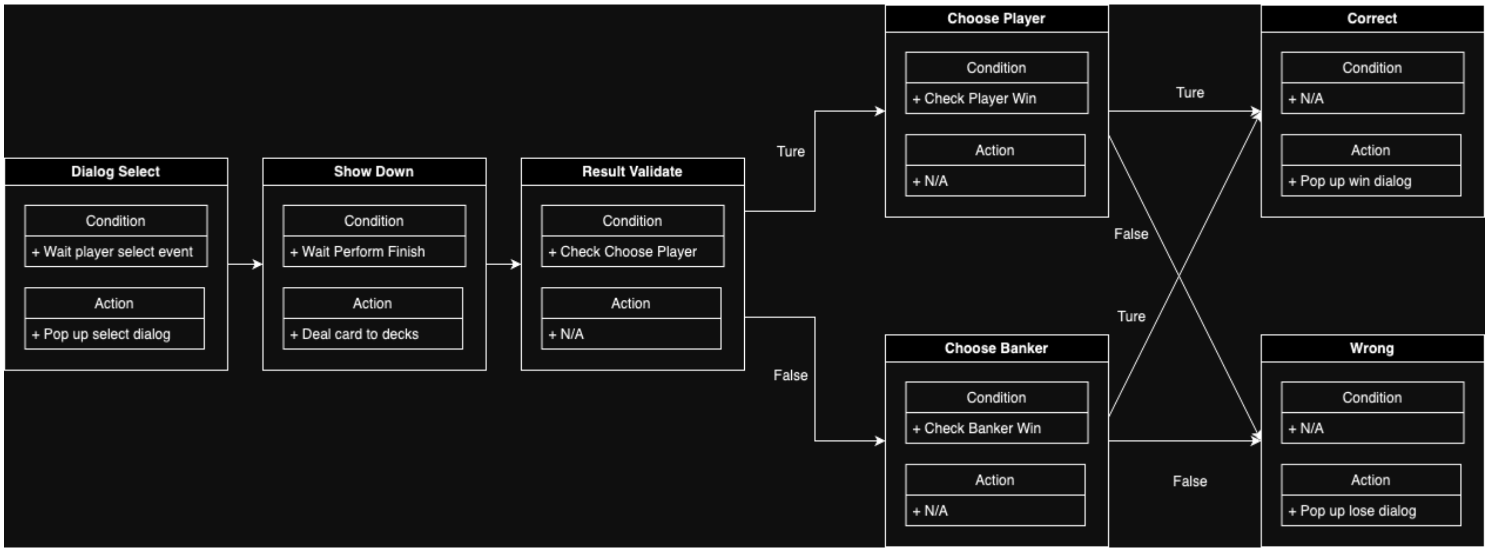
Quick Start

This guide will walk you through the essential steps to create a **High Card** game in Unity using the **DEAL Toolkit**. By the end, you'll have a fully functional card game and a solid understanding of how to use the DEAL framework effectively.

Define High Card FSM



In this **High Card** game, the player guesses who will win—either the **Player** or the **Banker**. After the cards are revealed, the game checks if the player's guess was correct.

You can either create a new FSM Scriptable Object (SO) or use the provided `HighCard_FSMSO` located in `Sample/DataSO`.

The FSM consists of 7 states, each with specific functions, explained below:

---

1. Dialog Select State

- **Overview**: The player selects who they think will win: **Player** or **Banker**. This state manages the selection process.

Steps:

1. Show a dialog for the player to make their choice. Create an action Scriptable Object (SO) using `ShowDialogAction`. Reference the `SelectWinnerDialog_Action` and add it to the **Entry Actions** of this state.
2. Use the `ChoosePlayer_Condition` SO to check if the player selected "Player" (index 0).

---

2. Show Down State

- **Overview**: This state reveals the cards and determines the winner for the current round.

Steps:

1. Deal 1 card to both the **Player** and **Banker**. Add the following 4 actions to the Entry Actions:
    - `TileDraw1Action`
    - `TileDealToPlayer_PostAction`
    - `TileDraw1Action`
    - `TileDealToBanker_PostAction`
2. Add the `CountDown3sAction` to wait for the card flip animation to complete (3 seconds).
3. Unlock the FSM using `EndOfSystemLock_Condition` after 3 seconds.

---

3. Result Validate State

- **Overview**: This state validates whether the player's selection matches the actual result of the showdown.

Steps:

1. Use `ChoosePlayer_Condition` SO to verify if the player’s selection (Player or Banker) was correct.

---

4. Choose Player State

- **Overview**: This state checks if the **Player**’s card value is higher than the **Banker**’s, determining if the guess was correct.

Steps:

1. Use the `PlayerWin_Condition` SO to check if the Player’s card score is higher than the Banker’s.

---

5. Choose Banker State

- **Overview**: This state checks if the **Banker**’s card value is higher than the **Player**’s, determining if the Banker wins.

Steps:

1. Use the `BankerWin_Condition` SO to check if the Banker’s card score is higher than the Player’s.

---

6. Correct State

- **Overview**: If the player guessed correctly, this state triggers a dialog displaying the winning result.

Steps:

1. Add the `ShowCorrectDialogAction` SO to the **Entry Actions** to display the correct result.

---

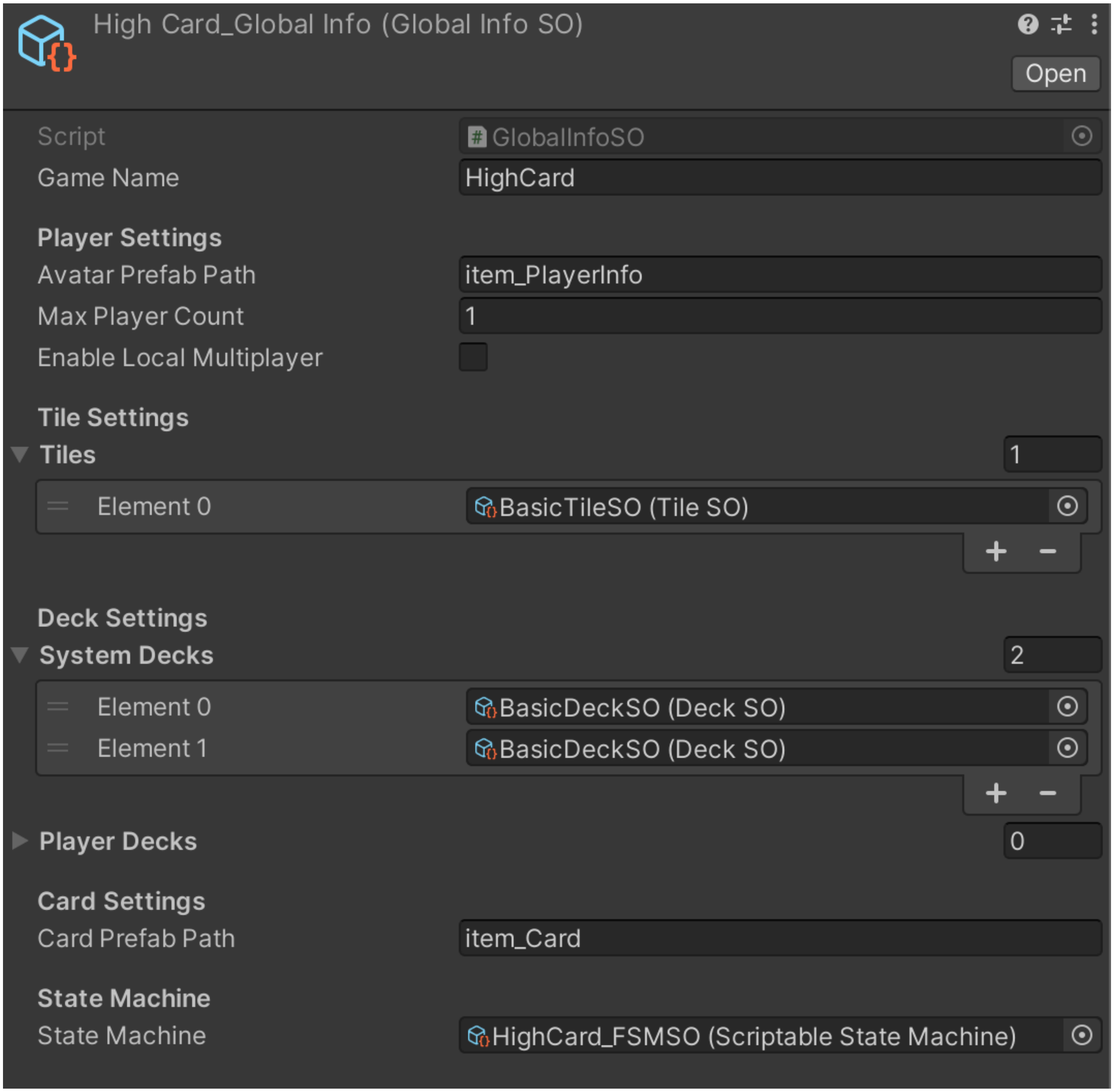
7. Wrong State

- **Overview**: If the player guessed wrong, this state triggers a dialog displaying the losing result.

Steps:

1. Add the `ShowWrongDialogAction` SO to the **Entry Actions** to display the incorrect result.

Global Info Configuration


High Card_Global Info (Global Info SO)

Open

Script	GlobalInfoSO
Game Name	HighCard
Player Settings	
Avatar Prefab Path	item_PlayerInfo
Max Player Count	1
Enable Local Multiplayer	☐
Tile Settings	
Tiles	1
Element 0	BasicTileSO (Tile SO)
Deck Settings	
System Decks	2
Element 0	BasicDeckSO (Deck SO)
Element 1	BasicDeckSO (Deck SO)
Player Decks	0
Card Settings	
Card Prefab Path	item_Card
State Machine	
State Machine	HighCard_FSMSO (Scriptable State Machine)

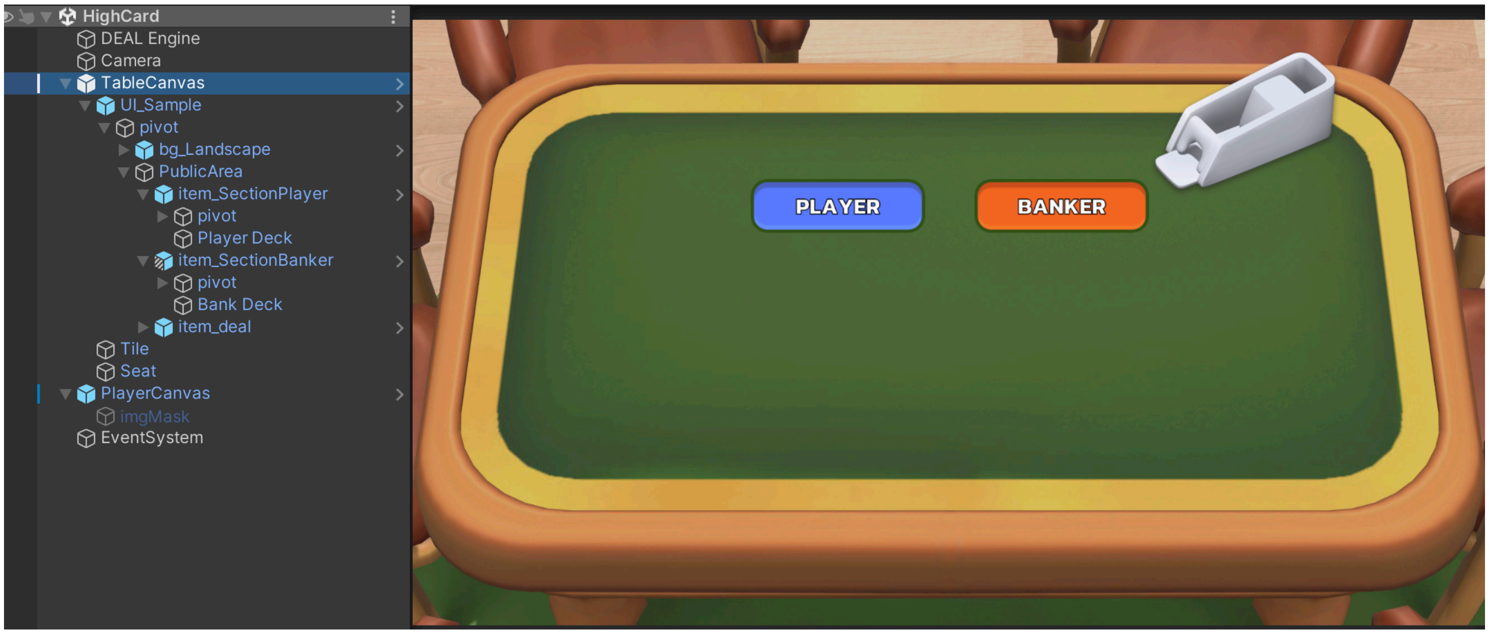
The **Global Info** Scriptable Object (SO) holds the global configuration settings for the game, including details like the game name, FSM, player settings, and any necessary global variables.

Steps:

1. Create a **Global Info** SO by right-clicking in the Project window and selecting `DealToolkit > Global Info`. Name it `HighCard_GlobalInfo`.
2. Set the **Max Player Amount** to 1 (for a single-player game).
3. Set the **Tile/Deck Settings**:
   - Use one `BasicTileSO` for drawing cards.

   - Use two `BasicDeckSO` (one for Player, one for Banker) for placing cards.
4. Set the **Avatar Prefab Path** and **Card Prefab Path**:
   - For avatars, use the `item_PlayerInfo` prefab.
   - For cards, use the `item_Card` prefab.
5. Set the **State Machine** to `HighCard_FSMSO` for the game FSM logic.

Scene Management



Steps:

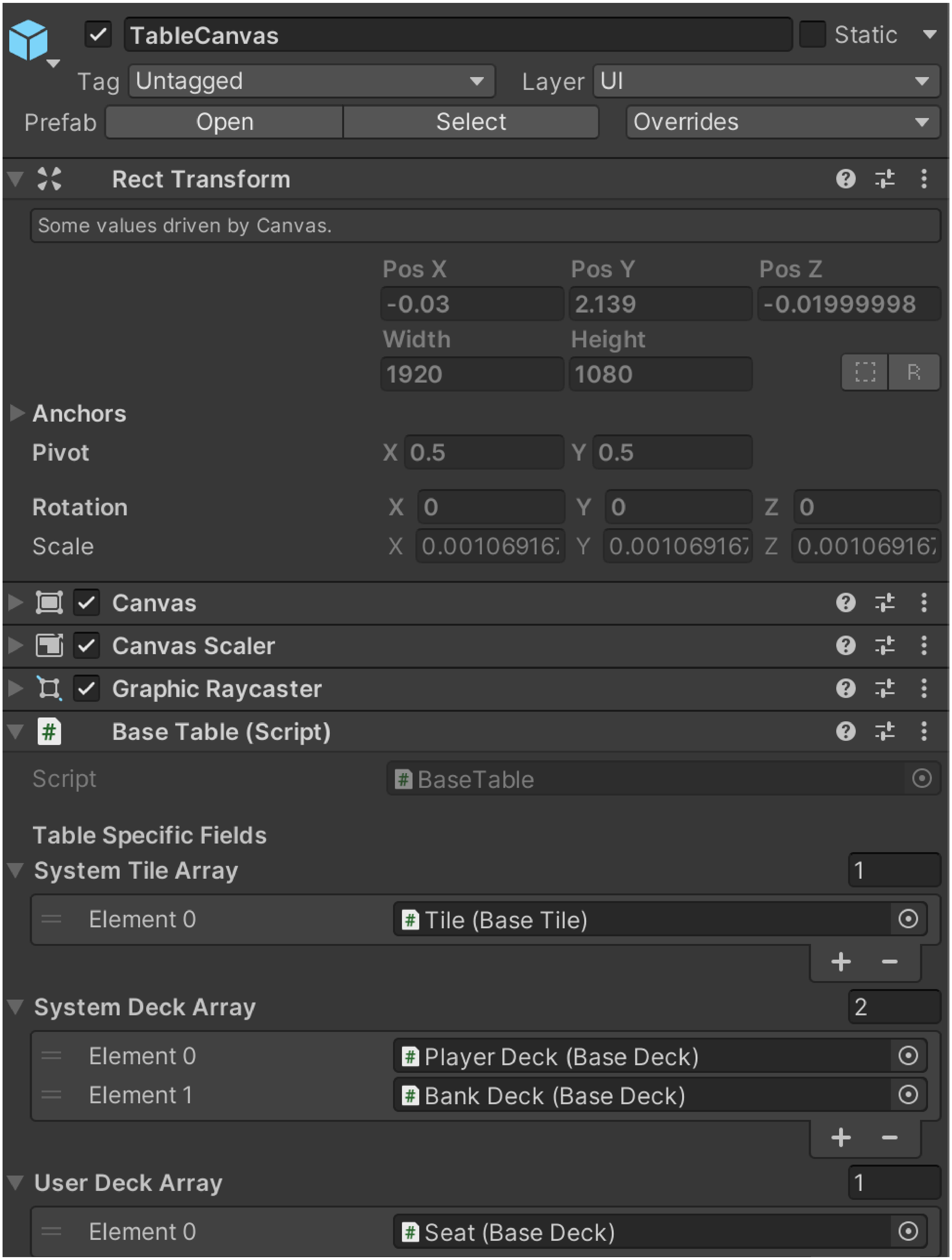
1. Create a new scene and add the following prefabs from the sample resources folder:
   - `DealEngine` Prefab
   - `Table Canvas` Prefab
   - `Player Canvas` Prefab
2. Assign the `HighCard_GlobalInfo` SO to the **Global Info SO** reference in the `DealEngine` component.
3. Assign the **Table Canvas** to the `Table` reference in the `DealEngine` component.
4. Assign the **Player Canvas** to the `Dialog Manager` reference in the `ViewManager` component.

> Congratulations! You've successfully set up the **High Card** game using the DEAL Toolkit. Your game is now ready to play, and you've learned how to configure both the FSM and the global game settings.

How to Set Up Game Table

This guide explains how to configure the **game table** in your card game using the DEAL Toolkit. By following these steps, you'll create the main table layout, which includes components like avatars, seats, tiles, and decks.

Table Canvas


TableCanvas ✓ Static

Tag: Untagged | Layer: UI

Prefab: Open | Select | Overrides

Rect Transform

Some values driven by Canvas.

	Pos X	Pos Y	Pos Z
	-0.03	2.139	-0.01999998
	Width	Height	
	1920	1080	

Anchors

Pivot: X 0.5, Y 0.5

Rotation: X 0, Y 0, Z 0

Scale: X 0.00106916, Y 0.00106916, Z 0.00106916

Canvas ✓

Canvas Scaler ✓

Graphic Raycaster ✓

Base Table (Script)

Script: BaseTable

Table Specific Fields

System Tile Array: 1

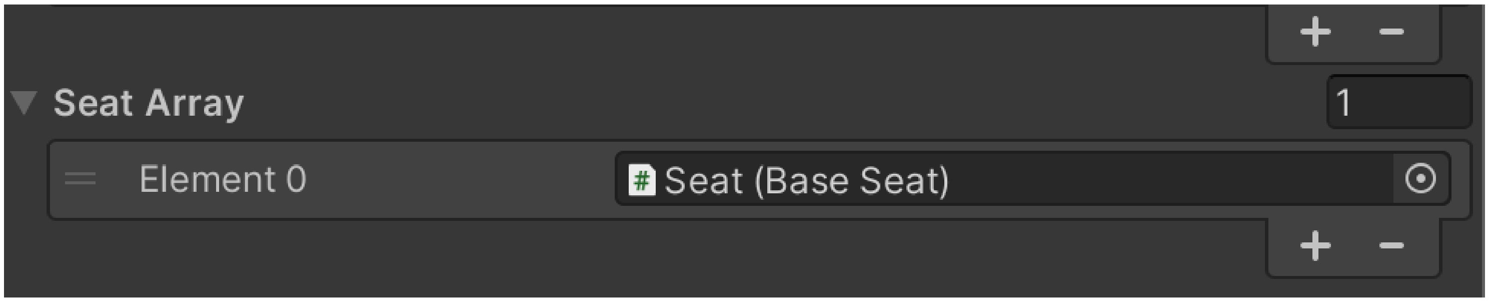
Element 0	Tile (Base Tile)

System Deck Array: 2

Element 0	Player Deck (Base Deck)
Element 1	Bank Deck (Base Deck)

User Deck Array: 1

Element 0	Seat (Base Deck)



- **Overview**: The game table consists of essential components like **avatars**, **seats**, **tiles**, and **decks**. These elements are managed by the `BaseTable` component within the **Deal Engine**.

Key Components:

1. **System Tile Array (Base Tile)**: Public card tiles on the table (e.g., community or shared tiles).
2. **System Deck Array (Base Deck)**: Public card decks on the table (e.g., shared community decks).
3. **User Deck Array (Base Deck)**: Personal card decks assigned to individual seats (e.g., player hand decks).
4. **Seat Array (Base Seat)**: A list of player seats used to join the table.

Generate Cards Data

- **Overview**: To create cards, you need to define **Scriptable Objects (SOs)**. These SOs will define the unique attributes and functionalities for each card, such as its appearance and card value.

Steps:

1. In Unity, right-click in the **Project** window and navigate to `Create > DealToolkit > Card Data`.
2. Define the properties of the cards, including the **Index**, **Suit**, and **Number**.

Argument	Description
Index	The unique index of the card, used to identify it.
Card Graphic	The visual display of the card, including both the front and back.
Card Data	The card's poker data, including suit and number.

> DEAL provides a set of basic poker card data in the `DEAL/Core/CommonSO/BasicCardSO` folder.

Generate Tile and Deck Scriptable Objects

- **Overview**: **Tiles** and **decks** are foundational elements in your card game. Tiles represent the locations on the table where cards can be placed, while decks are collections of cards that players or the system can draw from.

Steps:

1. **Create Tiles**:
    - In the **Project** window, right-click and navigate to `Create > DealToolkit > Tile Data`.
    - Edit the Tile data and link all 52 poker card data to the Tile's **Card Pool**. Alternatively, you can use the pre-existing `BasicTileSO` from the `DEAL/Core/CommonSO` folder.
2. **Create Decks**:
    - Create the **Player** and **Banker** decks by navigating to `Create > DealToolkit > Deck Data`.
    - In this specific case, no card data is required for the **Deck Card Pool**, so you can use the pre-existing `BasicDeckSO` located in the `DEAL/Core/CommonSO` folder.

Additional Notes:

- **Tile and Deck Settings**: The tiles and decks you configure are linked directly to the game's visual and functional flow. Ensure you have properly assigned the correct data (like card pools) to these objects for a smooth game setup.
- **Component Relationships**: The **BaseTable** component ensures that all elements (seats, decks, and tiles) work together seamlessly. Be sure to assign the appropriate arrays and references in the component inspector.

What is a Finite State Machine?

A **Finite State Machine (FSM)** is a widely-used game design concept and a mathematical model of computation. In essence, it's an abstract machine that can be in exactly one of a finite number of states at any given time.

Real-World Example: A Door

Consider a simple real-world example: a door. A door can be in an "opened" state or a "closed" state. The FSM helps define these states and the transitions between them.

States:

- **Opened**: The door is fully open.
- **Closed**: The door is fully closed.

Transition:

- **Condition**: A transition occurs based on an input, such as pressing a button or sensing motion. For instance, if a motion sensor detects someone approaching, it triggers the condition to transition from the "Closed" state to the "Opened" state. Conversely, if no motion is detected for a certain period, the door transitions back to the "Closed" state.

Summary:

In this example, the FSM manages the door's behavior, allowing it to switch between "Opened" and "Closed" based on specific conditions (like motion detection). Transitions dictate when and how the door moves between these states, ensuring a smooth operation based on real-world inputs.

For more details, please refer to the Wikipedia page.

Basic FSM Components in DEAL

The DEAL Toolkit's FSM is composed of four main Scriptable Objects (SO): **State Machine**, **State**, **Action**, and **Condition**. Each of these components can be fully customized to fit your game's needs.

Game designers can define any turn-based game using several states. The relationship between the components is structured as follows:
`ScriptableStateMachine -> ScriptableState -> ScriptableCondition -> ScriptableAction`

Scriptable StateMachine

This is the core FSM Scriptable Object (SO) that holds a list of transitions, each leading to different Scriptable States. The `ScriptableStateMachine` SO is the primary data source for the `CardStateMachineComponent` class, which executes the entire game flow.

> Create a new state machine SO by navigating project window and right-click: `Create > DealToolkit > FSM > StateMachine`.

Each transition element includes four variables:

1. **Origin State (ScriptableState SO)**: This is the current state (required).
2. **Condition (ScriptableCondition SO)**: This checks whether the transition should occur (optional).
3. **True State (ScriptableState SO)**: The state to transition to if the condition is met (optional).
4. **False State (ScriptableState SO)**: The state to transition to if the condition is not met (optional).

The Condition returns a boolean value, determining whether to transition to the next state. If the next state is not specified, the FSM will remain in the Origin State until a state change occurs.

Scriptable State

This is the core structure representing a state within the FSM, responsible for executing every action listed in the target `ScriptableAction[]` array.

> Create a new state SO by navigating project window and right-click: `Create > DealToolkit > FSM > State`.

Each `ScriptableState` can trigger different actions at various points:

- **OnEnterState**: Actions to be executed when entering the state.
- **OnExitState**: Actions to be executed when leaving the state.
- **OnUpdateState**: Actions to be executed during each frame while in the state.

State	**OnEnterState**	**OnExitState**	**OnUpdateState**
Action List	*entryActions*	*exitActions*	*updateActions*
Invoke Timing	When entering the state	When leaving the state	Every frame during the state

Scriptable Action

Scriptable Actions define the behaviors that should be performed in the game, such as card animations or displaying dialogs.

> Create a new action SO by navigating project window and right-click: `Create > DealToolkit > FSM > Action`.

For example, `TileDeal_1Card_PreAction` will pick one card from the tile. This action includes three variables:

Variable	**Description**
Actor	The target performer (e.g., Tile)
Actor Params	Parameters for the Actor (e.g., Tile Number)
Card Choice	Options for card count, draw rule, and visibility (Is Revealed)

Scriptable Condition

Scriptable Conditions determine whether the FSM should transition from one state to another based on specific conditions.

> Create a new condition SO by navigating project window and right-click: `Create > DealToolkit > FSM > Condition`.

For example, the `WaitChangePlayer_Condition` will trigger a state transition upon receiving the `ChangePlayer` event.

Variable	**Description**
Event Name	The event that triggers the condition (e.g., ChangePlayer)

Common Usage

The Card State Machine is a system that manages the actions ScriptableObject taken during a card game.

A `ScriptableStateMachine` consists of a list of `ScriptableState` and `StateTransition` SOs. Each `ScriptableState` contains a list of `ScriptableAction` SOs that are executed when the state is entered, exited, or updated. Each `StateTransition` contains a `ScriptableCondition` SO that checks if the transition should be made from the origin state to the true state or false state.

Default Action Types

- `ChangePlayerAction`: Rotate to the player specified by `ActionActor` enum and params
- `DealAction`, `DrawAction`: Deal means actor will give out card to target, Draw means actor will pick card from target.
- `TrasnsferAction`: Similar to Deal/Draw, but for transferring chips.
- `PreAction`, `PostAction`: During PreAction, the action will acquire the involved decks and players, then request `CardStateMachineComponent` to WaitTurnLock until the actor has made a valid `CardChoice`, which will then be executed by PostAction.
- `CardChoice`: Serializable structure that can be set to request the state machine to host a **Card Pick**.
    - `count`: How many cards should be picked
    - `rule`: Enum CardChoiceRule that records the pick rules (like *FromTop*, *Random*, *Specific*)
    - `isRevealedToPicker`: Should the picker see the cards when picking?
- `EliminatePlayerAction`: Eliminate the player from the game
- `ModifyPlayerPropertyAction`: Modify the target player's property with reference player's property
- `UpdatePlayerPropertyAction`: Update the target player's property with the given value
- `ShowDialogAction`: Pop up a dialog to selected players
- `UpdateStateEventAction`: Update the state machine's state

Default Condition Types

- `EndOfTurnCondition`: check if state is locked by WaitTurnLock
- `EndOfSystemLockCondition`: check if state is locked by WaitSystemLock
- `AlwaysTrueCondition`: always passes, used for transition phase
- `CheckDeckCountCondition`: checks if given actor has given amount of cards, normally used in intial deal
    - Example usage: Checks if all players have 5 cards, so that the game can start, else continue dealing cards to players
- `CheckDefeatedCondition`: checks if the specific player has been defeated

- `CheckGameOverCondition`: set the amount of `maxEliminatedPlayerCount`, if the amount of eliminated players is equal to or greater than `maxEliminatedPlayerCount`, the game is over
- `CheckPlayerPropertyCondition`: check target player's property against the given value `rawPropertyValue`
- `ComparePlayerPropertyCondition`: compare the target player's property with the reference player's property
- `CompareScoreCondition`: compare the subject player's card score with the control player's card score
  - `XXXCheckScale`: check only the cards from player's buffer, or check the whole deck
    - Buffer: When player selects one or many cards from their deck, but hasn't perform any actual transfer yet (for example: about to deal), these cards are temporarily placed in the buffer
  - `XXXCounter`: method to count the scores
    - `SumNumber`: sum all the card numbers
    - `Max`: find the max number
    - `Min`: find the min number
    - `MostOccuredNumber`: find the most occurred number
    - `Average`: calculate the average number
    - `Mid`: find the middle number
    - `GivenMinusSumNumber`: given score (stored in `CounterParams`) minus the sum of all cards
    - `PokerEval`: evaluate the poker hand using external `PHEval` library
    - `BaccarateRangeSum`: sum the card numbers using Baccarate rules
- `MatchScoreMultipleCondition`: check subject player's card score against a group set of rules
  - Example usage: Check if the player's score is larger than 3 and smaller than 6
- `WaitEventCondition`: wait until a state event is triggered locally
- `WaitForGroupEventCondition`: wait until a state event is triggered for all members in selected group

Extend usage

If you feel like customizing the action or condition, you can create your own `ScriptableAction` or `ScriptableCondition` SOs. Check out the `ExtendFSM` article for more information.

General Number Card Game Mechanism

Please refer to wiki for example games:

- Texas hold 'em
- Blackjack
- Big Two
- Sevens
- Rummikub
- Uno

Card Tile

The Card Tile is an essential component of card games. It determines the available cards for dealing and gameplay. Key aspects of the Card Tile include:

- Card Pool: The total number of cards in the deck and the specific cards included.
- Dealing Order: The sequence in which cards are distributed and which player to deal next.
- Card Drawing: The number of cards can be drawn by each player during the game.

Types of Card Tile:

	Uno	**Texas Hold'em**	**Rummikub**
Pool	108	1 Set without Jokers	2 Sets with 2 Joker
Dealing Order	Transpose	Fixed	Fixed
Card Drawing	Single/Multiple	Single	Single

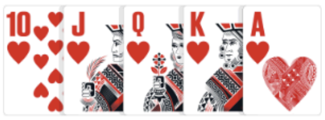
Hand Deck

The hand Deck refer to the cards held by individual players during the game. Key aspects of hand cards include:

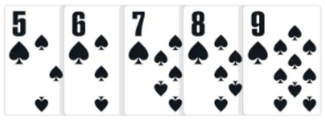
- Cards in the Player's Hand: The number of cards each player can hold.
- Deck Sorting: The order in which cards are arranged in the hand. This can vary based on game rules.

- Poker Pattern Types:

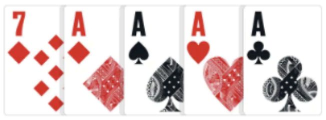
Royal Flush



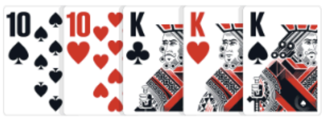
Straight Flush



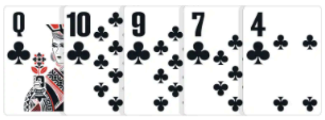
Four of a Kind



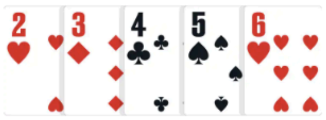
Full House



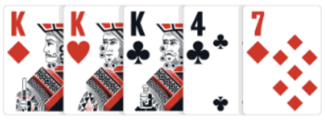
Flush



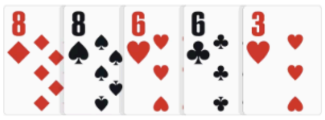
Straight



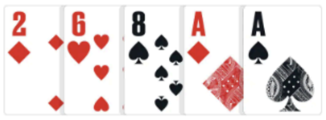
Three of a Kind



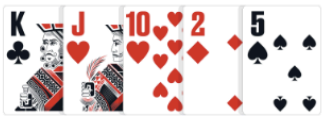
Two Pair



One Pair



High Card



Types of Hand Cards:

	Big Two	**Texas Hold'em**	**Rummikub**
Amount	13	2	14
Deal Type	Pattern	None	Single
Usage Scenarios	Compare Pattern	Final hand evaluation	Sets Collection

Showdown Pile

The showdown pile refers to the designated area where cards are revealed and placed after the final betting round in a poker game. It is where players showcase their hands and determine the winner. Key aspects of the showdown pile include:

- Types of Piles:
  - Community Pile: A pile where the community cards, also known as the board cards, are placed.
  - Player Piles: Individual piles where each player reveals and displays their hole cards.

Types of Settlement Piles:

	BlackJack	**Sevens**	**Texas Hold'em**
Community Pile	None	Multiple (e.g., 4 suits)	5 community cards on board
Player Piles	Single	Single	2 hole cards per player
Usage Scenarios	Scoring	Link/Scoring	Final hand evaluation

Player Behavior

The player behavior refers to the actions that players can take during the game. Key aspects of player behavior include:

Behavior	**Description**
Draw	Players draw cards from the Tile, Deck, or Pile..
Deal	Place a single card or a set of pattern cards into a Tile, Deck, or Pile.
Discard	Players choose to discard or abandon the cards they own.
Pass	Players decide to give up or skip the current round.

Beting Action

For Extra Player Interaction, the game can be designed with betting actions. Take the `Texas Hold'em` as an example, the common betting process is as follows:

Type	**Timing**	**Action Description**
Blinds	Before the game starts	A Player place mandatory bets called blinds.

Type	Timing	Action Description
Check	During a betting round	A player chooses to pass the action to the next player without placing a bet.
Bet	During a betting round	A player places chips into the pot, setting the amount for other players to match.
Call	During a betting round	A player matches the current bet to stay in the hand.
Raise	During a betting round	A player increases the current bet, forcing other players to match the new higher bet.
Fold	During a betting round	A player discards their cards and forfeits any bets made previously.