Durak Specification

Version 1.0.0

Eugen Kiss

An initial game of Durak is a triple (D, P, S). The components are defined as follows:

- D is a list (deck) of cards. A card c has a suit $c.s \in \{C, D, H, S\}$ and a face $c.f \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, J, D, K, A\}$.
- P is the set of players. Each Player $p \in P$ has the following elements:
  - id is a unique identifier for the player.
  - $hand \subseteq D$ is the set of cards currently in the player's hand.
  - $\sigma : S \times [C] \rightarrow E$ is a function representing p's strategy that maps the settings and a list of all game configurations so far to an event.
- S denotes the settings of the game. S may contain a selection of the following elements:
  - `TRANSFER`: Transfers are allowed.
  - `FLASH`: Flashes are allowed.

  The containment of an element from above in S signifies that this particular game setting is activated.

Configurations

C is the domain of the configurations. A configuration is a complete snapshot of a game's "situation". A configuration contains the following elements:

- Trump: The current trump suit or $\perp$.
- Deck: The (remaining) cards in the deck.
- Pile: The discarded cards.
- Field: The cards on the field as a set of tuples. Each first component is the attacker card and each second component is the respective defender card or $\perp$ in case of an as-of-yet unbeaten attacker card.
- Players: The list of all players. The order in this list determines the order of attacks.
- Inactive: The set of the players who are out.
- Active: The player whose turn it is or $\perp$.
- Principal: The principal attacker or $\perp$
- Attack#: The count of attacks in an attack phase. Needed to determine the end of an attack phase.
- Durak: The id of the previous round's durak or $\perp$ if it is the first round.
- Phase: The current phase.
- Event: The last event or $\perp$.

We declare the player on the first position in Players to be the *defender*. Consequently, the players on the other positions are the *attackers* of the current turn. The player on the last position is the first attacker, the player on the position after the defender is the second attacker and so on.

Events

E is the domain of events. E contains the following events, most of which have associated elements:

- *Begin*: A pseudo event that is not initiated by a player and whose sole purpose is to kick off a new round of Durak.
- *Attack*: An attacker adds cards to the field.
  - attacker id.
  - $cards \subseteq D$: The attacker cards.
- *Pass*: An attacker does not want to add another card to the field.
  - attacker id.
- *Defend*: A defender beats some cards on the field.
  - defender id.
  - $pairs \subseteq \{(a, d) \mid a, d \in D\}$: The defender cards are associated with the corresponding attacker cards.
- *Take*: A defender signifies that he cannot or will not defend the cards.
  - defender id.
- *Transfer*: A defender adds one or more suitable cards to the field and passes them on to the left neighbour.
  - defender id.
  - $cards \subseteq D$: The transfer cards.
- *Flash*: Similar to *Transfer* with the difference that the defender only has to show a suitable trump.
  - defender id.
  - $flashed \in D$: The flashed trump.

Of course, the events *Transfer* and *Flash* are only allowed in a game with the appropriate settings S.
We declare an *Attack* event with an empty set of cards to be a *Pass* event and a *Defend* event with an empty set of cards to be a *Take* event for convenience. In the following we will neither use *Pass* nor *Take* but instead *Attack* and *Defend* as described.

Phases

The phases are as follows:

- **Start**: The beginning of a new round of Durak.
- **Attack**: The attackers add cards to the field in a turn.
- **Defend**: The defender tries to beat the cards on the field in a turn.
- **Take**: The defender surrenders and takes all cards from the field into his hand. The attackers have the chance to add additional cards for the defender to take.
- **End**: The end of a round of Durak.

Rules

The rules of Durak are encoded in the function $\delta : S \times C \times E \rightarrow C \cup \{\perp\}$. δ receives the settings, a configuration and an event. It maps them to either the resulting configuration or $\perp$ in case of an inconsistent triple of arguments. In the following, a description of δ's operation in pseudo code will be presented by using the concepts and terms introduced in the sections above.

```
let handCards  = 6
let fieldPairs = 6
template defender = players.first

def δ(settings, cfg, event):
    if !checkValidity(settings, cfg, event): return ⊥
    cfg.event = event
    // In the following 'cfg.' or 'cfg' as an argument will be omitted
    case event:
        is Start:
           // Deal cards
           if durak != ⊥:
              // Durak will deal
              while players.first != durak: rotate players
           let max = min(#deck-1, #players*handCards)
           for i in 1..max: players[i%#players].hand.add deck.pop
           // Determine trump
           let card = deck.pop
           trump = card.suit
           deck.addTrump card
           // Determine starter if there was no previous durak
           if durak == ⊥:
              // Put starter at last index (first attacker).
              // No player had a trump? => Keep the order.
              let starter = players.withLowestTrump(trump)
              if starter != ⊥:
                 while players.last != starter: rotate players
          active = players.last
          phase  = Attack

        is Attack:
            if principal == ⊥: principal = event.attacker
            // Put cards on field
            event.attacker.hand.removeAll event.cards
            for card in event.cards: field.addAttacker card
            // If deck empty and lost all cards
            dismiss
            // Game may end here
            if finished:
               durak = defender
               phase = End
               return cfg
            // Last attacker?
            // (It must be >= because attacker could have already
            // been dismissed when he throwed, his hand became
            // empty and deck was empty, too)
            if attack# >= players.lastIndex:
               // Successful defense
               if field.allBeaten:
                  discard; deal; dismiss
                  // Game may end here (when only one player
                  // received cards and others did not)
                  if finished:
                     durak = defender
                     phase = End
                     return cfg
                  rotate players
                  active = players.last
                  phase  = Attack
              // Unfinished defense
              else:
                 // Defender surrendered
                 if phase == Take:
                    // Take cards on field
                    defender.hand.addAll field.allCards
                    clear field
                    deal; dismiss
                    // Game may end here (when others have no cards
                    // anymore and didnt get any from dealing)
                    if finished:
                       durak = defender
                       phase = End
                       return cfg
                    // Rotate twice so that surrenderer
                    // does not start the next attack
                    rotate players; rotate players
                    active = players.last
                    phase  = Attack
                 // Defender defended some but not all cards
                 else: // => phase == Attack
                    active = defender
                    phase  = Defense
                    attack# = 1
            // There are still attackers to go
            else:
               active = attacker(attack#)
               attack#++

        is Defend:
            // Surrender
            if event.pairs.empty:
               active = attacker(0)
               phase  = Take
               return cfg
            // Put cards on field
            event.defender.hand.removeAll event.pairs.defenderCards
            for a, d in event.pairs: field.defender[a] = d
            // Successful defense?
            if #event.pairs == #field.unbeaten and
                (#field == fieldPairs or event.defender.hand.empty):
                // Defender defended all cards
                // and no more cards can be added
                discard; deal; dismiss;
                rotate players
                // Game may end here (when defender used all cards
                // for defense at end but doesn't receive new)
                if finished:
                   durak = defender
                   phase = End
                   return cfg
                active = players.last
            else:
                // Defended only part or all
                // but new cards can still be added
                active = attacker(0)
            phase = Attack

        is Flash:
            rotate players
            active = defender
            phase  = Defense

        is Transfer:
            // Put cards on field
            event.defender.hand.removeAll event.cards
            for card in event.cards: cfg.field.addAttacker card
            rotate players
            active = defender
            phase  = Defense
            // If deck empty and transferred all cards
            dismiss
            // Game may end here (transferred last cards in hand)
            if finished:
               durak = active
               phase = End

    return cfg


// Helper functions

def discard(cfg):
    pile.addAll field.allCards
    clear field

def deal(cfg):
    attack#   = 1
    principal = ⊥
    for i in 0..players.lastIndex:
        let player = if lastIteration: defender else: attacker(i)
        let toDraw = min(#deck, max(0, handCards-#player.hand))
        player.hand.addAll deck.pop(toDraw)

def dismissPlayers(cfg):
    if deck.notEmpty: return
    for p in players:
        if p.hand.empty:
           players.remove p
           inactive.add p

def finished(cfg):
    deck.empty and ∀ a in attackers (a.hand.empty)

// Defender is at index 0 in players list
// First attacker is at last index (to the right of defender)
// First attacker is result of attacker(0)
// Second attacker is at index 1 (to the left of defender)
// Second attacker is result of attacker(1)
// Third attacker is at index 2 (to the left of second attacker)
// Third attacker is result of attacker(2)
// ...
def attacker(cfg, i):
    assert 0 <= i < #players-1
    // If cards were transferred back to principal attacker
    // (is defender now), just deal cards cw to attackers
    if principal == defender or principal in inactive:
       return players[i+1]
    if i == 0: return principal
    // Important because of flash or transfer
    var principalIndex = -1
    for k in 1..players.lastIndex:
        if players[k] == principal: principalIndex = k
    var j = principalIndex
    i.times:
        j++
        if j == #players: j = 1 // skip defender
    return players[j]
```

Having seen how δ works, i.e. how to go from one situation into the correct next situation, a representation of a game would simply be a list of configurations such that the first configuration is a start configuration, each next configuration is the result of applying δ to the previous configuration and its event, and the last configuration is an end configuration.

Comments

A collection of various comments that extend this specification with useful background information.

- It is assumed that the reader is familiar with the card game Durak and typical notational programming conventions.
- The main motivation for this specification was for it to serve as a reference (or simply as an idea how to model Durak precisely enough to be programmed) for implementations of the rules/logic of Durak as a computer program. Therefore, it is written in an operational style. It is also not a completely detailed specification but detailed enough for the reader to fill the gaps. Maybe a better description would be a "sketch of a specification" but that does not sound as nice.
- If it helps, Durak can be thought of as a state machine where the alphabet are the events and an accepted word is a sequence of events that starts with the *Begin* event and leads the machine into an accepting state, that is, into the phase End.
- Why the fuss about configurations? The introduction of the concept of configurations makes the state of the whole game explicit and captures it centrally. That means, for instance, that when having a list of configurations the corresponding game can be fully reconstruced. Furthermore, configurations make programming a Durak game logic much easier as implicit state is not scattered all over the program. Plus, testing the program resp. the logic is a lot easier with configurations as for any test a special "situation" can easily be constructed as a configuration and the desired output can, too, be encoded as a configuration. Which also makes it easy to implement some kind of interactive tutorial.
- Why a fixed order for attacks? This makes the game deterministic and far easier to implement. Just imagine adding cards to an attack would be handled like in real life. That is, apart from the first attack, attackers can add cards to the field as desired - it is a question of speed. Bots would need to have a delay before they would add cards (otherwise bots would be too fast) - but how long should the delay be etc.
- The function `checkValidity` would have a similar structure to δ's. For instance, it would check that an *Attack* Event only happens in the phases Attack or Take. Or it would check that a player does not throw more cards than allowed etc. It simply verifies preconditions, invariants and postconditions.
- To allow cheating as a game mechanic the validity checks of the function δ could be relaxed.
- For a useful user interface for Durak further "synthetic" events, for example "dealing cards", need to be derived by comparing the previous configuration with the current one.
- For a natural-language description of the game have a look at [2] or [3]. An optimized natural-language description where decisions for matters of preference have been matched to the semantics of δ is provided in appendix A.
- *True Fool* [1] is a Durak game whose game logic has been successfully implemented as described in this document.
- The versioning scheme is basically Semantic Versioning 2.0.0. In this document PATCH refers to linguistic fixes and layout fixes, MINOR refers to additions (e.g. additional explanations, rewordings) that do not change the semantics, and MAJOR refers to semantic restructurings of this document.

References

[1] http://www.reddit.com/r/truefool/

[2] http://www.pagat.com/beating/durak.html

[3] http://en.wikipedia.org/wiki/Durak

A. Natural-Language Description of Durak

This description is based mainly on [2, 3].

Setup

Durak is played with a deck of 36 or 52 cards and at least two players. The deck is shuffled, and each player receives six cards. The top card on the remaining deck is made visible. This determines the trump suit, however the revealed card is actually a part of the deck, the last card to be drawn. The player with the lowest trump is the first attacker in the first round. The goal of the game is to get rid of all one's cards at the end of the game.

Gameplay

The starting player is the first attacker. The player to the attacker's left is always the defender. After each turn play proceeds clockwise. If the attack succeeds, the defender loses his turn and the attack passes to the player on the defender's left. If the attack fails, the defender becomes the next attacker.
The order of attackers in a turn is as follows. The player who attacked first is the principal attacker and has the highest priority. The second attacker is the player to the defender's left. The third attacker is the player to the second attacker's left and so on. This order also determines who gets dealt cards first after a turn.
Aces are high. Trumps always beat non-trump cards regardless of rank (e.g., a trump 6 beats a non-trump ace).

Attack

The attacker opens the turn by playing one card (or several of the same face) face up on the table as an attacking card. The player to the attacker's left is the defender. The defender has to attempt defense in response to the initial attack.

Defense

The defender attempts to beat the attacking cards by playing defending cards from his hand. One card is played in defense of each attacking card. Non-trump attacking cards may be beaten by either a) a higher card of the same suit or b) a trump. Trump attacking cards may only be beaten by higher trumps. The defending cards are placed on top of the attacking cards so that players can keep track of which card is defending against which.
At any point during a defense, the attacker or any third party can pitch in extra attacking cards, provided that for each new attacking card, there is already a card of the same rank on the table (either defending or attacking), and the total number of attacking cards does not exceed six or the number of cards in the defender's hand, whichever is less. The defender must also defend against these new cards.
If the defender is unwilling or unable to beat all attacking cards, he must pick up all the cards on the table - including all the cards the attackers pitched in - and incorporate them into his hand. At this point, the defense is abandoned and the attackers may throw additional cards but without breaking the aforementioned constraints. The defender may choose to abandon the defense at any point during the turn. This immediately ends the turn. The failed defender loses his turn to attack; hence the player to the defender's left attacks next.
If, however, the defender has beaten all attacking cards, and no other players are willing or able to add more, the defender has triumphed. The turn ends, all cards on the table are discarded from play to a discard pile, and play passes to the left; the successful defender opens the next turn as the new attacker.

End of Turn

At the end of each turn, whether or not the defense was successful, the following action is performed; starting from the principal attacker, followed by anyone else who contributed cards, and culminating with the defender, each player with fewer than six cards in his hand must draw cards from the deck until he has six cards in his hand. When the deck runs out of cards, play simply carries on without any more cards being drawn. At this point, when someone runs out of cards, he is done with the game, and everyone else continues. Each player draws as many cards as he needs. The order in which this is done is strategically important since the last card in the deck is by definition a trump.

Winning and Losing

The last person with cards left in the hand is the loser (Durak). The person to the right of the Durak will begin the next round.

Variants

Transfer

In the *transfer* variant, the defender may choose to either attempt defense or to transfer the attack clockwise around the table. In this case, the defender may only transfer the attack if he has in his hand a card of the same rank as the attacking card or cards. To pass on the attack, he adds this card to the attacking cards. The defender now becomes the new attacker, and the player to his left becomes the new defender and must beat all cards. If the person on the player's left (the new defender) has fewer cards in his hand than will be on a table after transfering, transfering is not allowed. The new defender must then decide upon a response for this new attack. In games involving four or fewer players, it is perfectly possible for the attack to pass all the way around the table, so that the original attacker ends up defending against his own attack.

Flash

The idea is similar to *transfer*. However, instead of placing a card down besides your opponent's and sending it along the circle, you can simply *flash* your card with the same value instead of setting it down, if yours is a trump.