ZTCG:HS Documentation

Ronan C. P. Lana

March 12, 2017

Contents

1 About ZTCG:HS

ZTCG:HS (Zelda Trading Card Game: Hyrule Showdown) is a fan-made, non-profit intended, game engine which aims to emulate an environment for playing the card game of same name. The TCG have it's own rules, which will be explained with details eventually in the text.

While this Card Game contains some elements of it's own, one can perceive certain similarities in mechanics with other TCGs. Indeed, this TCG makes some homage to several existent TCGs within it's mechanics, game and card concepts: game setting from *Maplestory iTCG*®[1], concepts such as bonuses, modifiers, card lists from *Yu-Gi-Oh*®[2] and many others homages from other sources.

Disclaimer The original card game, that promoted the development of this game engine, contains a set of 200 cards based of some concepts of one of Nintendo®'s bestsellers *The Legend of Zelda*®*: Ocarina of Time*®. These concepts, such as names and images, were "borrowed" from them and so must be credited to their respective owners.

The main objective on this card game is to lead your faction to victory, by defeating your oponent using many cards and delving many strategies to make up the point. The way one builds their deck beforehand and in-match builds the ability tree and employs their cards is of vital importance to reach great performance in the game.

By collecting cards and managing deck builds between the matches, one can almost constantly improve their deck and try new powerful combinations or game experiences.

To further improve the gaming experience for the players, this game engine implements an unique feature called *custom cards*. With it players can design their own cards, with their own rules, to be played in the game. Furthermore, one can disable the original 200 card to use only their own cards.

[1] "MapleStory® iTCG®", published and developed by Nexon® and Wizards of the Coast®, PC, 2007.
[2] "Yu-Gi-Oh® Power of Chaos: Yugi the Destiny®", published and developed by Konami®, PC, 28 nov. 2003

2 Game Menus

ZTCG:HS have interactive menus for creating new profiles and decks, rearranging existent decks with cards gained through matches and setting up new matches when ready.

2.1 Main Menu

Figure 1: Main Menu interface.

The first menu in the game, from here you can opt between start a match, manage profiles or visualize cards collected by a profile.

2.2 Profiles

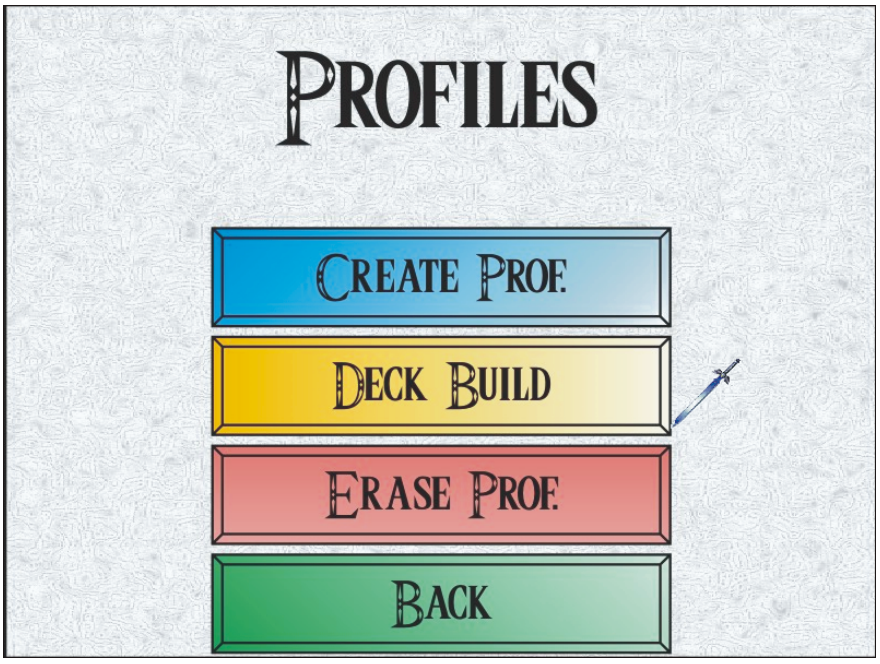


Figure 2: Manage Profiles interface.

This section guides the user to take 3 actions: create a new profile, draft a deck, or erase a existent profile. USE CAUTION when erasing a profile, deleted profile data is permanently destroyed!

2.2.1 Create Profile

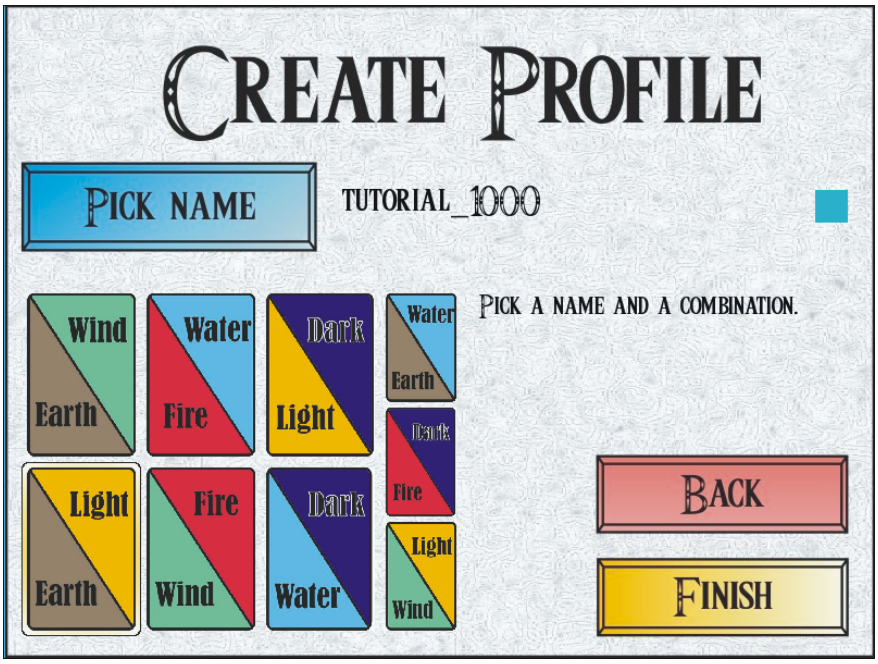


Figure 3: Create Profile interface.

Creating a new profile is simple, one must give an unique name and select one pair of elements which will define the initial deck class. Refer to the elements definition to get a glimpse about the available starter attributes.

To cancel this operation, hit Back.

Once the criteria has been matched and the Finish button has been pressed a full-fledged deck will then be generated, with cards given in such a way that card types and elements remains balanced within the deck.

Deck build:

2x Character, one for each element.

10x Mob of any element.

5x Cards from Type #1[3] of any element.

5x Cards from Type #2 of any element.

1x Field of any element.

14x Any card (except Character) of any element.

[3] Refer to elements for more information about element specialization.

2.2.2 Draft Profile

Figure 4: Draft Profile interface.

At this screen one can, with the cards earned through the matches played, move cards, make transactions and draft his/her deck to suit their needs. Let's take a look at the different Deck names the interface gives:

Main Deck The deck used to battle the opponents, there is a limit of 40 cards on this deck, and the number of copies of a card inside this deck is also limited, accordingly to the rarity of the card.[4]

Side Deck This place contains all the cards earned throughout played matches. Eventually one would like to swap best suited cards from the Side Deck to the Main Deck, or keep there unique cards as a collection. There are no restrictions to this deck.

Character Deck As the name suggests, only Characters are permitted on this deck. There are no limits here.

Shop One of the big features of this game, the Shop permits one to buy and sell desired cards. Cards present at the Side Deck can be sold easily, however at much lower price. Eventually, new cards appears at the shop to be bought, at prices varying accordingly to the user's profile level. Players

[4] Notice the Green box at the figure above: a GREEN box signalizes the swap from one deck to the other is permitted; a RED box signalizes prohibition, whether because the limit of 40 has been reached, or because the limit of copies of this card has been reached.

can find rare or high-leveled cards expensive at first (mainly if they try to buy it at lower levels), however such prices will become accessible over the level progression.[5]

To swap cards, left-click a card (it will appear to be selected afterwards) and hit Swap button. You can select one card of each side and hit Swap, causing a two-way swapping between be decks. Buying and selling cards also uses this mechanic.

Finally, having enough cards to do this process, one can set the game to autogenerate decks based-off the cards they have available. Use the Element Pick feature to filter which types you want this deck to be. Once the filters have been chosen, hit Generate Deck.

Generated deck build:

1x One copy of each character of any element.

2x Fields of any element.

10x Mobs of any element.

28x Any card (except Character) of any element.

Note that, if the maximum cannot be reached, the system will try to fill out the rest with any card of the filtered elements that are available, observing the limitation of copies.

After everything is done, hit SAVE to successfully commit the changes of the profile. Card swapping and transactions only will be updated after SAVE. Hit Cancel to exit without saving.

[5] Use caution, though Sold cards cannot be re-obtained, and if you are collecting cards, make sure you have ANOTHER of this card before selling and losing ownership.

2.2.3 Erase Profile

Figure 5: Erase Profile interface.

Use this screen to remove permanently profile data. Once deleted, there are no means to recover a profile, so use this thoughtfully.

2.3 Card List

Figure 6: Card List interface.

By browsing this section, one can easily see what cards a profile contains. The revealed cards shows that the selected profile contains at least one of this card at the Main Deck or the Side Deck.

Select a card to read it's description, by either left-pressing the rollbar buttons or moving the mouse wheel.

The cards are sorted by Type tabs, which can be toggled at the top of the screen. Inside a tab, cards are sorted by Element and finally alphabetically.

Once properly loaded, Custom Cards are too displayed on this screen.

Use Select Profile button to alternate between existent profiles and Quit to return to the Main Menu.

2.4 Play Game

Figure 7: Play Game interface.

Match set-up screen. Select 2 different profiles, with characters loaded, to start a match. When both profile sockets get the GREEN box, signalizing that they are ready, hit the Start Match button to initialize a game. Refer to the game rules for more detailed information about the match gimmicks.

Debug Mode This feature can be used to rapidly recover a state of a match. The first player defined at the "savestate.sav" file (inside folder "savestate" from the root directory) will *always* be the first to start in this mode. This file holds information about all deployed cards in a match, current cards on both hands and current HP and MaxHP of both players. Additionally to the savestate file, the "snapshot.sav" holds **volatile information about the table and player states** of the last turn played. In case of bug crashes use "snapshot.sav" (renaming it to "savestate.sav") to efficiently recover the turn and situation where the crash last happened.

3 Game Elements

3.1 Profile Features

A profile is a data structure which holds every information about a player's current deck. It holds things such as current Main Deck, Side Deck, Character Deck, profile Level & Exp, cash, and many other informations. After each match, values are updated accordingly to the player progression.

3.1.1 Decks

Refer to deck informations for details.

3.1.2 Level & Experience

Metrics that defines how well-confirmed a player is in the game. Higher levels means that the player already participated on many matches, and by that attained many privileges from playing, such as new cards at disposal and, consequently, better odds to have powerful deck combinations.

3.1.3 Rupees

In-game cash. Used for transactions with the Shop.

It is worthy to note that the game does some judgement about one's performance in a match, and rewards players with Exp and Rupees accordingly at the end of the match. At least one card is guaranteed for each player at the end of the match, and a booster pack when leveling up the profile.

3.2 Card Type

This attribute defines the nature of a card as well as it's behavior in the game.

Character: entity representing the player. Unit with the highest importance, if it dies the player loses the match. Grows in the match after learning new abilities as the turns goes on.

Mob: secondary entities which actively helps their owner once deployed. There is no need to maintain these units alive until the end of the match, however their constant presence ensures good performance in the match. Their unique trait is: if they remain alive at the start of the owner's turn, they fully recover their HP.

Equipment: tools which assist the character on the skirmishes. They cannot be directly targeted if there are mobs at the frontline, however given such space one can opt to target the equipment for minimal damage over targeting directly the character for a given damage.

Rapid Action: instant actions. Usually, these cards have great impact at the flow of the game, permitting even astounding roundabouts on a seemingly hopeless or so-called decided matches.

Field: card that gives some advantages to one party (owner's) or both parties (provided that the opponent of the field's owner does not placed a field yet). Can not be targeted by common means.

Jr.Boss & Boss: special mobs. These cards usually have a huge impact on the match once played, however they don't refresh HP every turn, like common mobs, but they do hoard considerable amount of HP. Only one Jr. Boss or Boss mob can stay deployed for each party at a time, and by no means two or more can stay at the same side at the same time.

3.3 Card Element

This attribute defines the elemental essence of a card.

 ***Earth*** Mastery: 

 ***Wind*** Mastery: 

 ***Fire*** Mastery: 

 ***Water*** Mastery: 

 ***Dark*** Mastery: 

 ***Light*** Mastery: 

In order to deploy a card in the match, your character must have at the build tree at least one card with the same element. In other words, to play a card one must have leveled up beforehand the card's element.

It is worthy to note that a multi-diversed deck can be of interest to the player. A deck having such traits can become potentially stronger, however one which uses too must elements becomes more of a hindrance than a boon. One must take it in consideration when drafting a deck.

3.4 Card Layout

Figure 8: Card details.

As depicted at the figure above, the greater portion of the cards of the game follows the presented design. Card level and type at the top-left corner, followed by the name and image of the card. Some stats (mob-based) and card informations at the middle area. And finally, at the bottom there is the level-up effect (except for Boss-type) and the element.

Card Name & Image Attributes that identifies the card in the game, for aesthetics purposes.

Card Level Value which defines how simple or tricky it is to call a card to the match. Usually higher-leveled cards are trickier to get played but, once deployed, have the potential to change the flow of the match.

Base Atk/HP Only on mob-type cards, it depicts the initial value of attack output and Health Points of a mob. These values can be changed during gameplay.

Card Info Conveys general information, such as the type of the card, subcategories[6] and other additional data. Card sub-types are classified using this string, and such filters can be used for some decisions in-match.[7] [8]

[6] Subcategories are used inside matches, isolating cards into niches (e.g. 'Weapon' equip, 'Recovery' tactic, 'NPC' mob).

[7] Example: a card that gives +20 HP for Mob cards that has sub-type "Mammal".

[8] Classification is held verifying if a queried sub-type is located somewhere inside the Card Info string of a card. If there is a matching, the card is of that sub-type. One can be of multiple sub-types as well.

Match Effects Defines rules for when a card is deployed and/or remains active on the field. Such effects are mitigated (does not activate or remains active) while the card is found under the state of silence.

Level-up Effects Defines rules that empowers the character's abilities. See build tree section for more information.

3.5 Build Tree

At the start of every turn, the player is given the opportunity to use up one card from his/her hand to level up the character. Every levelup requires one card from the player, and grants these bonuses (taking place first, card effects takes place right after these bonuses were given):

- Increases the current level by 10.
- Verify if HP + 20 surpass MaxHP value:
  - Increases HP and MaxHP by 40, if (HP + 20 >MaxHP).
  - Increases HP by 20, if not.
- Slide the card used to levelup to the bottom of the Build Tree (under the character).

If the card effect is a one-shot (), activate the effect immediately. This effect will not be activated during the character action phase.

What we can read from the image:

- Link is the character selected.
- Currently at level 30, Link will pass to 40 after leveling up with the sliding card.
- At level 30, he could use only his 3 natural skills (level 30, with 2 EARTH and 1 WIND attributes).
- Now he is level 40, with 2 EARTH and 2 WIND attributes.
- Now he can use Refresh and Stab abilities, in addition to his natural skills.
- Mercy was a One-shot skill, therefore it is simply ignored during Character Actions phase.
- Combo Hit requires the character to be at least level 80 and have at least 2 EARTH attributes.

Figure 9: Character ability tree.

3.6 Character Level and Attributes Learned

Level Represents the stage in which the character finds itself on the game. Depending on the level and the attributes learned through the Build Tree, some abilities can be unlocked.

Attributes Represents which elements the character unlocked and their respective level within the Build Tree.

3.7 HP & MaxHP

HP stands for both Health Points or Hit Points, and every unit in the game has both these values.

MaxHP Is the top value a unit can reach with it's HP. To have higher HP values one must improve the MaxHP first.

HP Current health value. If it reaches zero, the unit is destroyed and must be sent to the Graveyard. If a character reaches zero HP, the game is over and the one with HP higher than zero is the winner. If both characters reaches zero HP at the same time, a tie occurs.

3.8 Hand, Deck and Graveyard

Hand Set of cards visible only for the owner, on which one can analyze it and delve a strategy.

Deck Pile of cards disposed in such a way that none of the players can see their contents (cards face-down, and drawing from the to of the pile). One deck for each player.

Graveyard Pile of cards visible for both players, containing cards that were once used in the match and then discarded by the match rules. Each player has their own Graveyard.

3.9 Table Layout

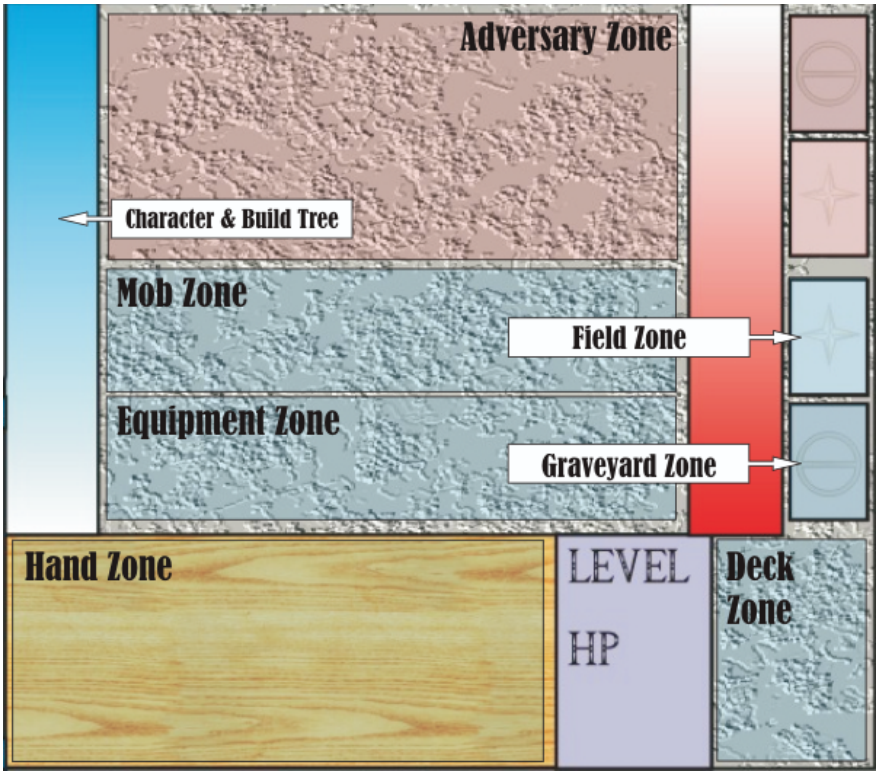


Figure 10: Table layout.

Your character is located at the left corner of the table, and immedately under the character is it's ability tree. Likewise, your opponent's character is located at the right corner with it's ability tree. At the center of the table, both parties shares the section, with the player using the bottom section and the opponent the top section.

The ZTCG table supports these numbers of concurrent cards for each side, separated by type:

7x Mob-like cards.

7x Equipment cards.

1x Field card.

1x Character card, filled from the start and remains there for all the match duration.

3.10 User Interface

The game's match UI implements several tools to help users interact with the card game. Be it for selecting a specific card in an array of cards, showing status of a deployed card in the table or showing an overview about the current character build, users have several tools at hands to support themselves.

Figure 11: Match screen.

From the match screen, one has overall information about the current state of the match:

- Whose player's turn.
- Number of cards on opponent's hand.
- Cards on player's hand.
- Level and HP of both characters.
- Both player deck's card left count.
- Current build tree of both players.
- Deployed cards of both players.

- Graveyard list – clicking the top card of the Discard Pile, a list of cards is displayed.
- Short message (yellow box) that shows the current expected action of a player.
- Tabs at the top section that further improves a player understanding of the current state of the match.

The left section shows detailed information of the last card that has been hovered with the mouse, by displaying a big version of the target card picture and box-rollable card text.

There are 3 tabs at the top-right corner, each one for a single interactive (but not changes the match state) purpose:

Build Displays graphical info about what kind of cards can be played by the specified player at the current state of the match. Displays max level of a card-type that can be deployed at the current character level (in color) and when all abilities are available (in grey). Additionally, on a player's turn, it is possible to overview max levels of card types *for the current turn* in a distinguished color. One can better plan their actions (such as what card to level up with next turn) with this tool. Accessible pressing F1.

Match Log Displays history of the last actions and their results in the match. Accessible pressing F2.

Info Displays additional information about both player's current performance in the match (overall damage dealt, overall healing, etc.). Accessible pressing F3.

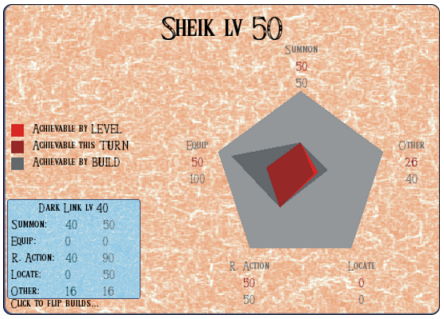


Figure 12: Build UI.

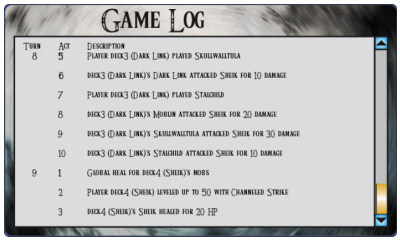


Figure 13: Match Log UI.

Status Box Deployed cards can change it's state too dinamically through actions in a turn. In order to maintain track about some deployed card status, the system offers the Status Box funcionality. It covers:

Figure 14: Build UI.

- Card type and name.
- Current card HP / MaxHP (always ceiling 40).
- If *thinking an attack*: base (blue) and togglable (green) damage to this card.
- If *character card*: profile and build informations.
- Card status (receiving field bonus, stunned, silenced, hijacked, ...).

4 Game Rules

4.1 Match Setup

To start a match, both players must have chosen a Character card and a deck to use up for all the match duration:

- Set the Character face-up on it's place.
- Place the deck, with cards face-down, at the Deck pile location.
- Decide who starts the game. The first to go draws 5 cards and the second draws 6.

Both Characters starts with their HP and MaxHP values equals to the depicted by the card design, at level zero, with no cards under their charaters.

4.2 End of a match

- The game ends when a character reaches zero HP, losing the match.
- Whoever remains alive at the end is declared the winner of the match.
- The match still goes on if the deck has been completely depleted.

4.3 Turn Phases

Like the most TCGs, this one is too turn-based. Said that, players must alternate control of the game between themselves sequentially, in order to receive feedback of the other side's actions and take proper decisions and countermeasures.

Every turn begins, follows sequentially a set of steps or objectives, and then finishes, passing control for the opponent to resume their game. Once the opponent finishes their actions, the control returns to the player, and so goes on unitl the end of the match. Each step of this set of actions is also called a Turn Phase.

4.3.1 Start phase

Takes place even before the Level-up phase. Here, no player command an action, only effects that takes place at the "start of the turn" may be unfolded. Also, the player's deployed mob-types (not Jr. Bosses or Bosses) refreshes their HP to the fullest at this stage.

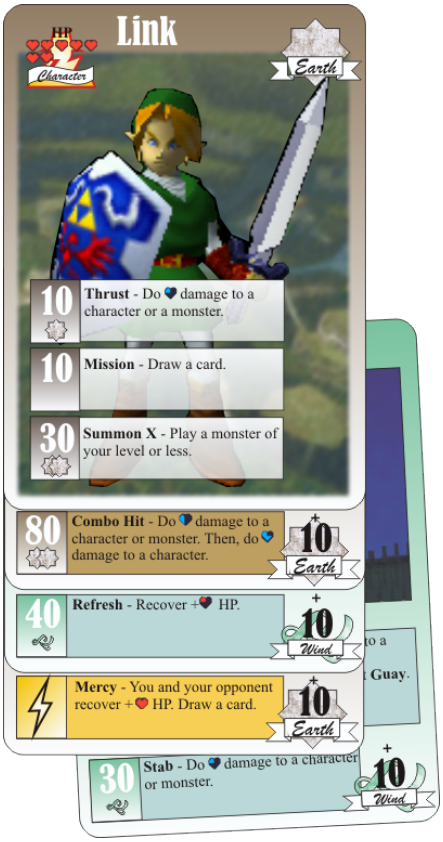
4.3.2 Level-up phase

At this step, the player decides if they use a card from the hand to level up their character. Opting to level up a character brings many benefits for the player: improving MaxHP, unlocking new abilities, etc.

4.3.3 Onset phase

Again, the player does not make commands here. Effects listed to be used "after level-up phase" are played here.

4.3.4 Character Actions phase



The character abilities are to be read and played sequentially, from the top to the bottom. Taking as example this 2 EARTH, 2 WIND, level 40 Link, the following actions will be played this turn:

- Thrust
- Mission
- Summon X – read as Summon 40 in this case.
- Refresh
- Stab

Figure 15: Character ability tree.

4.3.5 Mob Actions phase

At this step, the player does not make commands. Every mob that does not specify a taget phase for it's actions, but their action is not deemed as a "continuous" effect (such as aura or attack prevention) defaults to play it's action here.

4.3.6 Mob Attack phase

Finally, it time for the mobs to help the player. Every mob deployed by the player can use their base attack value as damage output for an opponent character or mob. This right can be overriden by the match rules, alternatively a unit cannot attack at all if it is under the effect of stun.

4.4 Game Mechanics

4.4.1 Attack instances

Counts towards attack instances every command of attack in the game. They always have a targeting unit, but can have or not a source unit (e.g. DOT). They define too some flags that can modify some behaviors and interactions in the game:

- Prevent – Offers the target the opportunity to **wear out** an attack instance without receiving any damage.
- Block – Offers the target the opportunity to **lower** the damage received by an attack instance.
- Counter – Offers the target the opportunity to create a new attack instance as **retaliation** to being targeted.

Every attack instance that does not define flags are essentially preventable, blockable and counterable.

Sometimes, units receive boons that permits them prevent attacks a defined set of times. In these cases it is possible to stack preventions from multiple sources, however **those from a same source overwrite the predecessor** with the new value.[9]

4.4.2 Damage Calculation

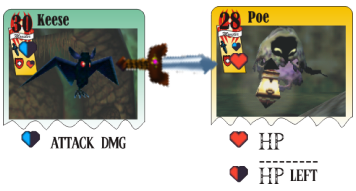


Figure 16: Attack calculation.

- No modifiers (auras, damage buffs, blocks) are being considered in this example.
- Keese has 20 attack damage, and Poe has 40 HP.
- Keese (Mob) declares attack on Poe (Mob).
- Poe will end up with 20 HP left. Keese used it's attack.

Damage instances on ZTCG:HS are attributed to source-types. This means that, if a equipment effect generates an attack instance, this attack comes FROM an equipment. Moreover, there can be disjointed attacks (e.g. DOT effects) that, although the DOT did come from an unit (let's say a Mob-type), it does not count towards a Mob attack but more like an "unknown" source attack; therefore, disjointed.

Similar to source-types, target-types are taken in consideration when evaluating an attack instance.

[9] Say a generic CHARACTER can prevent 2 attacks as a boon from MOB1 and 3 attacks as a boon from MOB2. In this case, CHARACTER can prevent up to 5 attacks. Now, MOB2 activates a new effect that permits CHARACTER to prevent 2 attacks. So, CHARACTER now can prevent 2 from from both mobs, making up to 4 attack preventions (in contrast to 5 before).

Notice that modifiers can come from a wide pletora of card effects throughout the game, in the shape of auras, one-turn effects (rapid actions), equipment effects, and much other sources.

One must observe these criteria when calculating the final damage output:

- Minimal damage dealt is zero.
- Maximal damage dealt is the current HP of the target.

Special traits for certain circumstances:

- Damaging a character

Figure 17: Attack calculation on characters.

  - No modifiers (auras, damage buffs, blocks) are being considered in this example.
  - Keese has 20 attack damage, and Sheik has 2 Mobs deployed.
  - Keese (Mob) declares attack on Sheik (Character).
  - Each poe blocks 10 damage for Sheik, but does not suffer HP loss.
  - Sheik remains unharmed. Keese used it's attack.

- Fixed Attack

  Attack instances flagged as a fixed attack does not work the same way as common attacks. They do identifies types and elements for source and target like common attacks, however **the damage output is deemed constant** (hence no buffs or nerfs are applied). Generally, they can be prevented or counterattacked.

- Directed Attack

  Work in the same fashion as a fixed attack, except that the target will always receive the attack. In other words, they can not be prevented by any means.

4.4.3 Stacking effects

As a TCG, unless stating otherwise, essentially every card in the game can have it's effects stacked by any means. From auras to damage amplifiers, this is a nice feature for permitting awesome combos of card effects.

Exception to this rule is the number of turns left for the effects of stuns and silences. In this case, the number of turns is the greater value between the current value and the applicable.

4.4.4 Equipment targeting

In this TCG, equipments makes a major role in supporting a character with their effects. Due to this, one may find useful to **get rid of the adversary gadgets** before engaging their character.

Like characters or mobs, equipments too have HP (or durability if you feel like it). When the HP reaches zero, the equipment is destroyed (removed from the table and placed in the Graveyard).

Calculating an equipment base HP:

$$20 + \frac{card\_level}{2} \tag{1}$$

Attacking an equipment has a different trait than attacking a character or a mob. While for characters or mobs one must calculate the final value of the damage, an attack to an equipment always deals (fixed) 10 damage, and acts like a directed attack.

- Equipments can be targeted whenever the opponent has no deployed mobs.
- Only attack instances that targets characters can target equipments too.
- Deals fixed 10 damage.
- Unpreventable, unblockable and unconterable.

Targeting an equipment does not damage the character, as the equipment is treated as an individual target.

4.4.5 Specific-Target Attack

Some cards specify in their card effects some *target-oriented* attack pattern (e. g. "to all opponent mobs and character"). In this case, a player has absolutely **no options** about whether they will commit this attack or not (no "thinking" or "passing" an attack) or, if a character is the target, they cannot opt to target an equipment instead. Every attack instance perceived as a "specific-target" will, within it's directive, **instantly commit an attack** at the very target specified. It's damage, however, can vary accordingly with the bonus present in-match.

4.5 Card Mechanics

4.5.1 Aura

Aura effect is defined as a continuous effect and lingers around while it's source remains deployed and it is not under effect of silence. They can appear as many flavors, like bonus attributes, protections and of other natures.

Cards are not destroyed by auras changes during the match, rather stays with minimal HP (10) under these circumstances.

4.5.2 Stun

Card affected with stun *cannot create attack instances*. In other words, unable to attack. Stunning an unit already stunned will refresh the count[a], if applied. Effect cannot be purged and wears out:

- Over time in case of turn-based register applied.
- The source of the effect is destroyed in cases where there is no turn count.

[a] Whichever is the *greater* between the current countdown and the applied value will prevail as the next countdown value.

4.5.3 Silence

Card affected with silence *cannot activate their in-game card effects*. Card effects becomes active again after lifting the silence effect. Silencing an unit already silenced will refresh the count[a], if applied. Effect cannot be purged and wears out:

- Over time in case of turn-based register applied.
- The source of the effect is destroyed in cases where there is no turn count.

[a] Whichever is the *greater* between the current countdown and the applied value will prevail as the next countdown value.

4.5.4 Damage-Over-Turn (DOT)

Cards affected with DOT *keep losing HP every turn*, while the DOT effect is active. One card can receive multiple DOT instances, however DOTs from same source are not stackable (already existent registry is immediately overwritten). Effect cannot be purged and wears out:

- Over time in case of turn-based register applied.
- The source of the effect is destroyed in cases where there is no turn count.

DOT effect takes place at the start of the opponent's turn.

4.5.5 Hijack

Cards affected with hijack are meant to *change sides* (becomes controlled by the opponent) immediately. The target card uses a slot of the opponent side and frees the owner's slot. However, given impossibilities when moving sides (due to already having a Jr.Boss or Boss present or no slots available), one of these actions are taken accordingly to who is the card's owner:

- Changing to opponent's side – card is destroyed and sent to the owner's Graveyard.
- Changing to owner's side – card returns to the owner's Hand.

Hijacking an unit already hijacked[a] will refresh the count[b], if applied. Effect cannot be purged and wears out:

- Over time in case of turn-based register applied.
- The source of the effect is destroyed in cases where there is no turn count.

[a] Trying to take back a hijacked unit by hijacking it **WILL NOT work**. The hijack countdown will be merely refreshed. It happens because, starting a hijacking process, every card **returns** to their owner's side, the process is executed, and then hijacked cards experience change sides again.

[b] Whichever is the *greater* between the current countdown and the applied value will prevail as the next countdown value.

4.5.6 Withdraw character

Characters affected with withdraw are *unable to create attack instances and can not be targeted by new attack instances*. Still takes damage from indirect attacks, such as DOT. Effect wears out over time. Multiples withdraw effects from same source are not stackable and, like preventions and DOTs, they are immediately overwritten when done again.

4.5.7 Draw Card

Permits the player to draw one card from the top of the Deck to their Hand. Does nothing if the Deck is empty.

4.5.8 Discard Card

Forces the player to choose and discard a card from their Hand to the Graveyard. Does nothing if the Hand is empty.

4.5.9 Peek Card

Permits the player to see the next card of their Deck (does not reveal to the opponent, except if it is needed for validation purposes), then return it to the top of the Deck face-down. Does nothing if Deck is empty.

4.5.10 Reveal Card

Turns public (visible for both players) a target card from one's Hand or Deck.

4.5.11 Deposit Card

Cards in-game can cointain other cards under them, similar to the build tree under a Character, as result of their effects. Cards "deposited" are not counted towards deployed, therefore their effects are not applicable yet in the game.

Whenever a deployed card containing deposited cards is destroyed or removed from the field, the deposited cards are sent back to their owner's Decks, at the bottom of the pile.

If a deployed card containing deposited cards changes side, all cards under changes side too (following the hijacked card).

4.5.12 Level Dropping

Similar to gradually increasing one's character level to unlock the potential to play high-leveled cards, a specific card's level can also be tinkered around to better suit their owner's needs.

Card affected with some kind of level changing effect will act in the game as it was of that situational level, instead of the one depicted in the card image. It affects mainly:

- Playing it regarding level restrictions.
- Accounting it's level when executing some card rules.

Note: Card levels **cannot go under zero**. Therefore, cards which level drop surpasses it's base level will ultimately be at level 0.

As it is with auras or DOTs, level dropping effects also **cannot be stacked by the same source card**, although from many sources these effects can.

5 Development Section

5.1 Custom Cards

ZTCG:HS implements a simple and intuitive way to add new cards to the game. Essentially, to create a new custom card one needs to:

- Firstly, *define a name* for the card to be used in the file system. Say it's card name is "My Card", so it's file system name should be "my_card" (all lower-case, underline for spaces);
- *Design an image* for the card using the template provided. 240 x 360 pixels, PNG file. Set it inside the folder: "./cards/pics/custom" and with name "<filesystem-name >.png";
- Open "metadata.txt" at "./cards" and *feed the text file* **appending** a new line containing "<filesystem-name >.lua". Note that there **must** be an empty line at the end of this file;
- Create a card descriptor file, which will contain *rules and definitions* for your card, inside "./cards/pics/info". Name this *text file* "<filesystem-name >.lua".

At first, creating a card descriptor file may seem a little overwhelming, however it is actually quite reasonable. One can alternatively start designing from a template, that can be obtained from "./cards/info/examples". There are available templates for all card types.

The custom card parser has been designed in such a fashion that users struggles less in properly adapting their code to feed the parser and focus more in properly elaborating their card rules. By proportionating flexibility of code and well-defined syntax structures, even those who never coded in Lua or never coded at all can create their own cards without dropping a sweat.

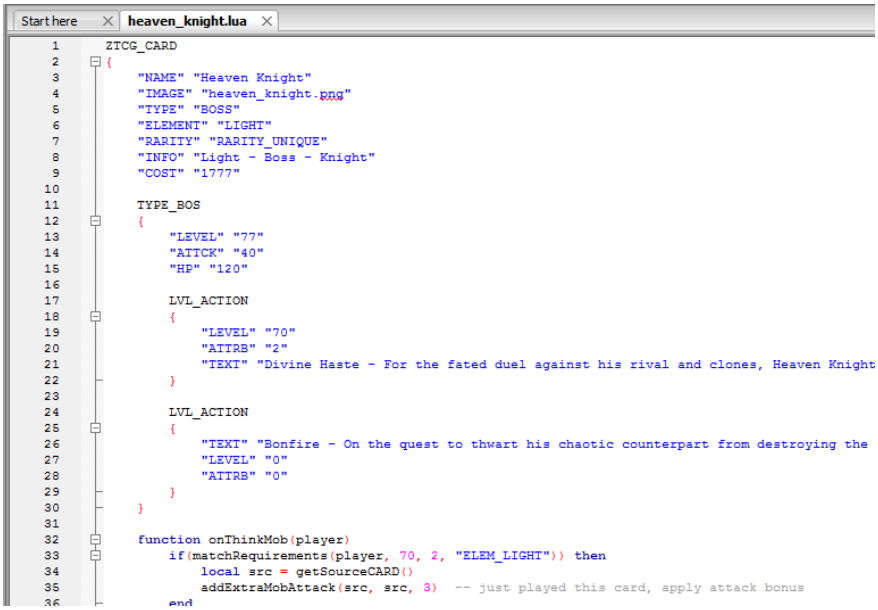


Figure 18: Card descriptor for a boss-type custom card.

Take the figure above as reference, reproduced as is from the custom card "heaven_knight.lua". From there, one can easily grasp the fundamentals about creating custom card descriptors:

- ZTCG_CARD is the name of the structure representing a **card object**. Every custom card defines one, and it holds all information about the card we want to design.
- Every card attribute for the designed card is defined **between quotes**.
- Basic informations (such as card name or rarity) are instantiated by a **key-value** syntax.
- Depending of the card type, an unique **internal structure** needs to be called. As of the example, we want to create a Boss card, so it is needed to define a TYPE_BOS structure. This structure must be a singleton[10].
- The parser is flexible in such a way that we can declare two LVL_ACTION's in non-sequential sections of the code, except that it MUST be located inside the TYPE_BOS structure.
- Functions are defined in the lowest section of the code, still inside ZTCG_CARD. Indeed, no piece of code should be defined outside of ZTCG_CARD, as it is the scope of the whole file.

One can get tricked into thinking that, because it uses a LUA extension, it is thoroughly a LUA script file. Wrong. The only LUA-worth code snippet here is the one defined inside functions, that **must** match the LUA language definitions. In other words, these descriptor files acts more like *hybrid-Lua*

[10] Singleton structures must occur once and only once in the code.

files. It gets the flavor of the flexibility when defining basic informations for new custom cards (like name or rarity) and of the powerfulness of the Lua script engine when defining rules for these cards.

Structures of the card descriptor:

- ZTCG_CARD – outer block, defines the scope for the custom card descriptor.
- TYPE_* – inner block, defines the scope for particular data of the card type.
- LVL_ACTION – inner block that either defines match effect informations for Characters or level-up effects for non-Characters. Additionally, nests inside TYPE_JRB — TYPE_BOS to define match effects for Jr.Bosses and Bosses.
- function – actual Lua functions, defines rules for the custom card.
- key-value line combo – defines custom card's basic information.

Block attributes marked with a * are *optionals* and, therefore, can be left undefined at the card descriptor file.

5.1.1 ZTCG_CARD

Fundamental block. Carries all data information for a custom card. This block **requires** instantiation of certains sub-blocks, as defined below, except those regarded as *optionals*:

Basic information Attributes and warranted value-types:

- NAME – String.
- IMAGE – file name of the card image.
- TYPE – ZTCG_TYPE context.
- ELEMENT – ZTCG_ELEMENT context.
- RARITY – ZTCG_RARITY context.
- INFO – INFO String.
- COST – Integer value.

Card Type **Required.** Inner block of a defined card type.

Level action Optional. Inner block of level action.

Lua-functions Optional. Those blocks of functions defines the essence of the custom cards, with such functions being what actually maintains the in-match effects of the cards.

It is recommended to firstly fill in the basic informations for the card, then fill the block for the targeted type of card, and finally define the card rules (Lua-functions).

5.1.2 LVL_ACTION

Block that defines an action for Characters, Jr. Bosses and Bosses.

- LEVEL – Integer value OR "ZTCG_MAXVALUE" (for one-shot actions). *Required character level to use ability.*
- ATTRB – Integer value. *Required element level to use ability.*
- TEXT – String. *Information of the ability.*

5.1.3 TYPE_CHA

Block that defines essential attributes for a Character.

- HP – Integer value. *Character's base HP.*
- LVL_ACTION – Block. *Character action #1.*
- LVL_ACTION – Block. *Character action #2.*
- LVL_ACTION – Block. *Character action #3.*

5.1.4 TYPE_MOB

Block that defines essential attributes for a Mob.

- LEVEL – Integer value. *Mob's level.*
- ATTCK – Integer value. *Mob's base attack.*
- HP – Integer value. *Mob's base HP.*
- * TEXT – String. *Mob's match effects.*

5.1.5 TYPE_EQP

Block that defines essential attributes for an Equipment.

- LEVEL – Integer value. *Equipment's level.*
- TEXT – String. *Equipment's match effects.*

5.1.6 TYPE_ACT

Block that defines essential attributes for a Rapid Action.

- LEVEL – Integer value. *Rapid Action's level.*
- TEXT – String. *Rapid Action's match effects.*

5.1.7 TYPE_FLD

Block that defines essential attributes for a Field.

- * BLOCK – Integer value. *Defines the increase on defense block for characters favored by the field.*
- TEXT – String. *Field's match effects.*

5.1.8 TYPE_JRB

Block that defines essential attributes for a Jr. Boss.

- LEVEL – Integer value. *Jr. Boss' level.*
- ATTCK – Integer value. *Jr. Boss' base attack.*
- HP – Integer value. *Jr. Boss' base HP.*
- * LVL_ACTION – Block. *Jr. Boss' match effects #1.*
- * LVL_ACTION – Block. *Jr. Boss' match effects #2.*

5.1.9 TYPE_BOS

Block that defines essential attributes for a Boss.

- LEVEL – Integer value. *Boss' level.*
- ATTCK – Integer value. *Boss' base attack.*
- HP – Integer value. *Boss' base HP.*
- * LVL_ACTION – Block. *Boss' match effects #1.*
- * LVL_ACTION – Block. *Boss' match effects #2.*

5.2 Flags

Flags can be defined as *key values* (or integer values) employed in situations where the system must try to make decisions and judges accordingly with the current match situation and the evaluation of the current atomic rules.

For example, say there is a rule stating that only *mobs* of *level 40 or lesser* can be played *from the hand* at this command. The command is **playing a card**, however only cards **from hand**, **MOB-typed**, **level 40-** must be played. There were used flags for FROM ("hand"), TYPE, LEVEL and COMPARATOR ("less than").

Flags are sorted accordingly with the **context** they are given, and functions representing atomic rules will frequently draw references to these flags[11], so custom card designers needs to stay sharp when using these flags. Use this tutorial as reference and you shall be ok.

It is even possible to **combine non-mutually excludent flags** using "—". For example: you want to call a Mob or Jr.Boss card to play. From "ZTCG_TYPE" context you have TYPE_MOB and TYPE_JRB. So, your final flag layout shall be: "TYPE_MOB — TYPE_JRB", meaning that **both** mobs and Jr. Bosses are eligible for this action.

[11] they will say what kind of flag they need with "ZTCG_*".

5.2.1 ZTCG_DONTCARE

Wildcard flag. Represents that a rule's argument value is considered *irrelevant* or the *defaulted* value[12] should be used.

5.2.2 ZTCG_NIL

Wildcard string. Represents that a rule's argument string is an **empty string**.

5.2.3 ZTCG_CARDID

Represents the identity of a card registered in the game. Rather than identifying a card in a match, where there can exist many copies of the same card, this kind of key represents the identity of an unique card in the system. Each *different* card receives one different key value. To discover key of cards, one must call atomic rules passing card objects as parameters.

5.2.4 ZTCG_RARITY

Defines rarity for custom cards. This flag can be entered at the field "RARITY" of custom card's basic informations structure.

- RARITY_COMMON – Creates rarity definition indicating the odds to find this card is quite **trivial** and the max number of copies in a deck is **4**.
- RARITY_SELDOM – Creates rarity definition indicating the odds to find this card is somewhat **unusual** and the max number of copies in a deck is **2**.
- RARITY_UNIQUE – Creates rarity definition indicating the odds to find this card is the **hardest** and the max number of copies in a deck is **1**.

5.2.5 ZTCG_TYPE

Categorizes *filters* for a specific combination of card types or *specify* a type. Filters can be combined.

- TYPE_MOB – Creates filter or type definition for Mob.
- TYPE_EQP – Creates filter or type definition for Equipment.
- TYPE_ACT – Creates filter or type definition for Rapid Action.
- TYPE_FLD – Creates filter or type definition for Field.
- TYPE_CHAR – Creates filter or type definition for Character.
- TYPE_JRB – Creates filter or type definition for Jr. Boss.
- TYPE_BOS – Creates filter or type definition for Boss.
- TYPE_ANYMOB – Creates a filter for Mob, Jr. Boss and Boss.
- TYPE_ANY – Creates a filter for any card type.

[12] Zero in most of the cases.

- LVL_ACTION – Creates a type definition for a Character Action.

5.2.6 ZTCG_ELEMENT

Categorizes *filters* for a specific combination of card elements or *specify* an element. Filters can be combined.

- ELEMENT_EARTH – Creates filter or element definition for Earth.
- ELEMENT_WIND – Creates filter or element definition for Wind.
- ELEMENT_FIRE – Creates filter or element definition for Fire.
- ELEMENT_WATER – Creates filter or element definition for Water.
- ELEMENT_DARK – Creates filter or element definition for Dark.
- ELEMENT_LIGHT – Creates filter or element definition for Light.
- ELEMENT_ANY – Creates filter for any element.

5.2.7 ZTCG_COLORID

Categorizes a *color* to be used within a card rule.

- COLOR_BLUE – Blue color.
- COLOR_RED – Red color.
- COLOR_GREEN – Green color.
- COLOR_GOLD – Gold color.
- COLOR_SILVER – Silver color.
- COLOR_WHITE – White color.
- COLOR_BLACK – Black color.

5.2.8 ZTCG_TABLESLOTID

Maps a slot in the table. "Player" stands for the rule's card owner and "Adversary" for their adversary.

- SLOT_PLAYERCHAR – Player's Character table slot.
- SLOT_PLAYERMOB1 – Player's Mob #1 table slot.
- SLOT_PLAYERMOB2 – Player's Mob #2 table slot.
- SLOT_PLAYERMOB3 – Player's Mob #3 table slot.
- SLOT_PLAYERMOB4 – Player's Mob #4 table slot.
- SLOT_PLAYERMOB5 – Player's Mob #5 table slot.
- SLOT_PLAYERMOB6 – Player's Mob #6 table slot.
- SLOT_PLAYERMOB7 – Player's Mob #7 table slot.
- SLOT_PLAYEREQP1 – Player's Equipment #1 table slot.
- SLOT_PLAYEREQP2 – Player's Equipment #2 table slot.
- SLOT_PLAYEREQP3 – Player's Equipment #3 table slot.

- SLOT_PLAYEREQP4 – Player's Equipment #4 table slot.
- SLOT_PLAYEREQP5 – Player's Equipment #5 table slot.
- SLOT_PLAYEREQP6 – Player's Equipment #6 table slot.
- SLOT_PLAYEREQP7 – Player's Equipment #7 table slot.
- SLOT_PLAYERFLD – Player's Field table slot.
- SLOT_ADVSRYCHAR – Adversary's Character table slot.
- SLOT_ADVSRYMOB1 – Adversary's Mob #1 table slot.
- SLOT_ADVSRYMOB2 – Adversary's Mob #2 table slot.
- SLOT_ADVSRYMOB3 – Adversary's Mob #3 table slot.
- SLOT_ADVSRYMOB4 – Adversary's Mob #4 table slot.
- SLOT_ADVSRYMOB5 – Adversary's Mob #5 table slot.
- SLOT_ADVSRYMOB6 – Adversary's Mob #6 table slot.
- SLOT_ADVSRYMOB7 – Adversary's Mob #7 table slot.
- SLOT_ADVSRYEQP1 – Adversary's Equipment #1 table slot.
- SLOT_ADVSRYEQP2 – Adversary's Equipment #2 table slot.
- SLOT_ADVSRYEQP3 – Adversary's Equipment #3 table slot.
- SLOT_ADVSRYEQP4 – Adversary's Equipment #4 table slot.
- SLOT_ADVSRYEQP5 – Adversary's Equipment #5 table slot.
- SLOT_ADVSRYEQP6 – Adversary's Equipment #6 table slot.
- SLOT_ADVSRYEQP7 – Adversary's Equipment #7 table slot.
- SLOT_ADVSRYFLD – Adversary's Field table slot.

5.2.9 ZTCG_FLAGTYPE

Categorizes which *attribute* of a card is intended to be used.

- FLAG_TYPE – Intended to use Type-flag.
- FLAG_ELEM – Intended to use Element-flag.

5.2.10 ZTCG_COMPARATOR

Categorizes a comparator.

- LESSER – Intended to filter equal or lower values.
- GREATER – Intended to filter equal or higher values.

5.2.11 ZTCG_PLAYERTYPE

Selects between current card's owner or adversary.

- IS_PLAYER – Select current owner.
- IS_ADVERSARY – Select current owner's opponent.

5.2.12 ZTCG_PLAYERMODE

Categorizes that a game rule will be applied to a specific player.

- ONLY_ADVSRY – Applies only to the **adversary** of the player that called the card rule.
- ONLY_PLAYER – Applies only to the **player** that called the card rule.
- ANY_PLAYER – Applies to **both players**.

5.2.13 ZTCG_PREVENTTYPE

Applies a new instance of Prevent damage for designated types of units in the match. Characters always benefits from this.

- PREVENT_ANY – *Characters* and *Mobs* can prevent next attacks.
- PREVENT_CHARONLY – *Only Characters* can prevent next attacks.

5.2.14 ZTCG_RECOVERTYPE

Sets which option or filter of cards should be used when recovering a card by a game rule. Filters can be combined.

- RECOVER_ANY – any card is eligible to be selected for recovery.
- RECOVER_MOB – only Mob, Jr.Boss and Boss-typed cards are eligible for recovery.
- RECOVER_EQP – only Equipment-typed cards are eligible for recovery.
- RECOVER_ACT – only Rapid Action-typed cards are eligible for recovery.
- RECOVER_FLD – only Field-typed cards are eligible for recovery.
- RECOVER_TOP – only the card at the top of the targeted Graveyard is eligible for recovery. Does nothing if there is no card.

5.2.15 ZTCG_ATKSRC

Sets the type of the unit initiating an attack instance for the requiring game rule. When setting an attack source one must make use of *common sense*, applying appropriate attack sources for the attacker's card type, except if the used atomic rule states otherwise.

- ATKSRC_MOB – Signals a Mob, Jr. Boss or Boss card declared the attack.
- ATKSRC_CHA – Signals a Character card declared the attack.
- ATKSRC_ACT – Signals a Rapid Action card declared the attack.
- ATKSRC_EQP – Signals an Equipment card declared the attack.

- ATKSRC_NIL – Signals an "unknown" or anonymous source declared the attack. [13]

5.2.16 ZTCG_ATKRES

Sets restrictions for an attack instance.

- ATKRES_NIL – Signals an attack instance *without restrictions*, targeting both Characters and Mobs.
- ATKRES_DONT_HIT_CHAR – Signals an attack instance that targets *only Mobs*.
- ATKRES_DONT_HIT_MOBS – Signals an attack instance that targets *only Characters*.
- ATKRES_FIXED_SLOT – Signals an attack instance that targets an unit in a *specific table slot*. Rule **requires additional ZTCG_FIXEDTARGET** flag.
- ATKRES_FIXED_SLOT_DAMAGE – Signals an attack instance that targets an unit in a *specific table slot* and deals *unchangeable* damage value (ignores damage calculations). Rule **requires additional ZTCG_FIXEDTARGET** flag.
- ATKRES_FIXED_DAMAGE – Signals an attack instance that deals *unchangeable* damage value (ignores damage calculations).

5.2.17 ZTCG_FIXEDTARGET

Sets a target Character or Mob table slot of a specified player.

- SLOT_CHAR – Character table slot.
- SLOT_MOB1 – Mob #1 table slot.
- SLOT_MOB2 – Mob #2 table slot.
- SLOT_MOB3 – Mob #3 table slot.
- SLOT_MOB4 – Mob #4 table slot.
- SLOT_MOB5 – Mob #5 table slot.
- SLOT_MOB6 – Mob #6 table slot.
- SLOT_MOB7 – Mob #7 table slot.

5.2.18 ZTCG_TRUESTRIKE

Sets visibility criteria for targeting a card for an attack instance.

- STRIKE_NORMAL – Cards are allowed to try to evade an attack.
- STRIKE_TRUE – Cards **will be targeted** regardless possible evasion.

[13] E.g. Attacks from DOT effects.

5.2.19 ZTCG_PREVENT

Sets availability for prevention in an attack instance rule.

- DISABLE_PREVENT – Disables opportunity for prevention.
- ENABLE_PREVENT – Enables opportunity for prevention.

5.2.20 ZTCG_COUNTER

Sets availability for counter in an attack instance rule.

- IS_STARTER – Signalizes availability for counterattacks.
- IS_COUNTER – Signalizes either that it is already a counterattack or counterattacks are disabled.

5.2.21 ZTCG_GLOBALMODE

Used within atomic rules that involve broadcasted attack instances. In other words, attack atomic rules that targets multiples units. Flags can be combined.

- GLOBAL_HITCHAR – targets Character.
- GLOBAL_HITMOBS – targets all deployed Mobs.

5.2.22 ZTCG_AURAMODE

Applies broadcasted bonuses of attack, HP, and damage block auras for deployed units in the game.

- GLOBALAURA_PASS_PLAYER – Does not affect card's current owner.
- GLOBALAURA_PASS_ADVSRY – Does not affect adversary of the card's current owner.
- GLOBALAURA_BOTH_PLAYERS – Affects both players.

5.2.23 ZTCG_BLOCKAURA

Selects a list from a deployed unit's "status list".

- AURA_STUN – Selects the Stun list.
- AURA_SILENCE – Selects the Silence list.
- AURA_HIJACK – Selects the Hijack list.

5.2.24 ZTCG_DECKTYPE

Selects the kind of deck structure from the targeted player structure.

- DECK_HAND – Selects the player's Hand.
- DECK_DECK – Selects the player's Deck.
- DECK_GRAV – Selects the player's Graveyard.
- DECK_CARD – Does nothing.

5.2.25 ZTCG_DECKORIENT

Selects a side of a target deck structure.

- DECK_TOP – Selects the top entry of the target deck.
- DECK_BOTTOM – Selects the bottom entry of the target deck.

5.2.26 ZTCG_DECKMOVE

Defines some commands related to target deck management.

- TAKE_BYNAME – Takes the first card with the name passed as argument from the target deck.
- TAKE_CARDID – Takes the card represented by the id passed as argument from the target deck.
- TAKE_NEXT – Takes the top N cards from the target deck, with N an integer given as argument.
- PUT_TOP – Puts the defined card or set of cards at the top of the target deck.
- PUT_BOTTOM – Puts the defined card or set of cards at the bottom of the target deck.

5.2.27 ZTCG_PEEK

Decides whether to show or not the targeted list of cards.

- CARDLIST_HIDE – Does not reveal cards of the list.
- CARDLIST_PEEK – Reveals cards of the list.

5.2.28 ZTCG_FILTER

Decides whether to apply filtering or not on a defined set of card.

- NO_FILTER – Does not apply filtering on the list.
- USE_FILTER – Apply filtering on the list. Rule **requires** several additional informations.

5.2.29 ZTCG_SUMMONMODE

Modes for summoning a Mob, Jr. Boss or Boss card.

- PLAY_NORMALSUMMON – Default summon, will ask for a card to play.
- PLAY_SCOUTSUMMON – Will try to summon the last card in the Hand, regarding elements learned and current character level. Fails and does nothing if incompatibilities occurs or character is of lower level than the applied card or character did not learn this card's element yet.

- PLAY_FORCESUMMON – Will try to summon the last card in the Hand, ignoring elements learned and current character level. Does nothing if it fails due to card-type incompatibility (is not a mob-type) or other situations.

5.2.30 ZTCG_EQUIPMODE

Modes for equipping an Equipment card.

- PLAY_NORMALEQUIP – Default equip, will ask for a card to play.
- PLAY_GEARUPEQUIP – Will ask to play equipments of one of these sub-types: Weapon, Armor or Shield.

5.2.31 ZTCG_LOCATEMODE

Modes for locating a Field card.

- PLAY_NORMALFIELD – Default locate, will ask for a card to play.
- PLAY_FIRSTCARDFIELD – Will try to locate the first card in the Hand. Does nothing if it is not a Field-typed card.

5.2.32 ZTCG_PLAYCARDMODE

Essence of a "play card"-based character action, one can define with this context which card types are allowed to be played through an action. Only the **defaulted versions** for Summon, Equip and Locate are available from this context. Flags can be combined.

- PLAY_MOB – Supports playing a Mob, Jr. Boss or Boss-typed card.
- PLAY_EQUIP – Supports playing an Equipment-typed card.
- PLAY_ACTION – Supports playing a Rapid Action-typed card.
- PLAY_FIELD – Supports playing a Field-typed card.

5.2.33 ZTCG_CALL

Appoints to a character action slot. Intention is re-call a specific character action of a character or, defining a target card, use "CALL_ACTION" to call it's Character Action effect.

- CALL_ACTION – Action from the build tree.
- CALL_CHAR_ACTION1 – Character's Action #1.
- CALL_CHAR_ACTION2 – Character's Action #2.
- CALL_CHAR_ACTION3 – Character's Action #3.

5.2.34 Other ZTCG Flags

There are yet many other usable flags within the program. These flag contexts, however, are not employed as arguments for atomic rules, but rather as input signals received from pointcuts of the system. Input signals

comes from many flavors, be it ZTCG Flags (as defined in this section), card *id*'s[14] or mere booleans and integers representing something.

The main purpose of these flags is to report the current game state inside a pointcut. The card designer is free to do whatever they want with these informations within their card rules, as long as they use them consistently with the way they are designed for.

- ZTCG_SBOXTYPE

  Returns the card type of the current card referenced by the Status Box.

  - SBOX_MOB – Current target is a Mob, Jr. Boss or Boss-type.
  - SBOX_CHAR – Current target is a Character-type.
  - SBOX_EQUIP – Current target is an Equipment-type.

- ZTCG_TARGETTYPE

  Returns the type of the card that is currently being targeted by an attack. (Equipments are not evaluated for this flag.)

  - ATTACKED_CHAR – The target is a Character.
  - ATTACKED_MOB – The target is a Mob, Jr. Boss or Boss.

- ZTCG_ATTACKSTATE

  Determines whether the current player already declared an attack instance or is in thought process.

  - THINK_ATTACK – Attacking player is **pondering** about an attack.
  - APPLY_ATTACK – Attacking player **decided** to attack.

5.2.35 ZTCG_REMOVETYPE

Determines the nature of a removed card of the game. Card removal is considered the act of moving a card that was not on the Graveyard to it.

- IS_DESTROYED – Card was deployed and got removed. (Table)
- IS_DISCARDED – Card was not deployed and got removed. (Hand or Deck)

- ZTCG_GAMEHUBTYPE - *Deprecated.*

[14] An *id* is defined as an object descriptor of a card in a match.

Similar to ZTCG_PLAYERTYPE, however in this case the player is choosed by their Hub id, that is defined at the start of the game.

  - IS_P1 – Selects Player #1.
  - IS_P2 – Selects Player #2.

- ZTCG_DRAWMODE

  Specifies which kind of drawing will be applied to the next Effect.

  - DRAW_SOURCE – Will apply no tranformations on the given sprite.
  - DRAW_SCALED – Will apply only scale transformations on the given sprite.
  - DRAW_TINTED – Will apply only tint transformations on the given sprite.
  - DRAW_ROTATED – Will apply only rotate transformations on the given sprite.
  - DRAW_COMPLEX – Will apply all kind of transformations on the given sprite.

- ZTCG_DRAWFLAG

  Specifies flipping transformations on the next Effect.

  - FLIP_NONE – Applies no effect on the given sprite.
  - FLIP_HORIZONTAL – Flips horizontally the given sprite.
  - FLIP_VERTICAL – Flips vertically the given sprite.
  - FLIP_HORIZONTAL_VERTICAL – Flips horizontally and vertically the given sprite.

- ZTCG_DRAWBLEND

  Specifies blending transformations (color combination between sprite image and the one already on the screen) on the next Effect.

  - BLEND_ADD – Adds components.
  - BLEND_SRC_MINUS_DEST – Subtracts components from screen with the sprite's.
  - BLEND_DEST_MINUS_SRC – Subtracts components from sprite's with the screen.

- ZTCG_BLENDMODE

Specifies how components from an item (screen or sprite) will behave in the blending operation.

- BLEND_ZERO – No effect.
- BLEND_ONE – True RGBA.
- BLEND_ALPHA – Only A.
- BLEND_INVERSE_ALPHA – Only (255-A).
- BLEND_SRC_COLOR – Only components from source affect the composition.
- BLEND_DEST_COLOR – Only components from destination affect the composition.
- BLEND_INVERSE_SRC_COLOR – Only inverse of components from source affects the composition.
- BLEND_INVERSE_DEST_COLOR – Only inverse of components from destination affects the composition.

5.3 Custom Card Rules

Card rules are a set of functions defined by the card designer to represent some cards effects within the game. To understand how card effects should be implemented, one must comprehend the idea behind how the program interfaces two symbolic pillars of this TCG engine:

- Pointcuts – **Event-based sections** of the system where custom cards are received opportunities to **interact** with the main game engine. In other words, pointcuts are event entries in the game where the custom cards can execute their commands (atomic functions and/or other codes) to emulate card effects.
- Atomic Rules – Indivisible and *unique commands*, made simple in order to offer the best performance to the card designer, freeing the latter from the necessity of developing low-level code to supply their demands. There are over 132 unique rules to tinker around.

The Custom Card sytem has been developed thinking ahead almost every situation an user would like to implement in this TCG engine, so there are *no real need to implement low-level code* at this stage of the development. Pointcuts and atomic rules have been layed out in such a fashion that one only needs to *study the documentation* of these concepts to come with an idea for their custom cards.

5.3.1 Atomic Rules

These are the basic commands of the game. Commands such as "draw card", "attack opponent's mob", "refresh character's HP" are brought into action after the designer draw **references** to them in their card codes. Technically, these rules calls **assigned functions** from within the engine to execute what they are supposed to do.

These functions can not be changed by the card designer, however the way these functions are approached to satisfy the needs of the designer can be obtained via the use of game flags. Each atomic rule has it's own set of flags that can be used to modify the way the function reacts.

Use caution when calling those rules though, as they must follow a certain **pattern** to work properly. **Data types of arguments** that are to be given to these functions consist of:

- Boolean (*bool*) – Raw boolean value. *True* or *false*.
- Integer (*int*, *short*) – Raw integer value. Without quotes.
- String (*string*, *ZTCG_STRING*) – Raw sequence of letters. Within quotes.
- Wildcard-type (*WILD*) – Can come in form of Integer or String, at the discretion of the user.
- Player Id (*PLAYER*) – Pointcut flag value representing whose player's turn. It is a *boolean*: when *true* Player #1 is controlling the turn and when *false*, naturally, Player #2.
- Card Object (*CARD*) – In-game object reference of a card in the game. Must be retrieved by calling an atomic rule referencing a card in the game, and stored in a variable.
- Deck Object (*DECK*) – In-game object reference of a deck-type[15] list in the game. Must be retrieved by calling an atomic rule referencing the deck-type list, and stored in a variable.
- List/Target Object (*LIST*, *TARGET*) – In-game object reference of a generic sequence of cards in the game. Must be retrieved by calling an atomic rule referencing the card sequence, and stored in a variable.[16]
- Coordinates List Object (*COORDLIST*) – In-game object reference of a list of coordinates in the game. Must be retrieved by calling an atomic rule referencing the coordinate sequence, and stored in a variable.
- Flags (*ZTCG_<*>*) – Defines the way an atomic rule should behave when called. Argument is passable in both String form and Integer form, preferred the first.

Memory management Note that the system lets you manage *lists*. It is of vital interest of developers and designers to deal with lists to help them reach their coding goals, and here it is definitely not diferent! Said that, the game engine **"borrows" for you some memory** from the operating system to deal with lists and returns it when you are done. However, the same way you calls the engine to request some memory for your lists, **you should let the engine know you will not be using these lists anymore** and therefore want to return it. It is done using special atomic rules

[15] Hand, Deck or Graveyard.

[16] Although List and Target representations are of the **same data-type**, the intention of a List is to hold information of a *sequence of cards* while the intention of a Target is to appoint to a *specific card* in a sequence of cards.

(for LIST and COORDLIST structures) which sole purpose is return the lists' used memory for you (which is done automatically!). Note that **even empty lists** uses memory, therefore they too must be freed.

Automated memory management If the paragraph above made you worried because of the need to remember to free things everytime a list is used, relax! Even if the card designer forgot somewhere within their code to free the blocks of memory used by their lists, at the end of every match the system will automatically detect *unfreed* lists and get rid of them for you. No problem!

When using atomic rules, remember to **acknowledge** the signature of these functions passing up the arguments with of the right data type and in the right order, to prevent erroneous crashes or unpredicted behaviors. **Also important:** Whenever in the atomic function signatures a variable name appears at both **return values side** and **argument values side**, the variable holding the argument value MUST be referenced at the return slot with same name to overwrite the now-inconsistent value.

Notice that the "PLAYER" argument is commonly referred is these functions. This argument is generally related the **actor** of an action, many times related to the one which is playing that very turn. One can discover which player is playing that turn using the sole argument of the pointcuts: "player". Generally, the one playing the turn the pointcut is being referenced is the actor of an action, except when said otherwise. The argument "PLAYER" is essentially a boolean, so whenever it is need to refer the *adversary* to the one playing this turn, just *negate* the "PLAYER" argument in the code.[17]

The **signature** of all atomic rules are listed in the subsections below:

- ***DECK Object Descriptor***

  Object descriptor functions for manipulating DECK-type.

- *DECK card_list =* ***getPlayerDeck****(PLAYER, ZTCG_DECKTYPE)*

  Returns a DECK object descriptor.

- ***LIST Object Descriptor***

[17] In other words, having an argument "PLAYER" that was given by the pointcut, it is just right to write "not PLAYER" in Lua-code to refer the adversary of the one playing the current turn.

Object descriptor functions for manipulating LIST-type.

- *LIST card_list, int qty =* ***getListFromDeck****(DECK deck)*

  Returns a LIST object descriptor from a DECK capsule.

- *int list_length =* ***getListLength****(LIST card_list)*

  Returns the length of a card list.

- *bool empty_list =* ***isEmptyList****(LIST list)*

  Verifies emptiness of the list.

- *LIST list1 =* ***appendLists****(LIST list1,LIST list2)*

  Sequentially unifies two lists. The end of the first list links with the start of the second. "list1" variable **must be refreshed**[18], as the data turned inconsistent. "list2" became part of "list1", therefore it **must not** be freed (when freeing "list1" "list2" goes as well).

- ***destroyList****(LIST list)*

  Frees LIST list structure.

- ***CARD Object Descriptor***

  Object descriptor functions for manipulating CARD-type.

- *CARD board_card =* ***getCARD****(TARGET deck)*

[18] The same variable given as argument for "list1" is to be assigned at the return-side to receive the new list.

Gets a card object descriptor from a TARGET capsule.

- *CARD card =* ***getOnBoardCARD****(PLAYER, ZTCG_TABLESLOTID)*

  Gets a card object descriptor from a position in the table.

- *CARD target =* ***getSourceCARD****()*

  Gets a card object descriptor of the very card which code is being currently run.

- *CARD target =* ***getTargetCARD****()*

  Gets a card object descriptor of the card that has been branded as *target* by the current pointcut.

- *bool is_null =* ***isNullCARD****(CARD card)*

  Tests if the given card is Null.

- *bool is_same =* ***isSameCARD****(CARD card1, CARD card2)*

  Tests if the two cards object descriptors refers to the same card instance in the game. Two card instances in the game can have the same "unique card id" (given by the system to enumerate different cards), yet they are **different** card object descriptors.

- *int slot =* ***getSlotIdFromCARD****(PLAYER, CARD card)*

  Gets the localization of "card" in the PLAYER's table. Returns -1 upon failure.

- ***COORDLIST Object Descriptor***

Object descriptor functions for manipulating COORDLIST-type.

- *COORDLIST coord_list =* ***getTableCardCoordinates****(ZTCG_PLAYERMODE, short int min_atrib,int min_lv,int max_lv,ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring, PLAYER)*

  Creates a list structure which holds coordinates of every deployed card filtered by these filters.

- ***destroyTableCardCoordinatesList****(COORDLIST coord_list)*

  Frees COORDLIST list structure.

- *CARD card =* ***selectGameTableCardFromCoordinatesList****(string msg,COORDLIST coord_list,PLAYER)*

  Prompts "PLAYER" to *select* a deployed card's game object descriptor in the given list.

- ***DECK, LIST and CARD manipulation***

  Additional functions that manages several system operations between these structures.

- *LIST new_list, bool not_empty =* ***makeFilteredList****(PLAYER,LIST list,short int min_atrib,int min_lv,int max_lv,ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring)*

  Creates a list structure which holds copies of the card object descriptors that exist in "list" and were matched by the given filters.

- *LIST new_list, bool not_empty =* ***makeFilteredTableList****(PLAYER, ZTCG_PLAYERMODE,short int min_atrib,int min_lv,int max_lv,ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring)*

Creates a list structure which holds copies of the card object descriptors that were deployed in the game and were matched by the given filters.

- *TARGET card =* ***menuCards****(PLAYER,LIST list,string msg,ZTCG_PEEK)*

  Selects a card object between these listed in the "list" input. Returns an iterator of the given list.

- ***pickCardOrder****(PLAYER,LIST list)*

  Prompts the player to *select a ordering* for the cards present in "list". Order goes from LEFT to RIGHT (as TOP to BOTTOM of the list, respectively).

- *bool ret =* ***hasCardOnDeck****(ZTCG_CARDID,DECK deck)*

  Returns *existence of at least one copy* of the given unique card key in "deck".

- *bool discarded =* ***discardRandomCardFromDeck****(PLAYER,DECK deck)*

  Automatically discards one random card on given "deck", whose owner is the given player. Does nothing if "deck" is empty.

- *bool discarded, LIST deck =* ***discardRandomCardFromList****(PLAYER,LIST list)*

  Automatically discards one random card on given "list", whose owner is the given player. Does nothing if "list" is empty.

- *LIST card =* ***takeTargetCardFromDeck****(TARGET card,DECK from)*

  Creates a list structure which holds a copy of the target "card" supposed to be present in the given deck. Empty structure if the card was not found on given deck.

- *LIST cards =* ***takeCardsFromDeck****(DECK deck, int qty)*

  Creates a list structure holding up to the number of cards that were pulled from the top of the deck. Empty structure if no cards were found on given deck.

- *LIST from, bool not_null =* ***takeTargetCardFromList****(TARGET card,LIST from)*

  Simply pull off the target "card" from list "from". **"from" must be refreshed**. Note that after pulling "card", if nothing is done about "card" (such as passing to another list or some table slot), it's information is lost because it currently does not exist in any concrete structure in the game.

- *LIST out_list =* ***takeTargetCardFromListToDeck****(DECK deck,LIST out_list,TARGET card,ZTCG_DECKORIENT)*

  Pulls "card" from "out_list" and immediately puts the card object into a "deck" at the top or bottom side.

- ***moveCards****(DECK from,DECK to,ZTCG_DECKMOVE take,ZTCG_DECKMOVE put,int val)*

  Transactions cards between two decks. A card or a sequence of cards is taken from "from" using the first ZTCG_DECKMOVE flag (**TAKE_***) and put on the deck "to" using the second ZTCG_DECKMOVE flag (**PUT_***). The input that must be inserted in "val" argument pairs with the **TAKE_*** argument, being it:

  - An unique card id key[19] – TAKE_NAME.
  - The very card object descriptor (TARGET) – TAKE_CARDID.
  - The next "val" cards – TAKE_NEXT.

- *LIST from =* ***moveCardsFromListToDeck****(LIST from,DECK to,ZTCG_DECKMOV*
  take,ZTCG_DECKMOVE put,int val)

  Transactions cards between a list to a deck. A card or a sequence of cards is taken from "from" list using the first ZTCG_DECKMOVE flag (**TAKE_***) and put on the deck "to" using the second ZTCG_DECKMOVE

[19] Each different card receives an unique key by the system.

flag (**PUT_****). The input that must be inserted in "val" argument pairs with the **TAKE_**** argument, being it:

- An unique card id key – TAKE_NAME.
- The very card object descriptor (TARGET) – TAKE_CARDID.
- The next "val" cards – TAKE_NEXT.

- ***editCardRegister****(CARD card, ZTCG_CARDID,int slot,short int val,short int signal, ZTCG_POINTCUTID)*

  Overwrites the "slot"-th register of a source card id key ZTCG_CARDID of a given "card". Each card can use **up to 8** (slots 0 to 7) different registers. Besides holding records of a card in the match, card registers *can be used as reactors* inside pointcuts, creating possibilities such as enabling new functions after a specific event has occured. Registers permits use of both "val" and "signal" as customizable variables for the card designer to explore.

- *short int val, short int signal =* ***getCardRegister****(CARD card, ZTCG_CARDID,int slot)*

  Gets the current "val" and "signal" attributes from a "card" 's register, located at the "slot"-th position of the source card's unique id "ZTCG_CARDID". Gets nothing if the designated register has not been declared yet.

- ***displayCardRegister****(CARD card, ZTCG_CARDID,int slot,int font_sz, ZTCG_COLORID)*

  Shows a number in front of the "card" in-game image representing the current value under "val" inside the designated register ("slot"-th register of source "ZTCG_CARDID"). Customizable font size and color.

- ***CARD Object Attributes***

  Retrieve several informations from a CARD object.

- *int level =* ***getBaseLevelFromCARD****(CARD card)*

  Get base level of a CARD.

- *int level =* ***getCurrentLevelFromCARD****(PLAYER, CARD card)*

  Get CARD's current level on the match.

- *int basehp =* ***getBaseHpFromCARD****(CARD card)*

  Get base HP of a CARD.

- *int maxhp =* ***getMaxHpFromCARD****(CARD card)*

  Get max HP of a CARD.

- *int hp =* ***getHpFromCARD****(CARD card)*

  Get current HP of a CARD.

- *int att =* ***getBaseAttackFromCARD****(CARD card)*

  Get base attack damage of a CARD.

- *int rarity =* ***getRarityFromCARD****(CARD card)*

  Get rarity of a CARD.

- *ZTCG_STRING name =* ***getNameFromCARD****(CARD card)*

  Get name of a CARD.

- *int type =* ***getTypeFromCARD****(CARD card)*

Get type of a CARD.

- *int element =* ***getElementFromCARD****(CARD card)*

  Get element of a CARD.

- *int cardid =* ***getCardIdFromCARD****(CARD card)*

  Get the unique key of a CARD. Each different card in the system has a different key value, and copies of the same card holds the same Card Id.

- *ZTCG_STRING info =* ***getCardInfoFromCARD****(CARD card)*

  Get general information (sub-types, common location, etc.) of a CARD.

- ***Meta Data***

  About flag manipulations and records of card Registers.

- bool ret = ***hasSharedFlags****(ZTCG_TYPE (flag), ZTCG_TYPE (flag))*

  Compares bit-wise the flags, checking for a position where both bits are set to True. Returns *True* on success.

- bool ret = ***hasSharedFlagsCARD****(CARD card, ZTCG_FLAGTYPE, ZTCG_TYPE (flag))*

  Compares bit-wise one of the attributes of a card with of the given flag, checking for a position where both bits are set to True. Returns *True* on success. Attribues checked are or Type or Element.

- *ZTCG_TYPE val =* ***decodeZtcgFlag****(ZTCG_TYPE (flag))*

Return the position of the first bit which contents in a bit set to True. Value returned will cover the same type of the flag provided.

- *CARD card =* ***getCardPointer****(int slot)*

  Returns the current card object descriptor present at the given card pointer slot. Slot ranges from 0 to 3.

- ***setCardPointer****(int slot, CARD card)*

  Assigns a new card object descriptor to the card pointer slot. Slot ranges from 0 to 3.

- *int val =* ***getGameValue****(int slot)*

  Returns the current value being hold by the game engine at the given slot.

- ***updateGameValue****(int pos, int new_value (ZTCG_ID))*

  Assigns a new value to the given slot. This value can be anything but a string or other complex-typed data (like object descriptors).

- ***Sliding Cards Mechanics***

  Set of functions specialized in dealing with associated cards of a deployed card in the table.

- *LIST card_under =* ***putCardUnder****(CARD card,LIST card_under)*

  Takes the *first card* in the "card_under" list and slides it under "card". Does nothing if list is empty.

- *LIST ret =* ***takeCardUnder****(CARD card,TARGET target)*

Fetches for the "target" card under "card", taking it from there and **creates a new list** with the taken card.

- ***returnCardsUnderToDeck****(PLAYER,CARD card)*

  Returns all the card under "card" to their respective owner's deck.

- *LIST ret =* ***removeCardsUnder****(CARD card)*

  Creates a list with all the cards slided under "card", ultimately removing them from there.

- *TARGET ret =* ***getCardUnder****(CARD card,int slot)*

  Returns a card object descriptor of the "slot"-th card slided under "card".

- ***Table Mechanics***

  Table-oriented functions specialized into retrieving informations about the current situation of the table of the match.

- *int mobs =* ***getMobsOnTable****(PLAYER,ZTCG_PLAYERMODE)*

  Gets the number of mobs present at PLAYER's side.

- *int eqps =* ***getEquipsOnTable****(PLAYER,ZTCG_PLAYERMODE)*

  Gets the number of equipments present at PLAYER's side.

- *bool has_fld =* ***isFieldOnPlayerTable****(PLAYER)*

Verifies if a PLAYER has a Field deployed.

- *bool has_boss =* ***isBossOnPlayerTable****(PLAYER)*

  Verifies if a PLAYER has a Jr. Boss or Boss deployed.

- *bool res =* ***playerHasCardIdDeployed****(PLAYER, ZTCG_CARDID)*

  Verifies if a PLAYER has a *non-silenced* "ZTCG_CARDID" card deployed.

- *int res =* ***playerCountCardIdDeployed****(PLAYER, ZTCG_CARDID)*

  Counts the quantity of cards *not silenced* with unique id "ZTCG_CARDID" at PLAYER's table.

- *int ret =* ***nextFreeMobSlot****(PLAYER)*

  Returns the first empty mob slot on a PLAYER's table. Returns -1 on not finding an empty slot.

- *int ret =* ***nextFreeEquipSlot****(PLAYER)*

  Returns the first empty equipment slot on a PLAYER's table. Returns -1 on not finding an empty slot.

- ***UI & Game State Mechanics***

  Multi-purpose functions that acts supporting UI needs or some match state checkings.

- *int attr =* ***getPlayerAttributes****(PLAYER, ZTCG_ELEMENT)*

Returns the current element level of PLAYER for the given "ZTCG_ELEMENT".

- *bool match =* ***matchRequirements****(PLAYER,int level,short int qty, ZTCG_ELEMENT (flag))*

  Checks if PLAYER has enough character level and minimal element attributes for every "ZTCG_ELEMENT" defined inside the flag.

- ***refreshScreen****()*

  Commands the system to *redraw current screen* buffer.

- ***refreshBoard****(PLAYER)*

  Commands the system to *renew the current screen buffer* with the standard informations to the game and redraw the screen.

- ***displayMessageOnScreen****(PLAYER, ZTCG_STRING str, int msec)*

  Commands the system to pop a message at the text area of the UI for the given time.

- *bool ret =* ***makePrompt****(PLAYER,string text,string subtxt, WILD const1,WILD const2,string opt1,string opt2)*

  Creates a prompt screen to PLAYER, requesting them to pick one of the two given actions. "Text" introduces the action that generated this prompt and some context. "subtxt" displays additional information, such as values stored by variables, by using masks[20]. Up to 2 values can be shown this way.

- ***Passive Game Mechanics***

[20] Anywhere in subtxt: write %[d, s] to request an integer or a string, respectively. E.g.: "Card name %s, HP %d" will require 2 arguments: string "const1", integer "const2". Entry order matters.

Mechanics related to applying some game-changing effects in the game that is *not directly related* to employing a player command (attacking, drawing a card, shuffling the deck, etc.).

- ***insertTimedAura****(PLAYER, int cardid, int turns)*

  Timed aura instance is defined as a feature that a player inherits and uses through the course of a match. Always associated with an unique card id (makes no distinctions within copies of the same card), a new instance is generated by using this function, and it wears out at the end of the turn count.

- *int count =* ***countTimedAura****(PLAYER, int cardid)*

  Counts the current number of timed auras of the given card id a player has. **Alone, timed auras does nothing**; it is how it is used along some piece of code that defines the unique behavior of a timed aura.

- ***newBuff****(CARD card,CARD source,int atk,int hp,int def,short turns)*

  Instantiates a new buff instance on "card". Buffs that comes from same "source" are updated instead of instantiating a new one.

- ***updateBuff****(CARD card,CARD source,int incAtk,int incHp,int incDef)*

  Updates existent buff from "source". Does nothing if fails to find "source".

- ***removeBuff****(CARD card,CARD source,int atk,int hp,int def)*

  Removes an existent buff, even before it's turn count reaches zero. If "source" is not defined, one must match both atk, hp and def values to eliminate a buff instance. Otherwise, if a buff from "source" is found on "card", it is removed regardless of those 3 values.

- ***applyTargetBonus****(PLAYER,CARD card,CARD src,int atk,int hp,int def,short min_atrib,int min_lv,int max_lv,ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring)*

Applies a buff to a targeted "card" **only if** it matches with all given filters.

- ***removeTargetBonus****(PLAYER,CARD card,CARD src,int atk,int hp,int def,short min_atrib,int min_lv,int max_lv,ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring)*

  Removes a buff from a targeted "card" **only if** it matches with all given filters.

- ***applyAuraBonus****(PLAYER, ZTCG_AURAMODE,CARD src,int atk,int hp,int def,short min_atrib,int min_lv,int max_lv,ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring)*

  Applies a buff to all deployed cards that meets "ZTCG_AURAMODE" recognition and **matches** with all given filters.

- ***removeAuraBonus****(PLAYER, ZTCG_AURAMODE,CARD src,int atk,int hp,int def,short min_atrib,int min_lv,int max_lv,ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring)*

  Removes a buff from all deployed cards that meets "ZTCG_AURAMODE" recognition and **matches** with all given filters.

- *bool res =* ***hasBlockAura****(CARD target, ZTCG_BLOCKAURA, CARD src)*

  Checks if "target" has an active block aura instance (stun, silence or hijack **which source is a deployed "src" card**) from "ZTCG_BLOCKAURA" list.

- ***applyBlockAura****(CARD target, ZTCG_BLOCKAURA, CARD src)*

  Registers a new block aura instance (stun, silence or hijack **which source is a deployed "src" card**) to the defined "ZTCG_BLOCKAURA" list. Can create duplicated registries.

- ***removeBlockAura****(CARD target, ZTCG_BLOCKAURA, CARD src)*

Unregisters an active block aura instance (stun, silence or hijack **which source is a deployed "src" card**) from the defined "ZTCG_BLOCKAURA" list.

- ***setFieldRegister****(PLAYER, int newVal)*

  Sets a new value for a PLAYER's field register (that represents how many blocks a field can support the favored character in a turn).

- ***sendPlayerAway****(PLAYER, CARD src, ZTCG_PLAYERTYPE, int turns)*

  Renders PLAYER untouchable and unable to perform an attacking action for a set number of turns. "src" is the card object descriptor generating this effect (multiple effects of the same card source does not stack, rather is overwritten).

- ***preventDamage****(PLAYER, CARD src, ZTCG_PREVENTTYPE, int qty)*

  Gives PLAYER the ability to prevent "qty" attacks until the end of the turn. Effects from the same card descriptor does not stack, rather is overwritten.

- *bool res =* ***hpDrainPrevent****(PLAYER,int self_dmg,CARD target,int target_dmg)*

  Queries PLAYER about taking fixed "self_dmg" damage with their character in order to **prevent** "target_dmg" damage aimed to the declared "target". **This function is designed to be used only within the *onCheckPreventAttack* pointcut.**

- ***destroySelf****(PLAYER,ZTCG_TABLESLOTID)*

  Immediately sends to the Graveyard the card located at the designated slot of PLAYER's perspective.

- ***Active Game Mechanics***

  Functions that represent a *player's commands and course of actions* in the card game.

- ***recoverCard****(PLAYER, ZTCG_DECKTYPE, ZTCG_RECOVERTYPE, ZTCG_COMPARATOR, int level)*

  Prompts PLAYER to get a card from their Graveyard of "level" equal or other level matching with "ZTCG_COMPARATOR" nature, and brings said card to the final (right-side) of the ZTCG_DECKTYPE option (either DECK_HAND or DECK_DECK). Does nothing if no card available.

- ***recoverTopCard****(PLAYER, ZTCG_DECKTYPE)*

  Recovers the card currently at the top of the PLAYER's Graveyard and moves it to the end of PLAYER's target pile (ZTCG_DECKTYPE options: DECK_HAND or DECK_DECK). Does nothing if no card available.

- ***recoverDestroyedCard****(PLAYER,CARD card)*

  Fetches for the card object descriptor "card" at PLAYER's Graveyard and returns it to the end of PLAYER's hand.

- *bool ret =* ***summon****(PLAYER, ZTCG_SUMMONMODE,int level)*

  Prompts a PLAYER to play a Mob, Jr.Boss or Boss card to the table if all the requirements were met.

- *bool ret =* ***equip****(PLAYER, ZTCG_EQUIPMODE,int level)*

  Prompts a PLAYER to play an Equipment card to the table if all the requirements were met.

- *bool ret =* ***action****(PLAYER,CARD use_target,int level)*

Prompts a PLAYER to play a Rapid Action card to the table if all the requirements were met. If "use_target" was instantiated with a card object descriptor, the system will override the prompt screen forcefully using this card effect.

- *bool ret =* ***locate****(PLAYER, ZTCG_LOCATEMODE)*

  Prompts a PLAYER to play a Field card to the table if all the requirements were met. "PLAY_FIRSTCARDFIELD" flag from "ZTCG_LOCATEMODE will try to play the far-left card in PLAYER's Hand, giving up if said card is not a Field-type.

- *bool ret =* ***playCard****(PLAYER, string text, ZTCG_PLAYCARDMODE (flag), int level)*

  Prompts a PLAYER to play a card, with type covered by "ZTCG_PLAYCARDMODE flag options, to the table if all the requirements were met.

- ***shuffleDeck****(DECK deck)*

  Shuffles cards of the given "deck" object descriptor.

- *bool res =* ***throwCoin****(PLAYER)*

  Prompts PLAYER a coin side to select and returns True if PLAYER won the round, False otherwise.

- *bool draw =* ***drawCard****(PLAYER)*

  PLAYER draws the top card of their deck. Returns True if a card has been drawn.

- ***scoutMob****(PLAYER)*

  PLAYER checks the top card of their deck. If said card is a Mob, Jr. Boss or Boss-type, draws it. Furthermore, if it is of character's level or less and is playable by common means by the PLAYER, this card is immediately played.

- ***discardCard****(PLAYER)*

  Forces PLAYER to discard a card from their Hand. Does nothing if Hand is empty.

- ***discardRandomHandCard****(PLAYER)*

  Discards a random card from PLAYER's Hand. Does nothing if Hand is empty.

- ***peekNextCard****(PLAYER)*

  Previews for PLAYER only the top card of their Deck.

- ***revealCard****(PLAYER,string msg,CARD card)*

  Previews for both players the top card of PLAYER's Deck.

- ***levelUp****(PLAYER)*

  Prompts PLAYER about selecting a card in their Hand to use it to level up their Character.

- *bool res =* ***applyCharacterAction****(PLAYER,CARD card, ZTCG_CALL)*

  Permits PLAYER to call a Character Action effect from "card" of common type or, in case of "card" being a Character, a *specific* Character Action of the 3 initial abilities. Action called by this function will be played **regardless of requirements**.

- ***Attack Mechanics***

Player commands related to declaring or commiting an attack instance.

- ***addExtraMobAttack****(CARD card,CARD src,int qty)*

  Permits a Mob, Jr. Boss or Boss-type "card" more than one mob attack instance (usable only during Mob Attack phase) for the given turn duration or until card is removed from table. Buffs from same "src" does not stack, rather are overwritten.

- ***removeExtraMobAttack****(CARD card,CARD src)*

  Removes an applied buff on "card" from source "src".

- ***increaseNextAttackDamage****(PLAYER,int bonus)*

  Applies a buff on PLAYER, giving the next attack additional "bonus" damage.

- ***increaseDamageMultiplierByPercent****(CARD card,int incPercent)*

  Applies a buff on "card", that will be applied when said card performs an attack. Applied buff lingers around until the next turn or "card" is removed from table.

- *bool ret =* ***mobReaver****(PLAYER,CARD attacker, int mult, ZTCG_PLAYERMODE)*

  Applies an attack instance created by "attacker", targeting a character or mob. Attack value is defined based on *"mult"* times *number of Mobs, Jr. Boss and Boss* deployed on table sides that are covered by "ZTCG_PLAYERMODE". Returns True if attack succeeded (had no preventions, canceling of sorts).

- *bool ret =* ***equipReaver****(PLAYER,CARD attacker, int mult, ZTCG_PLAYERMODE)*

  Applies an attack instance created by "attacker", targeting a character or mob. Attack value is defined based on *"mult"* times *number of Equipments* deployed on table sides that are covered by "ZTCG_PLAYERMODE".

Returns True if attack succeeded (had no preventions, canceling of sorts).

- *bool used, CARD alvo =* ***attack****(PLAYER, CARD attacker, int damage, ATK_RES, ATK_SRC, ZTCG_FIXEDTARGET, ZTCG_TRUESTRIKE, ZTCG_PREVENT, ZTCG_COUNTER)*

  Creates an attack instance with the given arguments. PLAYER is the one commanding the attack, "attacker" is the card originating it, "damage" is the base attack value. Returns True if attack succeeded (had no preventions, canceling of sorts) and a card object descriptor of the target.

- *bool used =* ***attackGlobal****(ZTCG_GLOBALMODE (flag), PLAYER, CARD attacker, int damage, ATK_SRC, ZTCG_TRUESTRIKE, ZTCG_PREVENT, ZTCG_COUNTER, ZTCG_FILTER, short min_atrib, int min_lv, int max_lv, ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring)*

  Creates multicasting attack instances for every target that matched the given filters. Every target receives the same attack pattern. PLAYER is the one commanding the attack, "attacker" is the card originating it, "damage" is the base attack value. Returns True if at least one attack succeeded (had no preventions, canceling of sorts). **Can not target Equipments** as it is not defined as a directed attack.

- *bool killed =* ***lastAttackKilled****(PLAYER)*

  Checks if the last attack committed by PLAYER destroyed a card.

- ***applyGlobalDOT****(ZTCG_GLOBALMODE (flag), PLAYER, CARD src, int dmg, int turns, ZTCG_FILTER, short min_atrib, int min_lv, int max_lv, ZTCG_TYPE (flag), ZTCG_ELEMENT (flag), string substring)*

  PLAYER's "src" multicasts a DOT effect for all opponent mobs and characters. If targeted cards already had a DOT effect from "src" applied, the debuff is simply overwritten. Effects deals fixed "dmg" value, lingers for some "turns", and targets PLAYER's opponent cards within the given filters.

- ***Deployed Card Status***

  Functions that works around maintaning conditions of an unit in a match (such as stuns, silences, etc.).

- *bool stunned =* ***isStunned****(CARD card)*

  Checks if "card" is stunned.

- *bool hijacked =* ***isHijacked****(CARD card)*

  Checks if "card" is hijacked/swapped.

- *bool silenced =* ***isSilenced****(CARD card)*

  Checks if "card" is silenced.

- ***applyStun****(PLAYER,CARD card,int turns)*

  Player PLAYER applies a "turns"-timed instance of stun on opponent's "card".

- ***applyHijack****(PLAYER,CARD card,int turns)*

  Player PLAYER applies a "turns"-timed instance of hijack/swap on opponent's "card".

- ***applySilence****(PLAYER,CARD card,int turns)*

  Player PLAYER applies a "turns"-timed instance of silence on opponent's "card".

- ***applyDOT****(PLAYER, CARD target, CARD src, int dmg, int turns)*

Makes opponent's "src" card apply a "turns"-timed instance of fixed "dmg"-DOT (Damage-Over-Turn) on PLAYER's "target".

- ***applyGlobalSilence****(PLAYER)*

  PLAYER applies an **ambient silence** effect, where *no Mob, Jr. Boss or Boss-types* play their card effects during the whole turn.

- ***applyGlobalStun****(PLAYER)*

  PLAYER applies an **ambient stun** effect, where *no Mob, Jr. Boss or Boss-types* play their mob attack during the whole turn.

- ***editCardHP****(CARD card,int newHP)*

  Forcefully changes "card" current HP with the "newHP" value.

- *bool ret =* ***refreshHP****(PLAYER,CARD card,int incHP)*

  Replenishes PLAYER's "card" HP by given "incHP" value.

- *bool ret =* ***increaseMaxHP****(CARD card,int incHP)*

  Increments MaxHP of "card" by "incHP".

- ***newCardLevelDrop****(CARD card, CARD src, int level_drop, int turns)*

  Similar to *buffs*, a "src" card causes "card" to receive a "level_drop" for the given amount of turns. Level drop value cannot be stacked by the same "src". Refer to the level dropping section for detailed informations.

- ***removeCardLevelDrop****(CARD card, CARD src)*

Makes "card" to forcefully lose it's level drop attribute given from "src". Level drop instances with no attributed source only loses it's effect by wearing off over turns.

- ***Math Functions***

  Functions that deals with simple math calculations (such as roundings and border values).

- *int max =* ***getRoundedMax****(int val1,int val2,int val3)*

  Gets the rounded by 10 value of the maximum between the given 3 arguments.

- *int mid =* ***getRoundedMid****(int val1,int val2,int val3)*

  Gets the rounded by 10 value of the middle argument between the given 3 arguments.

- *int res =* ***getRoundedUp****(int val)*

  Gets the ceiling 10 value of the given argument.

- *int res =* ***getRoundedNearest****(int val)*

  Gets the rounding 10 value of the given argument.

- *int res =* ***getMinimum****(int val1,int val2)*

  Gets the lesser argument between the given 2 arguments.

- *int res =* ***getMaximum****(int val1,int val2)*

Gets the greater argument between the given 2 arguments.

5.3.2 Pointcuts

Pointcuts are breakpoints in the engine system where the card designer have the opportunity to insert their own card rules. Making use of atomic rules and some additional scripting maneuvers, one can implement their own card effects to be played in the match.

Essentially, only through the pointcuts (which triggers after an event in a match) the users can insert their card codes. To gain access through a pointcut, the card designer must, from within the custom card's scope, **create** a **Lua function** that:

- Holds the **same name** of the pointcut desired.
- This function must have one and only one argument, which represents a match input for the pointcut.

Created functions that defines behaviors and effects of a card on a pointcut **must** stay in terms with what is expected of a Lua function, otherwise this function will be ignored by the parser. Keep up with the console logs, when initializing the game system, to see how well your cards fare with the system. Errors arisen from **compilation failures** of the Lua code or initialization of the custom cards will be **reported to the console** during startup.

Generally, at a given event trigger one can find three unique versions of pointcuts.

- For **cards supporting** the current controlling player.
- For **cards opposing** the current controlling player.
- For a **specific target card** of the pointcut.

Complete Details Essentially, one can use pointcuts as means to implement their card codes in the game at the given event trigger the pointcut represents. However, it is possible to enrich such implementations by **using current match status and elements**, that are provided by some pointcuts. **Even interact with the match system**, by using these features. Check out the file "framework_pointcuts.ods" at "extra" folder, for further details about data usability of each pointcut.

- ***Prevent Target***

Specialized series of pointcuts which *requests a value to return* to signalize if a specific card can not be targeted by an incoming attack. Should the pointcut return True, said attack can not target the card.

- *preventTargetCharacter*
- *preventTargetMob*
- *preventTargetEquip*

- *onGetTargeted* – Triggered by mobs currently being searched to become a target. Called whenever one is cogitating to perform a attack or commiting to it. Observe pointcut flag "ZTCG_ATTACKSTATE".

- *onCheckPreventAttack* – Triggered as a last resort for a defending card, which queries the system for a chance to prevent incoming damage. If this pointcut **returns** True, defending card negates incoming damage (only if attack is "preventable").

- *onEquipBuff* – Verifies Equipment cards for a buff to apply. Called whenever one is cogitating to perform a attack or commiting to it. Observe pointcut flag "ZTCG_ATTACKSTATE".

- *renderCardRegister* – Pointcut intended to *write a card register* value. Use displayCardRegister.

- ***Buff modifiers***

  Toggles between applying and negating buff effects over all deployed cards. Usually called when instantiating actions that somehow modifies card status in the game. **It is extremely important that custom cards implement these two pointcuts as *polar opposites* of one another. Whatever one adds the other takes, and vice-versa.**

- *undoBuffs* – activates for cards of both sides simultaneously.

- *applyBuffs* – activates for cards of both sides simultaneously.

- ***Selecting Card from a List***

  Triggers whenever a player selected a generic card on a list of cards.

- *onSelectMenu*
- *onSelectMenuCard*
- *onOpponentSelectMenuCard*

- ***Card Refresh***

  Triggers whenever a card is healing a portion of it's HP.

- *onRefresh*
- *onRefreshCard*
- *onOpponentRefreshCard*

- ***Pick Card Order***

  Triggers whenever a player is visible-ordering a list of cards.

- *onPickCardOrder*
- *onOpponentPickCardOrder*

- ***Card Recovery***

Triggers whenever a player is taking out a card from the Graveyard.

- *onRecover*
- *onRecoverCard*
- *onOpponentRecoverCard*

- ***Throw Coin***

  Triggers whenever a player is tossing a coin. On common tossing, a player calls a coin side and verifies the result. On fixed tossing, a card defines behaviors for each side of the coin and the winning branch is chosen.

- *onThrowCoin*
- *onOpponentThrowCoin*
- *onThrowCoinFixed*
- *onOpponentThrowCoinFixed*

- ***Action Prompt***

  Triggers whenever a player receives a prompting screen from the system (usually related to requesting a player's binary-based decision).

- *onPrompt*
- *onOpponentPrompt*

- ***Card Drawn***

Intercept version triggers **before** a player draws a card, however it can not negate a card-drawing action. **"onDraw" refers only to the card being drawn and the "on**Card" pointcuts runs for every card deployed of a side of the table.** This pattern repeats through most of the pointcut families.

- *onInterceptDrawCard*
- *onOpponentInterceptDrawCard*
- *onDraw*
- *onDrawCard*
- *onOpponentDrawCard*

- ***Card Removed***

Triggers whenever a card is *removed from somewhere* and going directly to the Discard Pile (Graveyard).

- *onMoveToDiscardPile*
- *onCardMovedToDiscardPile*
- *onOpponentCardMovedToDiscardPile*

- ***Card Destroyed***

Triggers whenever a card gets removed *from the Table*, going to the Discard Pile.

- *onDestroy*
- *onCardDestroyed*
- *onOpponentCardDestroyed*

- ***Card Discarded***

Triggers whenever a card gets removed *from the Hand*, going to the Discard Pile.

- *onDiscard*
- *onDiscardCard*
- *onOpponentDiscardCard*

- ***Card Revealed***

  Triggers whenever card is revealed to both players.

- *onReveal*
- *onRevealCard*
- *onOpponentRevealCard*

- ***Card Discarded Randomically***

  Triggers whenever a card is randomly discarded.

- *onRandomDiscard*
- *onRandomDiscardCard*
- *onOpponentRandomDiscardCard*

- ***Card Peeked***

  Triggers whenever a card is shown to one player.

- *onPeek*
- *onPeekCard*
- *onOpponentPeekCard*

- ***Card Stunned***

  Triggers whenever a card gets stunned.

- *onReceiveStun*
- *onReceiveStunCard*
- *onOpponentReceiveStunCard*

- ***Card Hijacked***

  Triggers whenever a card gets hijacked.

- *onReceiveHijack*
- *onReceiveHijackCard*
- *onOpponentReceiveHijackCard*

- ***Card Silenced***

  Triggers whenever a card gets silenced.

- *onReceiveSilence*
- *onReceiveSilenceCard*
- *onOpponentReceiveSilenceCard*

- ***Equipment Destroyed***

  Triggers whenever an equipment gets destroyed.

- *onDestroyOpponentEquipment*
- *onDestroyedEquipment*

- ***Calculating an Attack***

  Triggers whenever a player is thinking on an attack an unit or committing it. Applies transformations on an attack output value.

- *onCalcAttack*
- *onCalcAttackCard*
- *onOpponentCalcAttackCard*

- ***Calculating a Defense***

  Triggers whenever a player is thinking on an attack an unit or committing it. Applies transformations on a defense/block output value.

- *onCalcDefense*
- *onCalcDefenseCard*
- *onOpponentCalcDefenseCard*

- ***Starting an Attack***

  Triggers whenever a player **commits** to an attack.

- *onStartAttack*
- *onOpponentStartAttack*

- ***Attack Instance Actors***

  Triggers only to a **specific pair of opposing cards**: the one attacking and the one receiving an attack.

- *onExecuteNormalAttack*
- *onReceiveNormalAttack*

- ***Attack Instance with a Fatality: Actors***

  Triggers only for the actors (attacker and attacked units) of an attack instance resulted in a destroyed defending unit.

- *onExecuteAttackAndDestroyed*
- *onReceiveAttackAndDestroyed*
- *onReceiveAttackAndSentToDiscardPile*

- ***Attack Instance with a Fatality: Remaining Cards***

  Triggers only when the defending card is destroyed, regarding the type of the destroyed card. Triggers to remaining cards on the field of a given side.

- *onAttackOpponentMobDestroyed*
- *onAttackMobDestroyed*
- *onAttackOpponentCharacterDestroyed*
- *onAttackCharacterDestroyed*
- *onMobSentToDiscardPile*
- *onOpponentMobSentToDiscardPile*

- ***Attack Launch***

  Triggers whenever a player commits to an attack. Counterattacks are instantiated only if the *defending unit remained alive* after said attack, and said attack was branded as "counterable".

- *onLaunchAttack*
- *onInterceptAttack*
- *onApplyCounterAttack*

- ***Attack Instance without a Fatality: Actors***

  Triggers whenever an attack instance terminated without destroying the target card. Applies only to the actors of this attack (attacker and attacked units).

- *onExecuteAttackAndSurvived*
- *onReceiveAttackAndSurvived*

- ***Attack Committed***

  Triggers whenever an attack is declared. Affects all cards of a side of the table.

- *onExecuteAttack*
- *onReceiveAttack*

- ***Card played***

  Triggers whenever a card is played on the table (does not trigger when sliding a card under a deployed card).

- *onPlayCard*
- *onOpponentPlayCard*

- ***Rapid Action played***

  Triggers whenever a Rapid Action is played.

- *onInitAction*[21]

[21] onInit: pointcut applies only to the card being played, at the moment of it's introduction.

- *onInterceptPlayAction*[22]
- *onInterceptOpponentPlayAction*
- *onThinkAction*[23]
- *onPlayAction*[24]
- *onOpponentPlayAction*

- ***Equipment played***

  Triggers whenever an Equipment is played.

- *onInitEquipment*
- *onInterceptPlayEquipment*
- *onInterceptOpponentPlayEquipment*
- *onThinkEquipment*
- *onPlayEquipment*
- *onOpponentPlayEquipment*

- ***Field played***

  Triggers whenever a Field is played. Observe that, if already exists a field in a player's side and they decide to play another, *the first is destroyed* before the second comes to the game.

- *onInitField*
- *onInterceptPlayField*
- *onInterceptOpponentPlayField*
- *onThinkField*
- *onPlayField*
- *onOpponentPlayField*
- *onDestroyField*

[22] onIntercept: pointcut applies to all deployed cards of one side of the table, at the moment of a card's introduction.
[23] onThink: pointcut applies only to the played card, at the moment it is brought into table.
[24] onPlay: pointcut applies to all deployed cards of one side, after it has been brought into table.

- *onOpponentDestroyField*

- ***Mob played***

  Triggers whenever a Mob, Jr. Boss or Boss is played.

- *onInitMob*
- *onInterceptPlayMob*
- *onInterceptOpponentPlayMob*
- *onThinkMob*
- *onPlayMob*
- *onOpponentPlayMob*

- ***Turn Started***

  Triggers whenever a player's turn starts.

- *onStartTurn*
- *onOpponentStartTurn*

- ***Onset Phase Started***

  Triggers whenever a player's Onset turn phase starts.

- *onStartCharacterActions*
- *onOpponentStartCharacterActions*

- ***Refresh Turn Card Effects***

Triggers every start of turn, refreshing card status and other modifiers that will be used for all the round-turn duration.

- *onRecoverStats* – activates for cards of both sides simultaneously.
- *onReactivateCardEffects*
- *onOpponentReactivateCardEffects*

- ***Character Actions Phase Started***

  Triggers whenever a player's Character Actions turn phase starts.

- *onStartCharacterActions2*
- *onOpponentStartCharacterActions2*

- ***Level Up Phase Started***

  Triggers whenever a player's Level Up turn phase starts. A pointcut detects one-shot character actions ("onLevelActionTrigger"), permitting the leveled card to use it's one-shot effect.

- *onLevelActionTrigger*
- *onLevelUp*
- *onOpponentLevelUp*

- ***Level Up Phase Ended***

  Triggers whenever a player's Level Up turn phase ends, and before the Character Actions phase starts (Onset phase).

- *afterLevelUp*
- *afterOpponentLevelUp*

- ***Starting a Character Action***

  Triggers whenever a character action is about to start.

- *onInterceptCharacterAction*
- *onOpponentInterceptCharacterAction*

- ***Executing a Character Action***

  Triggers whenever a player's character action is ready to be executed. Actions *from own character are numbered* from the top, and *any other actions at the build tree* uses the numberless version.

- *onActivateCharacterAction1*
- *onActivateCharacterAction2*
- *onActivateCharacterAction3*
- *onActivateCharacterAction*

- ***After a Character Action***

  Triggers whenever a character action has been played.

- *afterApplyCharacterAction*
- *afterOpponentApplyCharacterAction*

- ***Character Actions Phase Ended***

  Triggers whenever a player's Character Action turn phase ends, and before Mob Actions phase starts.

- *afterCharacterActions*
- *afterOpponentCharacterActions*

- ***Mob Actions Phase Started***

  Triggers whenever a player's Mob Actions turn phase starts. Affects only deployed mobs of both parties.

- *onActivateMobEffect* – activates for mobs of both sides simultaneously.
- *onRecoverMobStats* – activates for mobs of both sides simultaneously.

- ***Turn Ended***

  Triggers whenever a player's turn ends.

- *onEndTurn*
- *onOpponentEndTurn*

5.4 Effects Engine

To add flavor to a game, many 2D games adopt the use of Sprites and Particles to represent animation and some special effects on the screen. In this card game engine, the Effects engine allows the user to specify their own effects they want to deploy in the game. Cards shining, animating a card play, basically anything they want to draw in the screen can be done by defining a **spritesheet** to get the moving object and a **sprite descriptor** defining which images will be selected to animate the image.

Defining sprites and particle effects:

Sprite Closed sequence of images, drawn one-after-another (in a cycle fashion), generally representing something being animated.

Particle In-game consolidation of a sprite, modified according to circumstances: drawn rotated, flipped, scaled, for a set number of frames, etc.

This section revolves around the operations done under *"interface/effects"* from the root of the application.

5.4.1 Loading a Sprite

Figure 19: Example of a spritesheet.

Bluntly stating, any displayable effect on the card game is a series of juxtaposed images that, when displayed in a cycle, forms the idea of an object in movement.

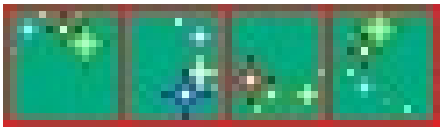


Figure 20: Sprite description.

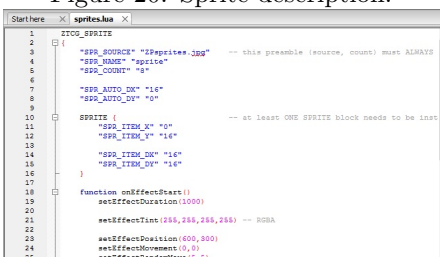


Figure 21: Sprite code description.

Take for example this set of sequential images. Each square is defined as an image that makes part of this set, which will be called an animated **sprite**. The user can either **manually list each picture** to be used for the new sprite or **automate the process** by defining a starting point on the sheet for the first image and length and width to get the next images in the array.

These requirements must be met when using the sprite creator engine:

- The **spritesheet** and **number** of images must be specified before stating an image block, denoted by "SPRITE" before entering brackets.
- Lua-functions must come after the definition of the image blocks.

Lua-functions in this context are used to *specify behaviors within a specific sprite* interaction with the game engine. The sprite engine supports three functions, namely:

- onEffectStart – triggers when a particle is registered in the game engine.
- onEffectUpdate – triggers every game frame. **Use caution** when creating new effects from here.
- onEffectFinish – triggers when a particle is unregistered from the game engine (it's life counts to zero).

*Note: These functions are **limited** to only create new effects within it's scope and can not alter the game state in any other way.*

5.4.2 Sprite/Effect Functions

These functions defines parameters for **managing an existent Effect** in the game.

- **resetEffectAttributes**() – defaults all parameters for the current effect.
- **setEffectDuration**(int duration) – sets the number of frames the current effect will last until it expires.
- **setEffectTint**(int r, int g, int b, int a) – sets color and alpha transparency for the current effect.
- **setEffectPosition**(int x, int y) – defines starting point of the current effect.
- **setEffectMovement**(int dx, int dy) – defines expected movement of the current effect.
- **setEffectRandomMove**(int rx, int ry) – defines randomness on each frame's calculated position of the current effect.
- **setEffectRandomStep**(int rsx, int rsy) – defines randomness on where the current effect will be drawn each frame, but does not affect it's next position.
- **setEffectScale**(x,y) – defines scalability of the current effect, when compared with the given source sample. Use numbers between 0 and 1 in 'x' and 'y' (regarding coordinates) to draw mini-versions of it.
- **setEffectRotationCenter**(int cx, int cy) – defines the anchor point of the current effect, from which it will rotate around.
- **setEffectRotationAngle**(angle) – in radians, defines rotation of the current effect.
- **setEffectFlags**(ZTCG_DRAWFLAG flags) – sets some flags for rendering flipped versions of the current effect. See flag options on the ZTCG_DRAWFLAG section.
- **setEffectBlendingFlags**(ZTCG_DRAWBLEND op, ZTCG_BLENDMODE src, ZTCG_BLENDMODE dest) – selects blending options to draw

the current effect. See flag options at the ZTCG_DRAWBLEND and ZTCG_BLENDMODE sections.

- **setEffectDrawType**(ZTCG_DRAWMODE) – defines the type of drawing that will be applied at the current effect. See flag options at ZTCG_DRAWMODE section.
- int duration = **getEffectDuration**() – returns the number of frames the effect will last until it expires.
- int r, int g, int b, int a = **getEffectTint**() – returns color and alpha transparency for the effect.
- int x, int y = **getEffectPosition**() – returns starting point of the effect.
- int dx, int dy = **getEffectMovement**() – returns expected movement of the effect.
- int rx, int ry = **getEffectRandomMove**() – returns randomness on each frame's calculated position of the effect.
- int rsx, int rsy = **getEffectRandomStep**() – returns randomness on where the effect will be drawn each frame, but does not affect it's next position.
- x,y = **getEffectScale**() – returns scalability of the effect, when compared with the given source sample.
- int cx, int cy = **getEffectRotationCenter**() – returns the anchor point of the effect, from which the effect will rotate around.
- angle = **getEffectRotationAngle**() – in radians, returns rotation of the effect.
- ZTCG_DRAWFLAG flags = **getEffectFlags**() – returns some flags for rendering flipped versions of the effect. See flag options on the ZTCG_DRAWFLAG section.
- ZTCG_DRAWBLEND op, ZTCG_BLENDMODE src, ZTCG_BLENDMODE dest = **getEffectBlendingFlags**() – returns blending options being used to draw the effect. See flag options at the ZTCG_DRAWBLEND and ZTCG_BLENDMODE sections.
- ZTCG_DRAWMODE = **getEffectDrawType**() – returns the type of drawing that will be applied at the given effect. See flag options at ZTCG_DRAWMODE section.

These functions defines parameters for **deploying a new Effect** in the game.

- **resetNewEffectAttributes**() – defaults all parameters for the new effect.
- **setNewEffectDuration**(int duration) – sets the number of frames the new effect will last until it expires.
- **setNewEffectTint**(int r, int g, int b, int a) – sets color and alpha transparency for the new effect.

- **setNewEffectPosition**(int x, int y) – defines starting point of the new effect.
- **setNewEffectMovement**(int dx, int dy) – defines expected movement of the new effect.
- **setNewEffectRandomMove**(int rx, int ry) – defines randomness on each frame's calculated position of the new effect.
- **setNewEffectRandomStep**(int rsx, int rsy) – defines randomness on where the new effect will be drawn each frame, but does not affect it's next position.
- **setNewEffectScale**(x,y) – defines scalability of the new effect, when compared with the given source sample. Use numbers between 0 and 1 in 'x' and 'y' (regarding coordinates) to draw mini-versions of it.
- **setNewEffectRotationCenter**(int cx, int cy) – defines the anchor point of the new effect, from which it will rotate around.
- **setNewEffectRotationAngle**(angle) – in radians, defines rotation of the new effect.
- **setNewEffectFlags**(ZTCG_DRAWFLAG flags) – sets some flags for rendering flipped versions of the new effect. See flag options on the ZTCG_DRAWFLAG section.
- **setNewEffectBlendingFlags**(ZTCG_DRAWBLEND op, ZTCG_BLENDMODE src, ZTCG_BLENDMODE dest) – selects blending options to draw the new effect. See flag options at the ZTCG_DRAWBLEND and ZTCG_BLENDMODE sections.
- **setNewEffectDrawType**(ZTCG_DRAWMODE) – defines the type of drawing that will be applied at the new effect. See flag options at ZTCG_DRAWMODE section.

And finally, with the right attributes set, the new effect must be deployed using the function:

createEffect(string sprite_name) – creates and registers a new effect using the sprite given by the "sprite_name" parameter. Does nothing if the provided sprite has not been found or has not been created upon system start-up.