eval($g.stepCheckEvasionAbility.getObject() == "isBlockValid end")
then
        // Reset stepCheckEvasionAbility
        $g.resetStepCheckEvasionAbility();
        // Se c'è almeno 1 blocco NON Valido ...
        if (! $g.blockValid) {
          // Il caso in cui sia stato dichiarato
          // un blocco non valido verrà trattato
          // nel dettaglio nella sezione 2.3
```

```
        } else {
          // ... proseguo con la dichiarazione dei bloccanti
          $g.stepDeclareBlockers.next();
        }
        update($g);
end
```

Ogni Evasion Ability dovrà quindi occuparsi di controllare se un attaccante che possiede quella particolare abilità possa essere effettivamente bloccato dalla creatura designata dal giocatore difensore.

Se il blocco non risulta valido, la regola può agire su un flag booleano della classe **Game**, chiamato `blockValid`, in particolare imposta il suo valore a `False`.

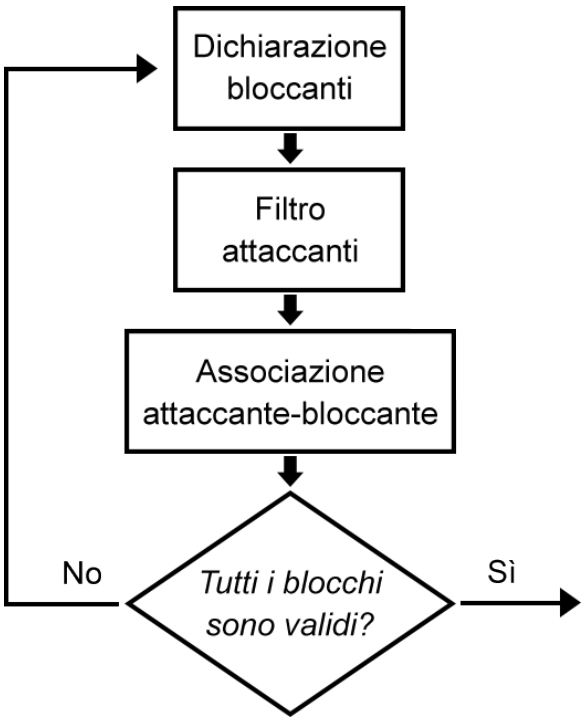
Nel caso di *"Shadow"* l'implementazione è la seguente:

```
rule "702.28b part 1"
when
        $g:Game(stage == Game.GAME_STAGE,
        stepCheckEvasionAbility.listReference.size() > 0,
        $sce: stepCheckEvasionAbility,
            $blockers: blockingCreatures.listReference,
            $attackers: attackingCreatures.listReference)
        eval($g.currentStep.getObject().name == "declare blockers")
        eval($g.stepCheckEvasionAbility.getObject() == "702.28b part 1")
then
  // Per ogni bloccante ...
  for (Permanent blocker : $blockers) {
    // ... se ha NON Shadow tra le evasion ability ...
    if (! blocker.checkKeywordAbility("shadow")) {
      // ... controllo le creature che blocca ...
      for (Permanent attacker : blocker.blockedCreatures.listReference) {
        // ... se almeno 1 attaccante ha shadow ...
        if (attacker.checkKeywordAbility("shadow")){
          // ... il blocco non è valido
         $g.blockValid = false;
        }
```

```
      }
    }
  }
  $sce.next();
  update($g)
end
```

Le condizioni di attivazione sono le stesse per tutte le Evasion Ability, l'unica differenza sta nel controllo della stringa puntata dal puntatore nella lista di successione delle abilità.

Lo schema seguente riassume graficamente gli step della fase di dichiarazione dei bloccanti.



2.3 Gestione dei blocchi illegali

Nel caso in cui sia stato dichiarato un blocco non valido, il giocatore che difende deve ricominciare dall'inizio la fase di dichiarazione dei bloccanti.

La gestione di questa situazione è affidata alla regola *isBlockValid end*, in particolare l'implementazione è la seguente:

```
// Se c'è almeno 1 blocco NON Valido ...
if (! $g.blockValid) {
    // ... ricomincio la dichiarazione dei bloccanti
    $g.stepDeclareBlockers.movePointer(0);
    // per ogni bloccante dichiarato ...
    for (Permanent blocker : $g.blockingCreatures.listReference) {
        // ... resetto la lista degli attaccanti bloccati
            blocker.resetBlockedCreatures();
    }
    // resetto la lista dei bloccanti
    $g.blockingCreatures.listReference.clear();
    $g.blockingCreatures.toHead();
    // resetto il defending player
    $g.defendingPlayers.toHead();

    GameEngine.sendToNode("Il difensore ha dichiarato un blocco illegale.");
}
```

L'intera fase di dichiarazione dei bloccanti è gestita a sua volta da una lista a scorrimento, pertanto la prima operazione che la regola deve effettuare è quella di spostare il puntatore di tale lista al primo elemento.

Successivamente occorre resettare, per ogni bloccante, la lista degli attaccanti al quale era stato assegnato, quindi si procede con pulizia della lista dei bloccanti.

Infine si resetta il riferimento al giocatore difensore.

2.4 Note sull'implementazione delle Evasion Ability

La gestione delle Evasion Ability sopra implica che l'implementazione di ogni abilità segua uno schema comune con tutte le altre, ovvero:

- Le regole di inizializzazione (sezione 2.2.1) sono identiche, l'unica differenza sta nella stringa di controllo (`"shadow"`, `"flying"`, ecc...).
- Le condizioni di attivazione dei controlli veri e propri sono anch'esse uguali, ad eccezione della stringa contenente il numero della regola.

Sono stati inoltre aggiunti alcuni metodi di utilità, in particolare la funzione `checkKeywordAbility` della classe **Permanent**, la quale, data una stringa, restituisce un valore booleano che indica la presenza o meno di una **Keyword Abilty** con nome uguale alla stringa in input, tra tutte quelle appartenenti alla carta.

L'unica difficoltà nell'implementazione di una Evasion Ability sta nel fatto di non poter accedere alla lista delle creature bloccanti a partire da un dato attaccante.

In altre parole, ogni **Permanent** attaccante ha un attributo chiamato `blockedBy` che contiene i riferimenti a tutte le creature che lo bloccano.

Tuttavia questa lista viene riempita solo nel momento in cui una creatura **dichiarata** come bloccante lo diventa a tutti gli effetti

***509.1g**: Each chosen creature still controlled by the defending player **becomes** a blocking creature ...*

ovvero dopo aver superato i controlli delle Evasion Ability.

verificareQuesto vuol dire che, per controllare che un blocco sia valido, occorre scorrere la lista di tutti i bloccanti, per ognuno di questi si controllano gli attaccanti al quale è stato assegnato che possiedono una certa Evasion Ability, se almeno una di queste non risulta soddisfatta, il blocco non è valido.

Nelle sezioni seguenti descriviamo nel dettaglio le singole Evasion Ability con la relativa implementazione, ad eccezione di *Shadow*, la quale è stata approfondita in precedenza.

2.5 Flying

Regola "702.9a" In questa regola definiamo il funzionamento di "Flying", la quale viene descritta come di seguito:

> Una creatura con flying può essere bloccata solo da da creature che hanno flying e/o reach.
> Una creatura che vola, può bloccare sia le creatura con flying sia le creature che non lo hanno.

Le condizioni per il trigger quindi sono:

1. Non può essere bloccata se chi difende non ha flying o reach.
2. La creatura in difesa (che ha flying) può bloccare sia se l'attaccante vola sia se non vola.

In questa regola, per ogni creatura che blocca, si controlla se non possiede ne **flying** ne **reach** fra le keyword ability.

```
for (Permanent blocker : $blockers)
{
  if (!(blocker.checkKeywordAbility("flying") OR
    blocker.checkKeywordAbility("reach")))
    {
    }
}
```

Se questo è vero, si controlla poi se l'attaccante possiede l'evasion ability in questione e, in caso, il blocco viene dichiarato non valido.

```
for (Permanent attacker : blocker.blockedCreatures.listReference)
{
  if (attacker.checkKeywordAbility("flying"))
  {
    System.out.println("Flying: Blocco non valido!");
    $g.blockValid = false;
  }
}
```

Figura 1: Flying

2.6 Intimidate

Regola "702.13b part 1" In questa regola definiamo il funzionamento di "Intimidate", la quale viene descritta come di seguito:

> *Una creatura con intimidate non può essere bloccata eccetto dalle creature artefatto e/o da creature che condividono i colori con essa.*

Se le condizioni vengono soddisfatte si utilizzano due variabili booleane `shareColor` e `typeArtifactCreature` inizializzate entrambe a false, le quali indicano rispettivamente la presenza o meno di un colore in comune tra l'attaccante e il bloccante e/o di una creatura artefatto come bloccante. In seguito si controlla per ogni bloccante se le condizioni di blocco sono soddisfatte in particolar modo tramite

```
if(blocker.cardType[0].contains("creature") &&
   blocker.cardType[0].contains("artifact"))
   {
   typeArtifactCreature = true;
}
```

Verifichiamo se il bloccante è di tipo "creatura artefatto", in caso positivo settiamo la variabile `typeArtifactCreature = true`, mentre

```
for(String coloreAttaccante: attacker.getColorIndicator()[0])
{
  if(blocker.getColorIndicator()[0].contains(coloreAttaccante))
  {
    shareColor = true;
  }
}
```

Controlliamo se fra i colori del bloccante è presente almeno uno dei colori dell'attaccante, in caso positivo settiamo la variabile `shareColor = true`. Per concludere definiamo quando il blocco non è valido con il seguente controllo

```
if (attacker.checkKeywordAbility("intimidate") &&
   !(shareColor == true || typeArtifactCreature == true))
   {
   System.out.println("Blocco non valido!");
   $g.blockValid = false;
   }
}
```

Dobbiamo assicurarci che l'attaccante abbia "intimidate" come **keyword ability** e che il bloccante non verifichi uno dei requisiti necessari per il blocco.

Figura 2: Intimidate

2.7 Landwalk

Regola "702.14c": In questa regola ricerchiamo l'evasion ability "Landwalk", che viene descritta come:

*Una creatura che possiede **Landwalk**, non può essere bloccata fintanto che il giocatore in difesa controlla almeno una terra con il tipo di terra specificato (esempio **"islandwalk"**)*

Le condizioni per il trigger quindi sono:

1. Può essere bloccata se il difensore possiede terre con il tipo specificato dall'abilità

Questo termine è presente all'interno delle regole nella seguente forma "**[type]walk**", dove *type* di solito indica il tipo di una terra (island → islandwalk).
In questa regola si controlla per ogni bloccante, ricavati tramite *blockingCreatures.listReference*, quale creatura intende bloccare e si controlla se dispone di una delle abilità **landwalk**.

Successivamente viene controllato se sono presenti delle terre (appartenenti al difensore) che combaciano con il tipo specificato nell'abilità landwalk (della creatura attaccante) e, in caso, il blocco viene considerato non valido.

```
rule "702.14c"
dialect "mvel"
no-loop true
agenda-group "general"
  when
    // Condizioni per il trigger della regola
  then
    // Questa variabile contiene il tipo di terra in base
    // a quale landwalk possiede il mostro attaccante.
    String landwalk = "";
    // Si scorre la lista dei bloccanti
    for (Permanent blocker : $blockers)
    {
      // Si controllano le creature che bloccano
      for (Permanent attacker : blocker.blockedCreatures.listReference)
      {
        // Si verifica se l'attaccante ha una tra le possibili
        // abilità landwalk.
        // Alla variabile "landwalk" viene assegnato il tipo
        // di terreno corrispondente alla landwalk trovata.
        // Esempio:
        if(attacker.checkKeywordAbility("islandwalk"))
        {
          landwalk = "Island";
        }
        // Scorre le carte presenti nel terreno
        for(Permanent perm : $g.battleField)
        {
          // Considera solo le carte di tipo terra
          if(perm.cardType.size() > 0 && perm.cardType[0].contains("land"))
          {
            // Controlla che appartengono al difensore
            if(perm.getIdController == $id)
            {
```

```
                // Verifica se la terra è dello stesso tipo
                // specificato dalla variabile "landwalk"
                if perm.subtype[0].contains(landwalk)
                {
                  // Se è così, il blocco non è valido.
                  SystemF.out.println("Blocco non valido!");
                  $g.blockValid = false;
                }
              }
            }
          }
        }
    }
    $sce.next();
    update($g)
end
```

Figura 3: Mountainwalk

2.8 Horsemanship

Regola "702.31b": In questa regola ricerchiamo l'evasion ability "Horsemanship", che viene descritta come:

Una creatura con horsemanship non può essere bloccata da una creatura senza horsemanship. Una creatura con horsemanship può bloccare una creatura senza horsemanship..

Le condizioni per il trigger quindi sono:

1. Non può essere bloccata da creature senza horsemanship.
2. Può bloccare creature senza horsemanship.

In questa regola, una creatura che ha "Horsemanship" come evasion ability,se il bloccate ha horsemanship può bloccare creature con o senza horsemanship, e un attaccante con horsemanship non può essere bloccato da creature senza horsemanship. Faccio un controllo prendendo ogni bloccante e controllo se non ha horsemanship tra le evasion ability

```
for (Permanent blocker : $blockers)
{
  if (! blocker.checkKeywordAbility("horsemanship"))
  {
  }
}
```

Poi controllo le creature che il bloccante, che non ha "horsemanship" tra le evasion ability, può bloccare e se almeno un attaccante ha "horsemanship" tra le evasion ability allora il blocco non è valido

```
for (Permanent attacker : blocker.blockedCreatures.listReference)
{
  if (attacker.checkKeywordAbility("horsemanship"))
  {
    System.out.println("Blocco non valido!");
    $g.blockValid = false;
  }
}
```

Per la seconda condizione del trigger non c'è bisogno di implementare codice dato che è un blocco comune.

Figura 4: Horsemanship

2.9 Fear

Regola "702.36b": In questa regola ricerchiamo l'evasion ability "Fear", che viene descritta come:

Una creatura con fear non può essere bloccata ad eccezione delle creature artefatto e/o le creature nere..

Le condizioni per il trigger quindi sono:

1. Può essere bloccata da creature nere;
2. Può essere bloccata da creature artefatto;
3. Non può essere bloccata da nessun'altra carta;

Nella regola vengono create due variabili booleane, `colorBlack` e `typeArtifactCreatureFear`, inizializzate entrambe a false, che serviranno poi per le condizioni poste dalla regola. `colorBlack` indentifica un bloccante di colore nero, mentre `typeArtifactCreatureFear` indentifica un bloccante che è di tipo creatura artefatto.

Dopo le due variabili si controlla per ogni bloccante le creature che blocca:

```
for (Permanent blocker : $blockers)
{
  for (Permanent attacker : blocker.blockedCreatures.listReference)
  {
  }
}
```

Verifichiamo poi se il bloccante è di colore nero con la seguente condizione, e se lo è la variabile colorBlack viene settata a true

```
if(blocker.colorIndicator[0].contains("black"))
{
  colorBlack = true;
}
```

Seguendo la stessa idea, si fa lo stesso per verificare se il bloccante è una creatura di tipo artefatto, e se lo è la variabile typeArtifactCreatureFear viene settata a true

```
if(blocker.cardType[0].contains("creature") &&
   blocker.cardType[0].contains("artifact"))
   {
   typeArtifactCreatureFear = true;
}
```

Infine si fa un controllo unico per definire quando il blocco non è valido, cioè quando i due controlli fatti prima siano a false e considerando che l'attaccante deve avere "Fear" come **keyword ability**.

```
if ((colorBlack == false && typeArtifactCreatureFear == false)
{
  if (attacker.checkKeywordAbility("fear"))
  {
    // Il blocco non è valido
    System.out.println("Blocco non valido!");
    $g.blockValid = false;
  }
}
```

Figura 5: Fear

2.10 Menace

Regola "702.111b": In questa regola ricerchiamo l'evasion ability "Menace", che viene descritta come:

Una creatura con menace non può essere bloccata eccetto da due o più creature..

In questa regola, una creatura che ha "Menace" come evasion ability, non può essere bloccata se l'avversario ha solo una creatura che blocca. Inizialmente si fa un selezione per ogni attaccante, si controlla se ha Menace tra le evasion ability e si inizializza un contatore per il numero di creature che lo bloccano.

```
for (Permanent attacker : $attackers)
{
  if (attacker.checkKeywordAbility("menace"))
  {
    int numero_bloccanti = 0;
  }
}
```

Poi si fa un controllo dove per ogni bloccante dichiarato, controlliamo che creatura sta bloccando. Se l'attaccante bloccato è quello con Menace che stiamo controllando il contatore dei bloccanti viene aumentato di uno.

```
for (Permanent blocker : $blockers)
{
  if(blocker.blockedCreatures.listReference.contains(attacker))
  {
    numero_bloccanti ++;
  }
}
```

Infine si fa un controllo sul numero di bloccanti che sono stati trovati, e se il numero è uguale a 1, quindi minore di 2, il blocco non è valido.

```
if (numero_bloccanti == 1)
{
  // ... il blocco non è valido
  System.out.println("Menace: Blocco non valido!");
  $g.blockValid = false;
}
```

Figura 6: Menace

2.11 Skulk

Regola "702.118b": In questa regola ricerchiamo l'evasion ability "Skulk", che viene descritta come:

Una creatura con skulk non può essere bloccata da creature con forza maggiore

Una creatura che ha skulk tra le evasion ability non può essere bloccata da creature che hanno un attacco maggiore, quindi solo da creature più deboli.
Si prende ogni bloccante e si ottiene il proprio valore di attacco, visto che va controllato quello, e poi vengono controllate le creature che blocca.

```
for (Permanent blocker : $blockers)
{
    b_power = Integer.parseInt(blocker.power[0])
    for (Permanent attacker : blocker.blockedCreatures.listReference)
    {
    }
}
```

All'interno dell'ultimo for insieriamo il controllo per il blocco, dove se l'attaccante ha skulk come evasion ability e l'attacco dell'attaccante è minore del bloccante, allora il blocco non è valido

```
if (attacker.checkKeywordAbility("skulk") &&
   Integer.parseInt(attacker.power[0]) < b_power)
   {
   // ... il blocco non Ã¨ valido
   System.out.println("Skulk: Blocco non valido!");
   $g.blockValid = false;
   }
}
```

Figura 7: Skulk

3 Deck nuovi

3.1 Deck verde

Utilizzando delle carte presenti nel file card.json si è potuto implementare un nuovo deck. I deck presenti sono strutturati nel seguente modo:

1. Deck con creature rosse e nere (con land relative, Mountain e Swamp);
2. Deck con creature blu e bianche (con land relative, Island e Plains);

Il deck che è stato aggiunto contiene creature verdi con relativa land, forest. Esempio delle carte presenti nel deck verde:

Figura 8: Creatura verde e Land Forest

Nel file clientL.js è stato dichiarato il nuovo deck con i relativi id delle carte presenti nel file card.json. Tramite un array si seleziona le quantità di carte presenti nel deck con un massimo di 60 carte .

```
// - DECK 3 DECLARATION ---------------------------------------
var deck3= Array(); // green
i=0;
for(i; i<=9;i++)
  deck3.push(1068); //Forest
for(i; i<=15;i++)
  deck3.push(2886); //Immerwolf (creature green)
...
for(i; i<=37;i++)
  deck3.push(1511); //Frontier guide (creature green)
...
for(i; i<=51;i++)
  deck3.push(2166); //Enchant Decomposition (green)
...
for(i; i<=59;i++)
  deck3.push(2161); //Enchant Armor of thorns (green)
```

Sempre nel file clientL.js si è aggiunto il riferimento al deck3 nella scelta del deck tramite valore, come si può vedere nel codice sotto. (questo è

stato fatto sia per la configurazione del multiplayer sia per la configurazione di 2 player).

```
nick = document.getElementById("nickname").value;
if ((document.getElementById("deck").value) == "1") {
  deck = deck1;
} else if ((document.getElementById("deck").value) == "2") {
  deck = deck2;
} else if ((document.getElementById("deck").value) == "3") {
  deck = deck3;
}
```

Per poter selezionare il deck verde nell'interfaccia grafica, e quindi utilizzarlo, è stato modificato il file clientL.html, aggiungendo nel form della scelta del deck verde.

```
<form>
    <label>Choose deck</label>
    <select id="deck" name="deck">
        <option value="1">Rosso Nero</option>
        <option value="2">Blu Bianco </option>
        <option value="3">Mono Verde</option>
    </select>
</form>
```

3.2 Deck con sole creature

Per un fattore di comodità, si è implementato anche un deck con sole creature, in modo da poter provare le evasion ability e in generale per poter provare le varie creature di ogni mazzo. In questo modo se voglio prima provare una determinata creatura rossa contro una blu, e poi contro una verde, non devo per forza riavviare una nuova partita, ma posso fare tutto nella stessa. In questo mazzo sono state inserite tutte le land utilizzate negli altri deck.

```
// - DECK 0 DECLARATION ---------------------------------------
var deck0_creatures_only = Array();
deck0_creatures_only[0] = 1065; // Island
deck0_creatures_only[1] = 1066; // Swamp
```

```
...
deck0_creatures_only[9] = 880; // Forgotten creation (Creature, Skulk, Blue)
...
deck0_creatures_only[12] = 2194; // Boggart Brute (Creature, Menace, Red)
...
deck0_creatures_only[17] = 2886; // Immerwolf (Creature, Intimidate, Green)
...
deck0_creatures_only[26] = 1201; // Llanowar Elves (Creature, Green)
```

A differenza degli altri mazzi, questo mazzo non è completo. Ogni creatura e land è inserita una volta sola. Questo in modo che ogni volta che viene inserita una nuova creatura in un deck, può essere inserita anche qui senza modificare i vari parametri dei for per non uscire dal totale delle 60 carte. Inoltre essendo un deck con sole creature di prova, non c'è bisogno che sia completo. Anche in questo caso si è aggiunto il riferimento del deck sole creature nella selezione dei deck, sempre nel file clientL.js

```
nick = document.getElementById("nickname").value;
if ((document.getElementById("deck").value) == "1") {
  deck = deck1;  // rosso / nero
} else if ((document.getElementById("deck").value) == "2") {
  deck = deck2; // blu / bianco
} else if ((document.getElementById("deck").value) == "3") {
  deck = deck3; // mono verde
} else if ((document.getElementById("deck").value) == "4") {
  deck = deck0_creatures_only;
}
```

E anche per questo deck è stato modificato il form nel file clientL.html, aggiundo una nuova opzione per selezionare il deck con sole creature.

```
<form>
    <label>Choose deck</label>
    <select id="deck" name="deck">
        <option value="1">Rosso Nero</option>
        <option value="2">Blu Bianco </option>
        <option value="3">Mono Verde</option>
        <option value="4">Solo Creature</option>
    </select>
</form>
```