# Kolyma Card System Documentation

Elizabeth Arnold
3/18/24

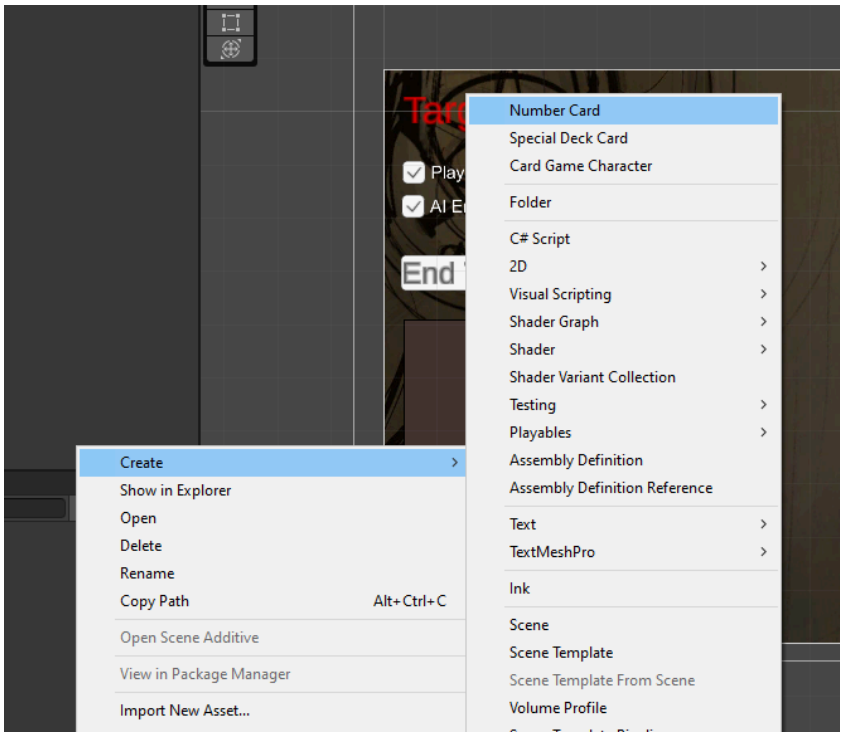
At time of writing, card functionality is located in Assets/Scripts/Card Game (Elizabeth Version) and ./Card Game (Elizabeth Version)/Cards. Relevant scripts are:

- CardGameManager.cs : Contains logic for card effect execution, drawing, discarding, etc. Governs the whole card game.
- DisplayCard.cs : Controls instances of CardPrefab which displays the visible UI of a card to the player, such as art and text.
- DragCard.cs : Functionality for the player to drag around a CardPrefab, dragging and dropping to various zones on the board is how the player can play, discard, or otherwise interact with the cards.
- GenericCard.cs : Abstract parent class for all types of cards, useful for when functionality is shared between special and number cards. Child class of ScriptableObject.
- NumberCard.cs : Child class of GenericCard representing a number card. A scriptable object class which contains all data needed for a given number card to function visually and in gameplay.
- SpecialDeckCard.cs : Child class of GenericCard representing a number card. A scriptable object class which contains all data needed for a given number card to function visually and in gameplay.
- CardGameCharacter.cs : This one is actually in ./Card Game (Elizabeth Version)/Characters. Child of ScriptableObject. Represents all data needed for a player participating in the card game, such as their current hand, their decklist, name, artwork, etc.

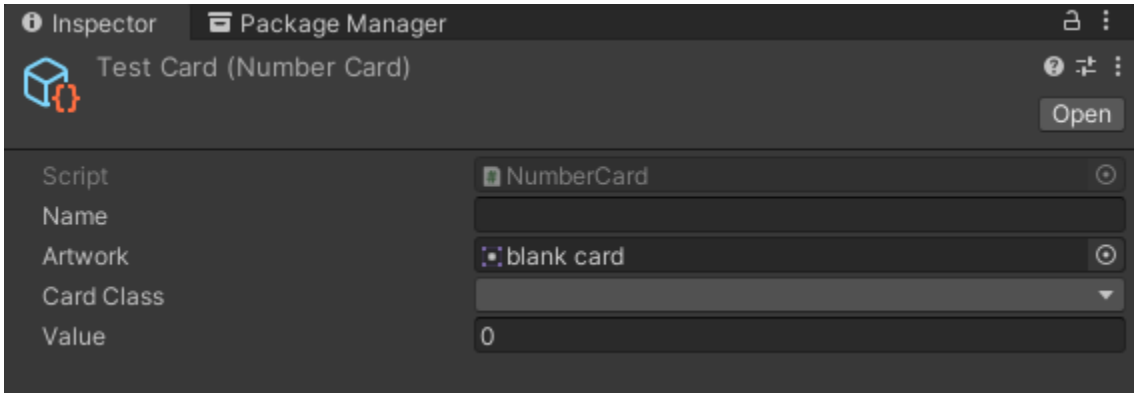
## Creating A Number Card

Cards are represented as scriptable objects. To make a new one, a new file must be created, then fill in the settings, then add it to the relevant lists to be included in gameplay. At the time of writing, number card files are stored in ./Card Game (Elizabeth Version)/Cards/Number Cards

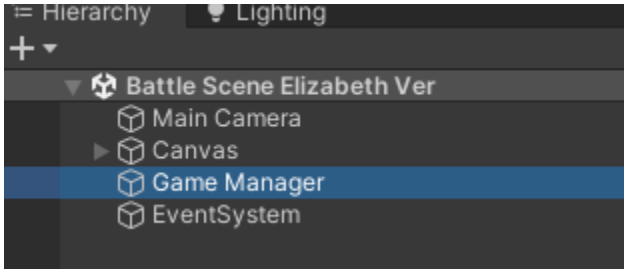
1. Create the file by right clicking in the desired folder in the Project window, then selecting Create (Located at the top), then Number Card (Located at the top of the sub-menu).




2. Name your new file, then select it and fill out its settings in the Inspector window.

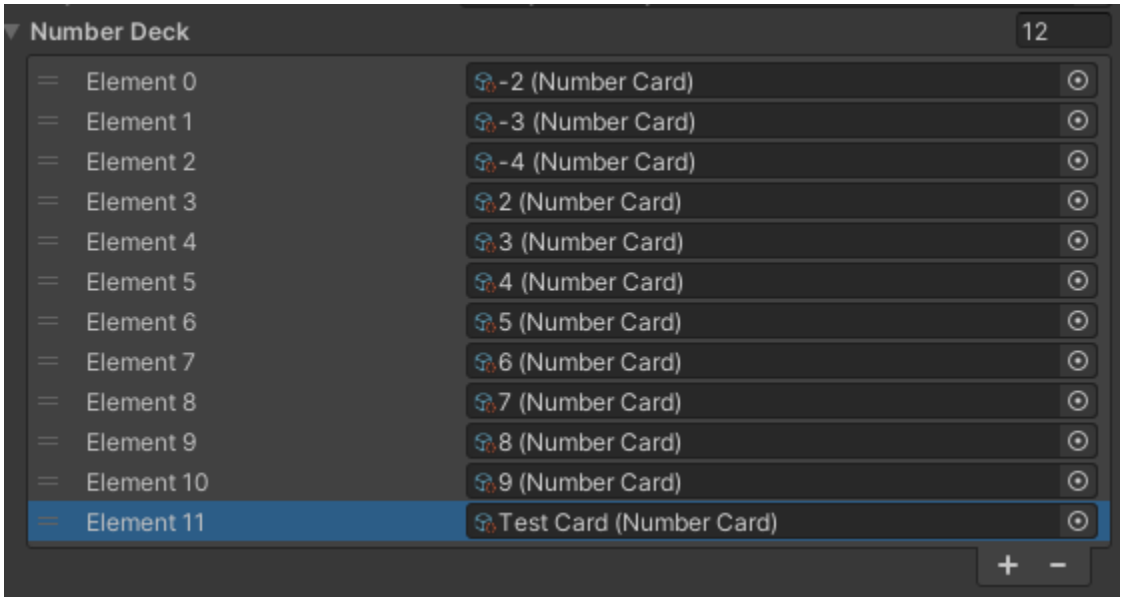



3. Open the scene Battle Scene Elizabeth Ver in ./Card Game (Elizabeth Version). Locate the Game Manager object in the Hierarchy and select it to view it in the Inspector.




4. In the Inspector, expand the Number Deck dropdown. Click + in the lower right of the dropdown to add an element to the list. Drag and drop the new number card file from the

Project window into that new element.




5. SAVE. All done!

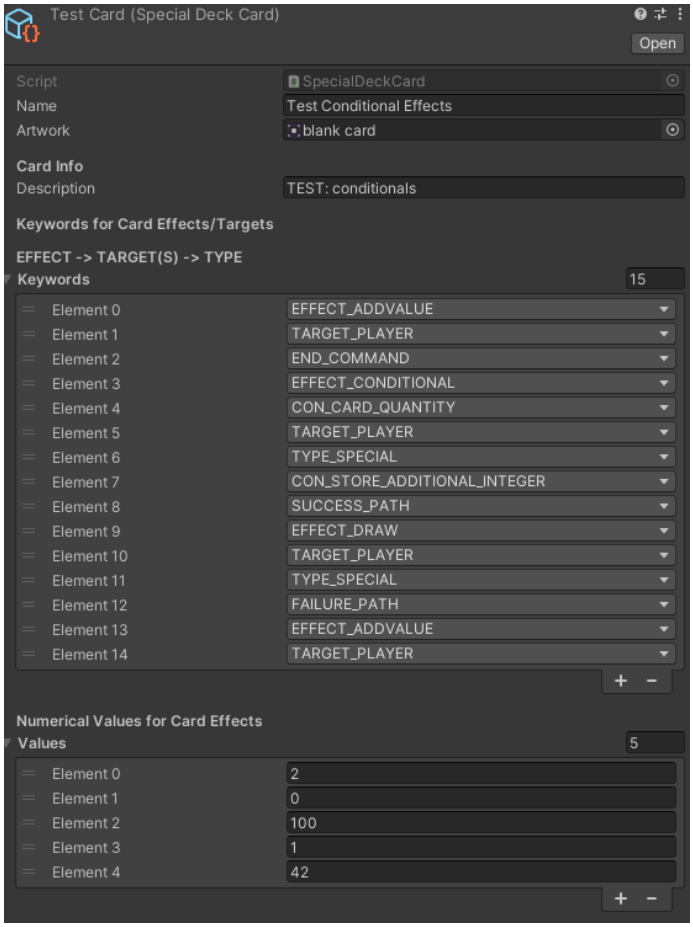
**To edit an existing and already set up number card, simply open the file in the Inspector and change the settings.**

## Creating A Special Card

Cards are represented as scriptable objects. To make a new one, a new file must be created, then fill in the settings, then add it to the relevant lists to be included in gameplay. At the time of writing, special card files are stored in ./Card Game (Elizabeth Version)/Cards/Special Cards.

1. Follow steps 1 and 2 from Creating A Number Card, selecting Special Deck Card in the creation window instead of Number Card, and storing it in the Special Cards folder instead.
2. Special cards will have different settings in the inspector versus number cards. After creating a new special card, refer to the section Special Card Keywords when filling out

the Keywords and Values dropdowns to ensure proper card execution.




3. To place a card in a character's deck, navigate to the desired character file, currently stored in ./Card Game (Elizabeth Version)/Characters. Open the character file in the Inspector and add the special card to the character's Deck List dropdown.

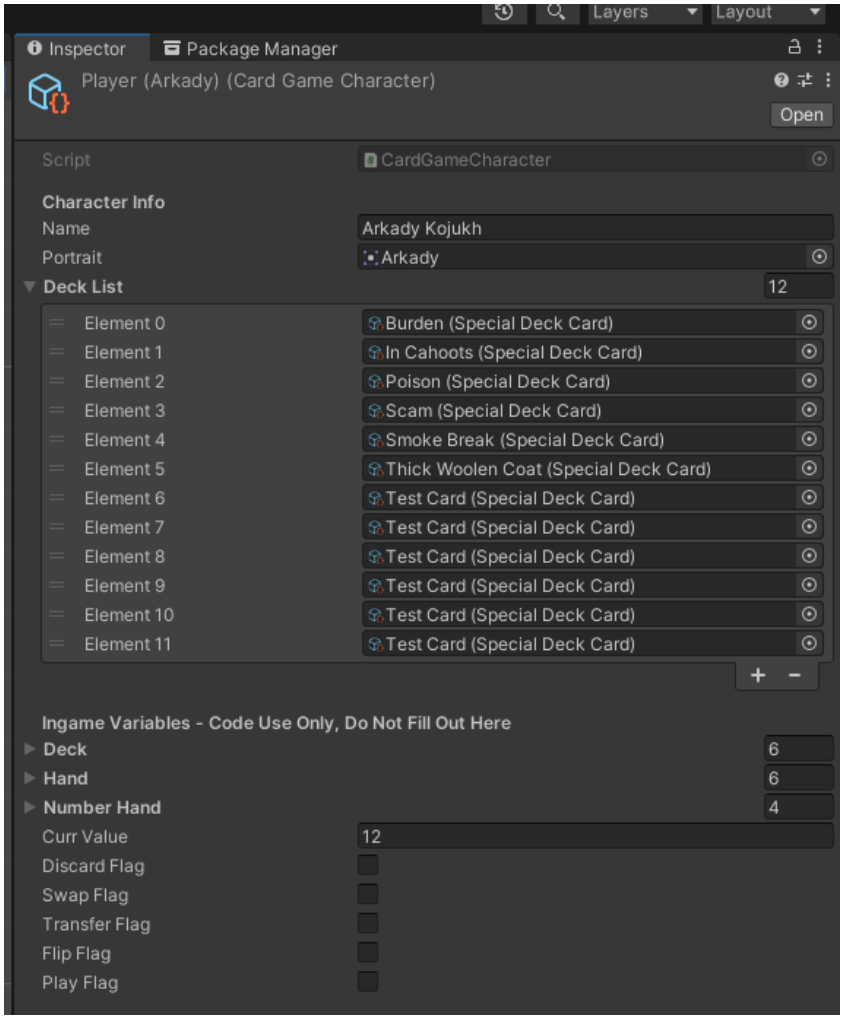
**To edit an existing special card, just edit its settings in its file.**

# Creating A Card Game Character

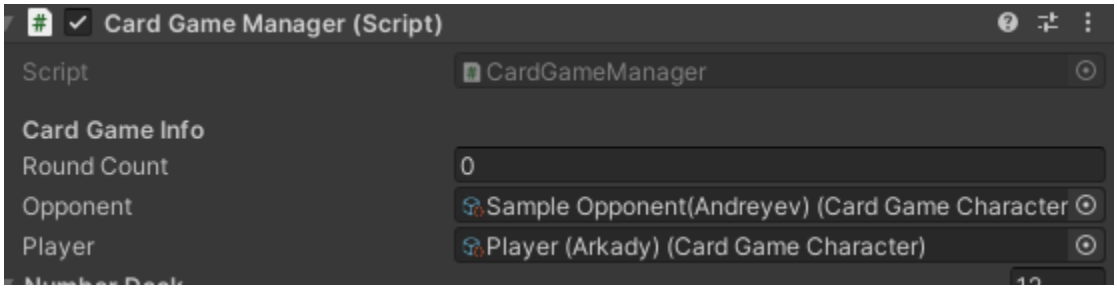
Player settings in the card game are stored in scriptable objects too. These are located in ./Card Game (Elizabeth Version)/Characters.

1. Follow steps 1 and 2 from Creating A Number Card, selecting Card Game Character in the creation window instead of Number Card, and storing it in the Characters folder instead.
2. Note that in the Card Game Character's inspector settings, there is a section labeled Character Info and a section labeled Ingame Variables - Code Use Only, Do Not Fill Out Here. As the name suggests, the Ingame Variables section is not to be edited in the inspector. When setting up a character, just fill out the Character Info section and leave

Ingame Variables alone.




3. Follow step 3 from Creating A Number Card to access the scene and Game Manager if you wish to play as the new card game character, or have the new character be the opponent.
4. In the Game Manager inspector, drag and drop your character's file into the spot for opponent or player as desired.




5. SAVE.

**To edit an existing character, just edit its file. Remember, don't touch Ingame Variables through the inspector. Nothing would break from doing that (I think) but there's no point.**

**Those variables are handled by CardGameManager.cs while the game is running to track the character's state during a match.**

# Special Card Keywords

Every special card has an associated list of Keywords and Values that CardGameManager.cs uses to interpret and execute the card's effects. Think of the keywords list as a sort of sentence. Some keywords expect an integer value associated with them, specifying things such as the number of cards to impact with the effect. Different effects expect different sequences of keywords to follow them. If using keywords expecting values, add the values to the value list in the same order as the keywords requiring them appear.

See table below for details. If a keyword expects a value associated with it to appear in the values array, one must be added- even if it is zero. The expected following keywords lists the types of keywords, if any, that are expected to appear after a given keyword in order. These are specified by their prefixes, and whether multiple keywords of a given type can be entered is noted. When keywords requiring values are present in an expected following sequence, the purpose of the values is listed in {}.

Usage:

1. Select the desired EFFECT and put it in keywords.
2. Enter the Expected Following Keywords for that effect in the specified order.
3. As you enter keywords, check if a value is required for that keyword, and if so, enter it.
4. As you enter keywords, check if a keyword requires following keywords, and if so, enter those. Consider the system nested, so if you encounter another keyword requiring following keywords (the inner keyword) while already entering keywords following another one (the outer keyword), the following keywords for the inner keyword take priority before returning to the remaining following keywords for the outer keyword.
5. A niche note: Due to the nature of the looping used to execute multiple effects on one card, note that effects at the top of the list will execute (and be applied) last; if a card has two or more effects that MUST be executed in a certain order to achieve the desired effect, enter the effect that must go first at the *bottom* of the keywords list.
6. WARNING: When using EFFECT_CONDITIONAL, if using multiple effects (such as EFFECT_ADDVALUE), input EFFECT_CONDITIONAL and its following keywords *last.* At present, the code is not capable of processing effects unrelated to the conditional after the conditional branch, unless such effects are present in the success or failure paths. A brief example in which the user wants EFFECT_ADDVALUE to activate regardless of the conditional success or failure:

   Would work (with appropriate following keywords/values):
   EFFECT_ADDVALUE
   END_COMMAND

EFFECT_CONDITIONAL
SUCCESS_PATH
FAILURE_PATH

EFFECT_ADDVALUE executes independently of the conditional in this case since it was entered first.

Would not work:
EFFECT_CONDITIONAL
SUCCESS_PATH
FAILURE_PATH
END_COMMAND
EFFECT_ADDVALUE

In this case, EFFECT_ADDVALUE executes only in the failure path.

Would work:

EFFECT_CONDITIONAL
SUCCESS_PATH
END_COMMAND
EFFECT_ADDVALUE
FAILURE_PATH
END_COMMAND
EFFECT_ADDVALUE

In this case, EFFECT_ADDVALUE executes in both paths because it is present in both paths.

| **KEYWORD** | **DESCRIPTION** | **EXPECTS VALUE?** | **EXPECTED FOLLOWING KEYWORDS** |
|---|---|---|---|
| EFFECT_NONE | Placeholder default, does nothing. | NO | N/A |
| EFFECT_ADDVALUE | Adds/subtracts from target's total value | NO | [TARGET(s)] {Value to add} |
| EFFECT_DRAW | Draws card types from deck to hand, number or special. | NO | [TARGET(s)] {# to draw} -> [TYPE] |
| EFFECT_DISCARD | Discards selected number and type of cards. | NO | [TARGET(s)] {# to discard} -> [TYPE] |
| EFFECT_TRANSFER | Transfers the selected | NO | [TARGET(s)] {# to |

| | | | |
|---|---|---|---|
| | number and type of cards to the other player's hand. | | transfer} -> [TYPE] |
| EFFECT_SWAP | Discards the selected number and type of cards, then replaces them with new ones drawn from the deck. | NO | [TARGET(s)] {# to swap} -> [TYPE] |
| EFFECT_CONDITIONAL | Denotes the need to check for a condition, with a branching path of effect based on the check results. | NO | [CON(s)] -> [SUCCESS_PATH] -> [FAILURE_PATH] |
| TARGET_PLAYER | Denotes the character who played the card as the target of the effect. | YES, see relevant EFFECT for purposes | N/A |
| TARGET_OPPONENT | Denotes the opponent of the character who played the card as the target of the effect. | YES, see relevant EFFECT for purposes | N/A |
| TYPE_NUMBER | Denotes number cards as the card type used in the effect, such as for drawing or discarding | NO | N/A |
| TYPE_SPECIAL | Denotes special cards as the card type used in the effect, such as for drawing or discarding | NO | N/A |
| END_COMMAND | Used for applying multiple effects to one card; use at the end of a complete command if and only if another command will follow it. | NO | N/A |
| CON_STORE_ADDITIONAL_INTEGER | Used for certain conditional flags requiring more than one integer value to be passed. A placeholder in the Keywords list, holding a spot in the Values list. Represented as [STORE_INT] in CON following keyword lists. | YES, see associated CON for purposes | N/A |

| | | | |
|---|---|---|---|
| CON_HAS_CLASS_CARD | Checks if target has equal or greater number of a given color class of number card | NO | [TARGET] {Color class, represented as int. See NumberCard.cs} -> [STORE_INT] {Count} |
| CON_HAS_DUPLICATE | Checks if target possesses duplicates of given card type in hand. | NO | [TARGET] {N/A/ but must enter something anyways, -1 or w/e works} -> [TYPE] |
| CON_DISCARD_FLAG | Checks if the target discarded a card this turn | NO | [TARGET] {N/A/ but must enter something anyways, -1 or w/e works} |
| CON_SWAP_FLAG | Checks if the target swapped a card this turn | NO | [TARGET] {N/A/ but must enter something anyways, -1 or w/e works} |
| CON_FLIP_FLAG | Checks if the target flipped a card this turn | NO | [TARGET] {N/A/ but must enter something anyways, -1 or w/e works} |
| CON_TRANSFER_FLAG | Checks if the target transferred a card to their opponent this turn | NO | [TARGET] {N/A/ but must enter something anyways, -1 or w/e works} |
| CON_CARD_QUANTITY | Checks if the target has between X and Y (inclusive) cards of a given type in their hand. | NO | [TARGET] {X} -> [STORE_INT] {Y} -> [TYPE] |
| CON_HAS_VALUE_CARD | Checks if the target has at least X number cards of Y value in their hand. | NO | [TARGET] {Y} -> [STORE_INT] {X} |
| CON_COMPARE_AGAINST_TARGET | Checks the target's current total value against the round's target total value. True if current value >= target total, false if current value < total. | NO | [TARGET] {N/A/ but must enter something anyways, -1 or w/e works} |
| SUCCESS_PATH | Denotes path to execute if conditional is true | NO | [the entire effect/series of effects to execute] |

| | | | |
|---|---|---|---|
| FAILURE_PATH | Denotes path to execute if conditional is false | NO | [the entire effect/series of effects to execute] |

## Not Yet Implemented Stuff

Aka the to-do list. Stuff here will work as described above when it's implemented, but the functionality is not there yet.

- Flipping cards
- Flip flags for conditionals; at time of writing they will always return false.