# Agent(AgentState)

An agent is a state corresponding to a player's account, containing a list of addresses for the avatars they own.

- AgentState

## State

- Account Address: Addresses.Agent
- State Address: The address of the private key used to play the game.

### Get State:

```
public AgentState? GetAgentState(IWorld world, Address address)
{
    IAccount account = world.GetAccount(Addresses.Agent);
    if (account is null)
    {
        return null;
    }

    IValue state = account.GetState(address);
    return state switch
    {
        Bencodex.Types.List l => new AgentState(l),
        Bencodex.Types.Dictionary d => new AgentState(d),
        _ => null,
    };
}
```

# Avatar(AvatarState)

An avatar is a state that corresponds to a character, containing their name, level, and more.

- AvatarState

## State

- Account Address: Addresses.Avatar
- State Address: The avatar's address is derived from the agent's address.

```
public Address GetAvatarAddress(Address agentAddress, int index)
{
    return agentAddress.Derive($"avatar-state-{index}");
    // or
    return Addresses.GetAvatarAddress(agentAddress, index);
}
```

### Get State:

```
public AvatarState? GetAvatarState(IWorld world, Address address)
{
    IAccount account = world.GetAccount(Addresses.Avatar);
    if (account is null)
    {
        return null;
    }

    IValue state = account.GetState(address);
    return state switch
    {
        Bencodex.Types.List l => new AvatarState(l),
        Bencodex.Types.Dictionary d => new AvatarState(d),
        _ => null,
    };
}
```

# Inventory

Inventories exist for each avatar and contains a variety of items.

- Inventory

## State

- Account Address: Addresses.Inventory
- State Address: Use the address of avatar as it is.

### Get State:

```
public Inventory? GetInventory(IWorld world, Address address)
{
    IAccount account = world.GetAccount(Addresses.Inventory);
    if (account is null)
    {
        return null;
    }

    IValue state = account.GetState(address);
    return state switch
    {
        Bencodex.Types.List l => new Inventory(l),
        _ => null,
    };
}
```

# QuestList

QuestList exists for each avatar and contains the avatar's quest information.

- QuestList

## State

- Account Address: Addresses.QuestList
- State Address: Use the address of your avatar as it is.

### Get State:

```
public QuestList? GetQuestList(IWorld world, Address address)
{
    IAccount account = world.GetAccount(Addresses.QuestList);
    if (account is null)
    {
        return null;
    }

    IValue state = account.GetState(address);
    return state switch
    {
        Bencodex.Types.List l => new QuestList(l),
        Bencodex.Types.Dictionary d => new QuestList(d),
        _ => null,
    };
}
```

# WorldInformation

WorldInformation exists for each avatar and contains information about the avatar's adventures.

- WorldInformation

## State

- Account Address: Addresses.WorldInformation
- State Address: Use the address of your avatar as it is.

### Get State:

```
public WorldInformation? GetWorldInformation(IWorld world, Address address)
{
    IAccount account = world.GetAccount(Addresses.WorldInformation);
    if (account is null)
    {
        return null;
    }

    IValue state = account.GetState(address);
    return state switch
    {
        Bencodex.Types.Dictionary d => new WorldInformation(d),
        _ => null,
    };
}
```

# Adventure

Adventure content is available to players as a first step in developing their avatars. For a detailed description of Adventure content, see the official documentation.

## World

The adventures of the Nine Chronicles begin in Yggdrasil, and there are new worlds after that, in the following order: Alfheim, Svartalfheim, Asgard, and so on. This data comes from the `WorldSheet`.

- Nekoyume.TableData.WorldSheet

You can use the 9c-board service to view the `WorldSheet`.

- https://9c-board.nine-chronicles.dev/odin/tablesheet/WorldSheet
- https://9c-board.nine-chronicles.dev/heimdall/tablesheet/WorldSheet

Yggdrasil, where you begin your adventure, is open when you create your avatar. Subsequent worlds are unlocked by meeting certain conditions. More information can be found in the `WorldUnlockSheet`.

- Nekoyume.TableData.WorldUnlockSheet

Like `WorldSheet`, you can check it out in 9c-board.

- https://9c-board.nine-chronicles.dev/odin/tablesheet/WorldUnlockSheet
- https://9c-board.nine-chronicles.dev/heimdall/tablesheet/WorldUnlockSheet

## Stage Challenges and Rewards

Each world contains multiple stages, each with different rewards for completing them. Stage information can be found in the `StageSheet`.

- Nekoyume.TableData.StageSheet
- https://9c-board.nine-chronicles.dev/odin/tablesheet/StageSheet
- https://9c-board.nine-chronicles.dev/heimdall/tablesheet/StageSheet

### Stage Challenge Cost

When you look at the `StageSheet`, there is a field called `CostAP`. This field represents the number of action points required to progress through the stage. Players can spend their avatar's action points to progress through the adventure.

## Stage Waves

A stage consists of several waves, each of which must be cleared before moving on to the next wave. Wave information can be found in the `StageWaveSheet`.

- Nekoyume.TableData.StageWaveSheet
- https://9c-board.nine-chronicles.dev/odin/tablesheet/StageWaveSheet
- https://9c-board.nine-chronicles.dev/heimdall/tablesheet/StageWaveSheet

Nine Chronicles currently has all stages set to three waves. The number of waves your avatar completes determines the number of stars you earn in the stage. The number of stars also determines the rewards, as shown below.

- 1 star: EXP
- 2 stars: EXP, items
- 3 stars: EXP, items, unlock next stage

# Stage Buff

Stage buffs are buffs that the player can receive when progressing through a particular stage. These buffs can be found in the `CrystalRandomBuffSheet` and `CrystalStageBuffGachaSheet`.

- Nekoyume.TableData.Crystal.CrystalRandomBuffSheet
- Nekoyume.TableData.Crystal.CrystalStageBuffGachaSheet

# Item Slots

There are item slots available to you as you progress through your adventure. You can equip different items in these slots to give you an advantage in combat.
Items that can be used in an adventure include equipment, costumes, and food.

# Rune Slots

As you progress through your adventure, you can equip runes to gain various effects.

# Repeat Battle

Adventures can be repeated, and AP potions can be used to do so.

# Battle(StageSimulator)

Battle is turn-based, and players can use various items with their avatars to defeat enemies. Adventures utilize the StageSimulator class to run battle. This class manages the flow of battle and handles the events that occur on each turn.

When a battle begins, it creates a player avatar.

- Nekoyume.Model.BattleStatus.SpawnPlayer

## Wave

When a battle starts, the first wave begins. If you don't clear the wave, the battle ends, if you do, you move on to the next wave, and if you clear the last wave, the battle ends.

- Nekoyume.Model.BattleStatus.SpawnWave

## Turn

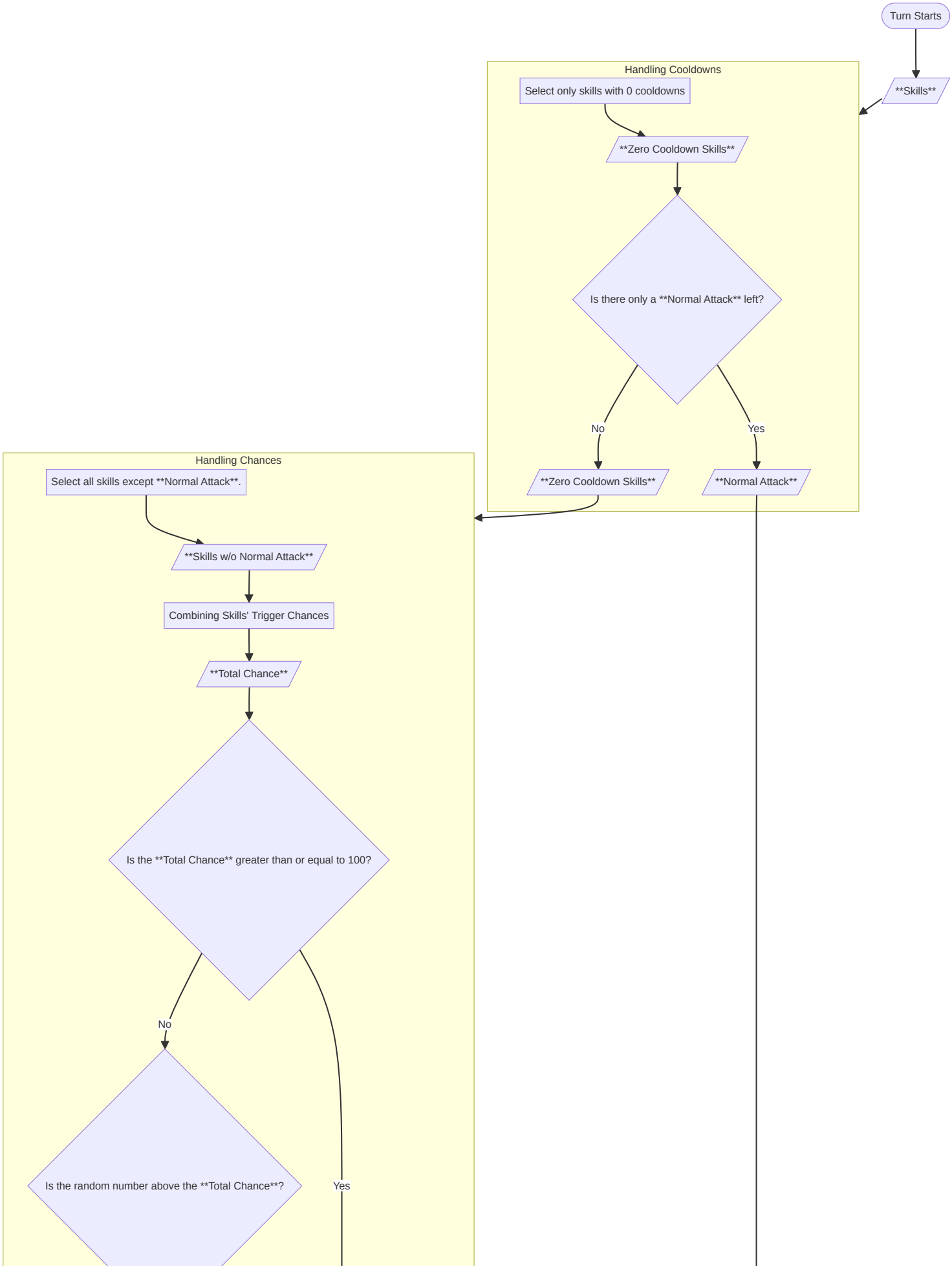
When a wave starts, a formula is used to give the highest priority character a turn, which is determined by the character's SPD(Speed) stat.

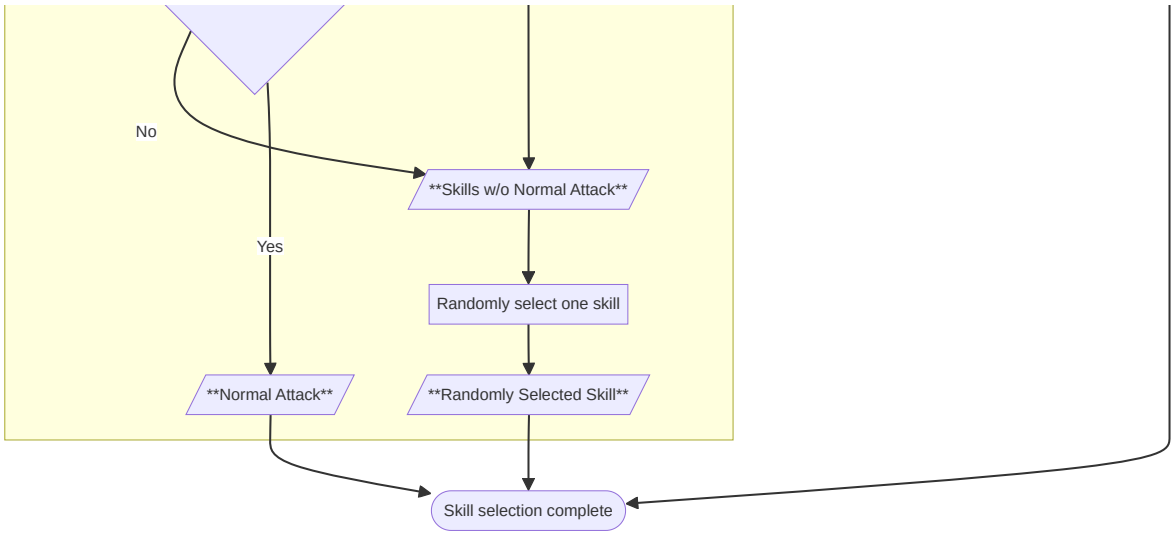
1. Wave starts.
2. The player avatar has a priority of **100[^simulator-turn-priority] / avatar's SPD stat**. The number `100` in the above formula, is the value used in the formula for determining turn priority in the simulator and can be found in Nekoyume.Battle.Simulator.TurnPriority.
3. All enemies in the wave have a priority of **100 / enemy's SPD stat**.
4. Give the turn to the character with the highest priority, i.e., the smallest **100 / SPD stat**.
5. The character granted a turn in (4) takes the turn.
6. Increase the priority of characters that were not granted a turn in (4).
   - Multiply by `0.9` if the character is a player and used a non-Basic Attack skill in the previous turn.
   - Otherwise, multiply by `0.6`.
7. Characters granted a turn in (4) again have a priority of **100 / character SPD stat**.
8. Repeat steps (2) through (7) until the player avatar dies or the wave is cleared.
9. Wave ends. Repeat from (1) for the next wave if one exists.

## Skill Activation

All characters, including avatars and enemies, can activate skills. Skills include a variety of offensive and buff skills, including **Normal Attack**. Which skill is activated is determined by the activation chance of each skill.

::: details Skill Activation Flowchart


Turn Starts
**Skills**
Handling Cooldowns
Select only skills with 0 cooldowns
**Zero Cooldown Skills**
Is there only a **Normal Attack** left?
No
Yes
**Zero Cooldown Skills**
**Normal Attack**
Handling Chances
Select all skills except **Normal Attack**.
**Skills w/o Normal Attack**
Combining Skills' Trigger Chances
**Total Chance**
Is the **Total Chance** greater than or equal to 100?
No
Yes
Is the random number above the **Total Chance**?




:::

**Skill Types**

- Nekoyume.Model.BattleStatus.NormalAttack: This is the **Normal Attack**.
- Nekoyume.Model.BattleStatus.BlowAttack
- Nekoyume.Model.BattleStatus.DoubleAttack
- Nekoyume.Model.BattleStatus.DoubleAttackWithCombo
- Nekoyume.Model.BattleStatus.AreaAttack
- Nekoyume.Model.BattleStatus.BuffSkill
- Nekoyume.Model.BattleStatus.HealSkill
- Nekoyume.Model.BattleStatus.BuffRemovalAttack
- ...

# Normal Attack Hits

Normal Attack is also treated as skills in Nine Chronicles, and whether or not they hit is largely determined by the attacker's and defender's levels and HIT stats, with a few other formulas.

- In the Nekoyume.Model.Skill.AttackSkill.ProcessDamage method, get the hit status of the **Normal Attack**: `target.IsHit(caster)`.
- The `IsHit` method above points to Nekoyume.Model.CharacterBase.IsHit(CharacterBase), which is overridden by the player character (Nekoyume.Model.Player) and so on.
- A quick look at the logic of hits in adventures:
  - When the defender is a player character, they cannot dodge an attacker's attack.
  - If the attacker is under the effect of a Focus buff(Nekoyume.Model.Buff.Focus), it will hit 100% of the time.

    - Otherwise, the hit is handled according to the result of the Nekoyume.Battle.HitHelper.IsHit method.

**Nekoyume.Battle.HitHelper.IsHit**

1. Get a value `A` between `-5 ~ 50` with `(attacker's level - defender's level)`.
2. Use `(Attacker's HIT stat * 10000 - Defender's HIT stat * 10000 / 3) / Defender's HIT stat / 100` to get a value `B` between `0 ~ 50`.
    - If the attacker's and defender's HIT stats are less than or equal to 0, then it is scaled by 1.
3. Use `A + B` to get a value `C` between `10 ~ 90`.
4. Find a random number `D` between `0 ~ 99`.
5. finally, if `C` is greater than or equal to `D`, treat it as a hit.

# Combo

Avatar's attacks can have combo effects.

- The maximum number of combos depends on the avatar's level. For example, at level 1, you can get up to 2 combos, and at level 250, you can get up to 5 combos.
- If you fulfill the combo increase condition after getting the maximum combo, you will start with 1 combo.

## Increase Combo

- Successfully executes a normal attack.
- Successfully executes a skill with the `SkillSheet.Combo` field set to `true`.

## Reset Combo

- Normal attack fails.

## Combo Effect

- Increases the damage of normal attack or skills with a `SkillSheet.Combo` field of `true`.
    - Nekoyume.Model.CharacterBase.GetDamage
    - Nekoyume.Battle.AttackCountHelper.GetDamageMultiplier
- Gain additional critical chance based on a condition.
    - Nekoyume.Model.CharacterBase.IsCritical
    - Nekoyume.Battle.AttackCountHelper.GetAdditionalCriticalChance

# Related actions

A list of actions associated with Adventure content:

- DailyReward: You get the action points you need to execute the action.

- HackAndSlash: The default adventure action.
- HackAndSlashSweep: This action clears stages with 3 stars.

# Arena

Once you've developed your avatar through the Adventure, you'll have access to Arena content where you can compete against other players. For more information, see the official documentation.

## Round

The Arena operates in rounds. Each round lasts for a certain amount of block range and is of one of the following types: seasonal, offseason, or championship.

- Championship ID the round belongs to: Nekoyume.TableData.ArenaSheet.Row.ChampionshipId
- Round number within a championship ID: Nekoyume.TableData.ArenaSheet.Row.Round
- Type of the round: Nekoyume.TableData.ArenaSheet.Row.ArenaType
- Duration of the round:
  - Start: Nekoyume.TableData.ArenaSheet.Row.StartBlockIndex
  - End: Nekoyume.TableData.ArenaSheet.Row.EndBlockIndex

## Join

To join a round, use the Nekoyume.Action.JoinArena action.
Each round has a requirement or cost to join, which you can see in the table below.

| | offseason | season | championship | ref |
|---|---|---|---|---|
| Conditions of entry | X | X | O | Nekoyume.TableData.ArenaSheet.Row |
| Entry Fee | X | O | O | Nekoyume.TableData.ArenaSheet.I |

## Conditions of Entry

All Championship rounds have their own medal items that must be collected in a certain amount to be eligible to participate in each Championship Round.

## Entry Fee

To participate in Seasons and Championship rounds, players must pay an entry fee in crystals that is proportional to the level of the participating avatar.

## States {#join-states}

#### List of round participants

- Account Address: Libplanet.Action.State.ReservedAddresses.LegacyAccount
- State Address: This will have a separate address based on the Championship ID and round.

```
public Address GetArenaParticipantsAddress(int championshipId, int round)
{
    return ArenaParticipants.DeriveAddress(championshipId, round);
}
```

- Type: Nekoyume.Model.Arena.ArenaParticipants

#### Participant's round information

- Account Address: Libplanet.Action.State.ReservedAddresses.LegacyAccount
- State Address: This will have a separate address based on the Championship ID and round and your avatar's address.

```
public Address GetArenaInformation(Address avatarAddress, int championshipId,
int round)
{
    ArenaInformation.DeriveAddress(avatarAddress, championshipId, round);
}
```

- Type: Nekoyume.Model.Arena.ArenaInformation

#### Participant's score

- Account Address: Libplanet.Action.State.ReservedAddresses.LegacyAccount
- State Address: This will have a separate address based on the Championship ID and round and your avatar's address.

```
public Address GetArenaScore(Address avatarAddress, int championshipId,
int round)
{
    ArenaScore.DeriveAddress(avatarAddress, championshipId, round);
}
```

- Type: Nekoyume.Model.Arena.ArenaScore

## Battle(ArenaSimulator) {#battle}

The battle uses the Nekoyume.Action.BattleArena action and utilizes the Nekoyume.Arena.ArenaSimulator class to perform the battle.

## Tickets {#battle-tickets}

Arena tickets are required to fight battles. When you first join a round, you are automatically issued the maximum number of tickets. These tickets are issued anew to participants in that round at each round's ticket reset cycle.

- Ticket maximum: Nekoyume.Model.Arena.ArenaInformation.MaxTicketCount
- Ticket reset interval: Nekoyume.Model.State.GameConfigState.DailyArenaInterval

**Buy Tickets**

If you run out of tickets, you can purchase them directly to use in battle. There is a maximum amount and price of tickets that can be purchased within a round or ticket reset cycle.

- Maximum number of tickets you can purchase within a round: Nekoyume.TableData.ArenaSheet.RoundData.MaxPurchaseCount
- Maximum number of tickets you can purchase within a ticket reset cycle: Nekoyume.TableData.ArenaSheet.RoundData.MaxPurchaseCountWithInterval
- Ticket purchase price: The purchase price of tickets is determined by a set price for each round and the number of tickets already purchased, as shown in the code below.

```
public FungibleAssetValue GetTicketPrice(
    ArenaSheet.RoundData round,
    int alreadyPurchasedCount)
{
    return GetTickgetPrice(
        round.TicketPrice,
        round.AdditionalTicketPrice,
        alreadyPurchasedCount);
}

public FungibleAssetValue GetTicketPrice(
    decimal price,
    decimal additionalprice,
    int alreadyPurchasedCount)
{
    return price.DivRem(100, out _) +
        additionalprice.DivRem(100, out _) * alreadyPurchasedCount;
}
```

## Battle Target Limit

In Arena, there is a limit to who you can battle. This is the score limit, see the code below.

https://github.com/planetarium/lib9c/blob/1.17.3/Lib9c/Action/BattleArena.cs#L274

```
if (!ArenaHelper.ValidateScoreDifference(
    ArenaHelper.ScoreLimits,
    roundData.ArenaType,
    myArenaScore.Score,
    enemyArenaScore.Score))
{
    // ...
}
```

In the code above, the ArenaHelper.ValidateScoreDifference method validates the difference between the player's and enemy's scores to determine if a battle is possible. It takes in ArenaHelper.ScoreLimits, the arena type from the round data, and the player's and enemy's scores as arguments to determine if the score difference is within an acceptable range.

The current score limits are shown in the table below.

| attacker score - defender Score | offseason | season | championships | |
|---|---|---|---|---|
| minimum score difference | - | -100 | -100 | -100 |
| max score difference | - | 200 | 200 | 200 |

For example, when fighting in the Offseason round, you can challenge anyone to a fight with no score limit, while in the Season or Championship round, if your player avatar has a score of 2000, the opponent you can challenge must have a score between 1900 and 2200.

## Rules(different from the Adventure) {#battle-rule}

Unlike the Adventure, Arena battles are fought between players(PvP). The basic battle rules are the same as for the Adventure, but see below for the differences.

**Attack skill hits**.

Battles in the Arena follow different rules than in Adventure when determining whether an attack skill hits.

- See also: Adventure > Normal Attack Hits
- Determines the hit of all attack skills, not just **normal attack**.
- Does not take into account the level difference between attacker and defender.

| Content | Hit or Miss Coverage | Variables |
|---|---|---|
| Arena | All attack skills, including **normal attack** | HIT stat |
| Adventure | **Normal Attack** | Avatar Level, HIT Stat |

Below is the order in which hits are determined in the Arena.

- Get the hit status of an attack skill in the Nekoyume.Model.Skill.ArenaAttackSkill.ProcessDamage method: `target.IsHit(caster)`.
- The `IsHit` method above points to Nekoyume.Model.ArenaCharacter.IsHit(ArenaCharacter).
- If we take a quick look at the logic of a hit in the Arena:
  - If the attacker is under the effect of a focus buff(Nekoyume.Model.Buff.Focus), it will hit 100% of the time.
    - Otherwise, handle hits based on the result of the Nekoyume.Battle.HitHelper.IsHitWithoutLevelCorrection method.

# Rewards

Rewards in Arena battles are divided into base rewards and victory rewards.

### Base Reward

Base Rewards are determined by Avatar's Arena Score and range in quantity and type based on Avatar's level.

- Rewards list:
  - Nekoyume.TableData.WeeklyArenaRewardSheet
  - https://9c-board.nine-chronicles.dev/odin/tablesheet/WeeklyArenaRewardSheet
- Number of rewards per Arena Score: Check out the Nekoyume.Arena.ArenaHelper.GetRewardCount method. | Score | 1000 | 1001~ | 1100~ | 1200~ | 1400~ | 1800~ | | :-: | :-: | :-: | :-: | :-: | :-: | :-: | | Number of rewards | 1 | 2 | 3 | 4 | 5 | 6 |

### Victory Rewards(Medals)

When you win an Arena battle, you receive one medal for the Championship that the round belongs to. For more information, see the Nekoyume.Action.BettleArena action and the Nekoyume.Arena.ArenaHelper.GetMedal method.

# Score

When you join a round, you start with a score of 1000 points(Nekoyume.Model.Arena.ArenaScore.ArenaScoreDefault).
After that, the score changes through the battle, which is affected by the difference between the attacker's and defender's scores. For more information, see the Nekoyume.Action.BettleArena action and the Nekoyume.Arena.ArenaHelper.GetScores method.

# Ranking

The ranking is determined by the score in the arena. It's important to note that the ranking is not handled by the blockchain state, but by an external service.

### Handle Ties

In Nine Chronicles' Arena Rankings, ties are broken by bundling them into a lower ranking. For example, if the highest score in a particular round is 2000 points and there are three ties, they will all be treated as third place.

# Actions

List of actions related to the arena:

- JoinArena: An action to join a specific round in the Arena.
- BattleArena: An action in which you battle other avatars in a specific round of the Arena.

# Account

Since Lib9c is implemented by using Libplanet, Lib9c has several accounts according to Libplanet's account state model. They are listed below and you can use them to query states via NineChronicles.Headless GraphQL, RPC, or something else.

| **Name (`Addresses.*`)** | **Address** |
|---|---|
| Shop | 0x0000000000000000000000000000000000000000 |
| Ranking | 0x0000000000000000000000000000000000000001 |
| WeeklyArena | 0x0000000000000000000000000000000000000002 |
| TableSheet | 0x0000000000000000000000000000000000000003 |
| GameConfig | 0x0000000000000000000000000000000000000004 |
| RedeemCode | 0x0000000000000000000000000000000000000005 |
| Admin | 0x0000000000000000000000000000000000000006 |
| PendingActivation | 0x0000000000000000000000000000000000000007 |
| ActivatedAccount | 0x0000000000000000000000000000000000000008 |
| Blacksmith | 0x0000000000000000000000000000000000000009 |
| GoldCurrency | 0x000000000000000000000000000000000000000a |
| GoldDistribution | 0x000000000000000000000000000000000000000b |
| AuthorizedMiners | 0x000000000000000000000000000000000000000c |
| Credits | 0x000000000000000000000000000000000000000d |
| UnlockWorld | 0x000000000000000000000000000000000000000e |
| UnlockEquipmentRecipe | 0x000000000000000000000000000000000000000f |
| MaterialCost | 0x0000000000000000000000000000000000000010 |
| StageRandomBuff | 0x0000000000000000000000000000000000000011 |
| Arena | 0x0000000000000000000000000000000000000012 |

| Name (`Addresses.*`) | Address |
| --- | --- |
| SuperCraft | 0x0000000000000000000000000000000000000013 |
| EventDungeon | 0x0000000000000000000000000000000000000014 |
| Raid | 0x0000000000000000000000000000000000000015 |
| Rune | 0x0000000000000000000000000000000000000016 |
| Market | 0x0000000000000000000000000000000000000017 |
| GarageWallet | 0x0000000000000000000000000000000000000018 |
| AssetMinters | 0x0000000000000000000000000000000000000019 |
| Agent | 0x000000000000000000000000000000000000001a |
| Avatar | 0x000000000000000000000000000000000000001b |
| Inventory | 0x000000000000000000000000000000000000001c |
| WorldInformation | 0x000000000000000000000000000000000000001d |
| QuestList | 0x000000000000000000000000000000000000001e |
| Collection | 0x000000000000000000000000000000000000001f |
| DailyReward | 0x0000000000000000000000000000000000000020 |
| ActionPoint | 0x0000000000000000000000000000000000000021 |
| RuneState | 0x0000000000000000000000000000000000000022 |
| Relationship | 0x0000000000000000000000000000000000000023 |
| CombinationSlot | 0x0000000000000000000000000000000000000024 |
| ClaimedGiftIds | 0x0000000000000000000000000000000000000025 |
| AdventureBoss | 0x0000000000000000000000000000000000000100 |
| BountyBoard | 0x0000000000000000000000000000000000000101 |
| ExploreBoard | 0x0000000000000000000000000000000000000102 |

| **Name (`Addresses.*`)** | **Address** |
| --- | --- |
| ExplorerList | 0x0000000000000000000000000000000000000103 |
| Guild | 0x0000000000000000000000000000000000000200 |
| GuildMemberCounter | 0x0000000000000000000000000000000000000201 |
| GuildApplication | 0x0000000000000000000000000000000000000202 |
| GuildParticipant | 0x0000000000000000000000000000000000000203 |
| EmptyAccountAddress | 0xffffffffffffffffffffffffffffffffffffffff |