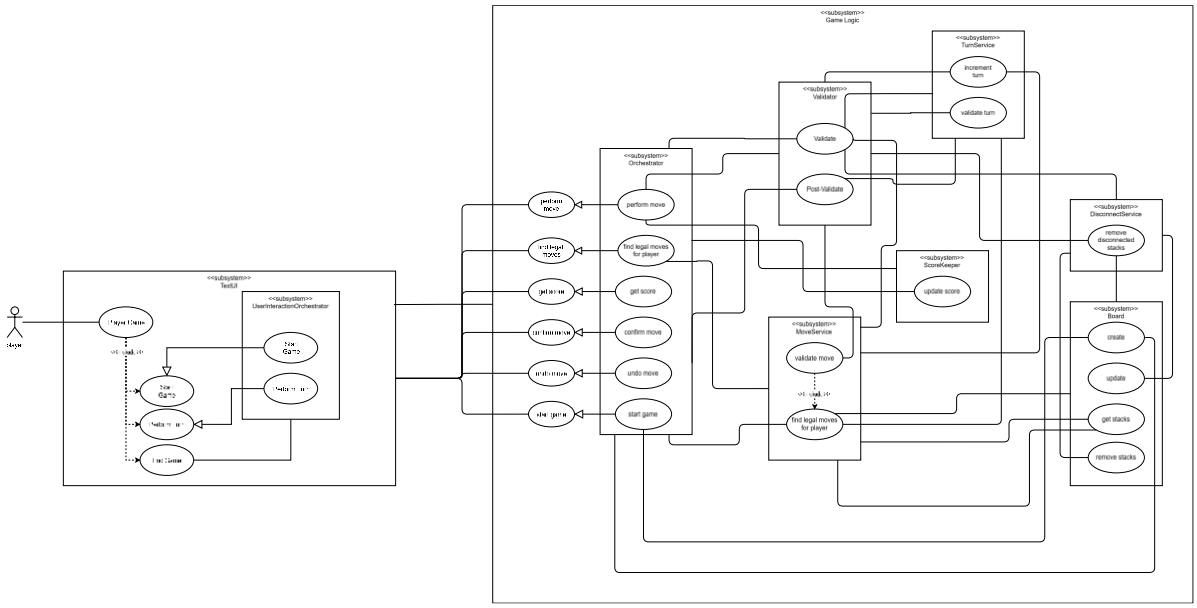
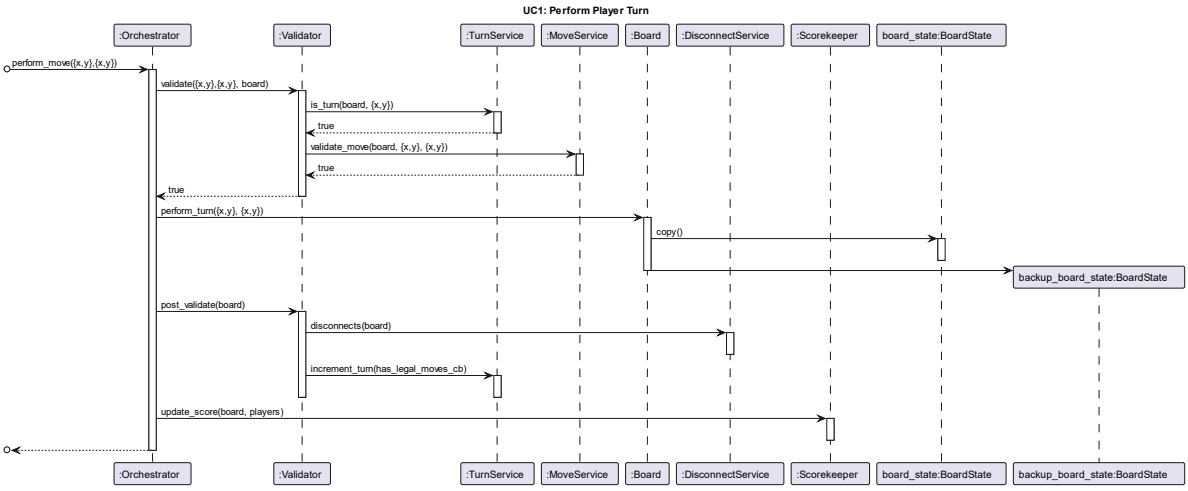
# Group 10: DVONN Milestone 2

# Use Case Diagram



# Sequence Diagrams

## Use Case 1: Perform Player Turn

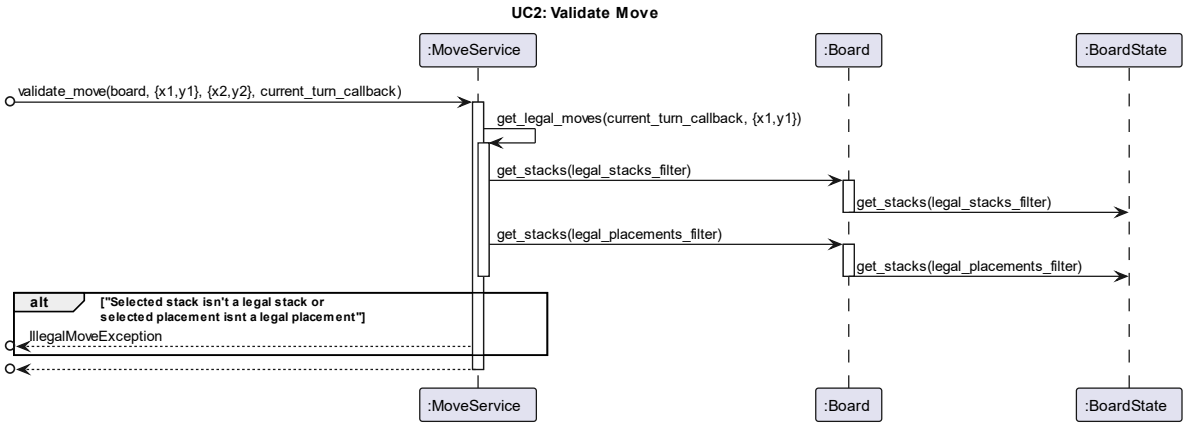



PlantUML Code

```
title UC1: Perform Player Turn

participant ":Orchestrator" as P1
participant ":Validator" as P2
participant ":TurnService" as P3
participant ":MoveService" as P4
participant ":Board" as P5
participant ":DisconnectService" as P6
participant ":Scorekeeper" as P7
participant "board_state:BoardState" as P8
participant "backup_board_state:BoardState" as P9

autoactivate on

[o-> P1 : perform_move({x,y},{x,y})

P1 -> P2 : validate({x,y},{x,y}, board)
P2 -> P3 : is_turn(board, {x,y})

P3 --> P2 : true

P2 -> P4 : validate_move(board, {x,y}, {x,y})
P4 --> P2 : true
```

```
P2 --> P1 : true

P1 -> P5 : perform_turn({x,y}, {x,y})
P5 -> P8 : copy()
deactivate P8
P5 -> P9 **
deactivate P5
P1 -> P2 : post_validate(board)

P2 -> P6 : disconnects(board)
deactivate P6

P2->P3 : increment_turn(has_legal_moves_cb)
deactivate P3
deactivate P2

P1 -> P7 : update_score(board, players)
deactivate P7

[o<--P1
```

# Use Case 2: Validate Move

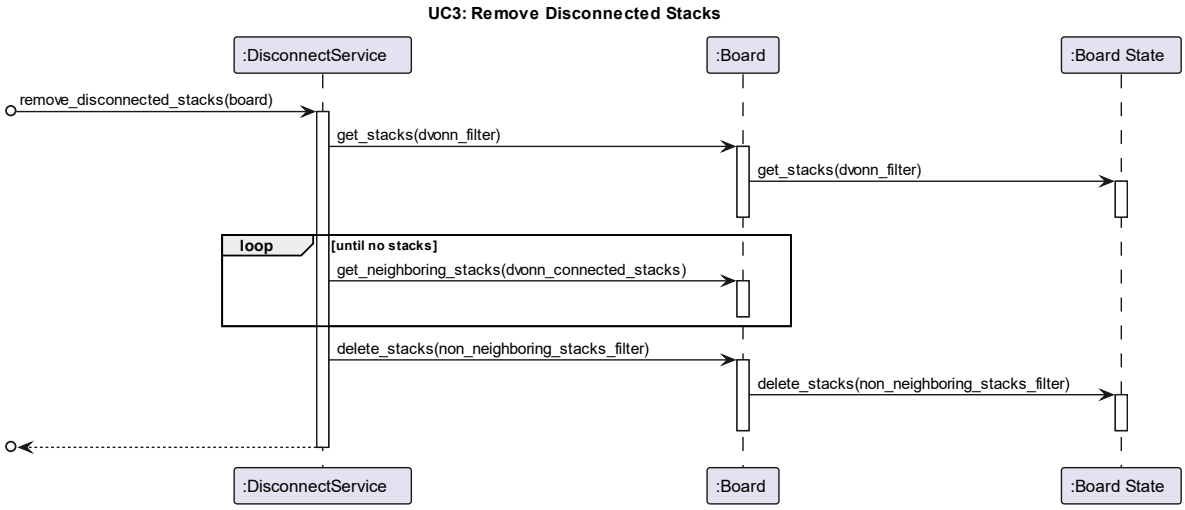



## PlantUML Code

```
title UC2: Validate Move

participant ":MoveService" as P4
participant ":Board" as P5
participant ":BoardState" as P6

[o-> P4 : validate_move(board, {x1,y1}, {x2,y2},
current_turn_callback)
activate P4

P4 -> P4 : get_legal_moves(current_turn_callback, {x1,y1})
activate P4

P4 -> P5 : get_stacks(legal_stacks_filter)
activate P5
P5 -> P6 : get_stacks(legal_stacks_filter)
deactivate P5

P4 -> P5 : get_stacks(legal_placements_filter)
activate P5
P5 -> P6 : get_stacks(legal_placements_filter)
deactivate P5

deactivate P4

alt "Selected stack isn't a legal stack or \nselected placement isnt a
legal placement"
```

```
[o<-- P4 : IllegalMoveException
end

[o<-- P4
deactivate P4
```

## Use Case 3: Remove Disconnected Stacks

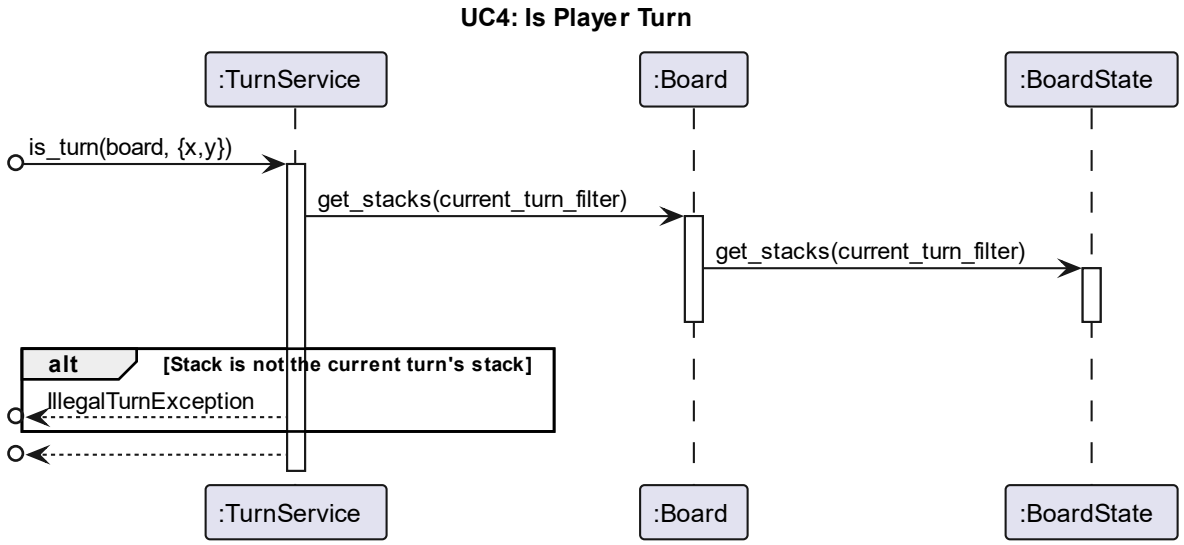



PlantUML Code

```
title UC3: Remove Disconnected Stacks


participant ":DisconnectService" as P6
participant ":Board" as P5
participant ":Board State" as P1


[o-> P6 : remove_disconnected_stacks(board)
activate P6

P6 -> P5 : get_stacks(dvonn_filter)
activate P5
P5 -> P1 : get_stacks(dvonn_filter)
activate P1
deactivate P1
deactivate P5
```

```
loop until no stacks
P6 -> P5 : get_neighboring_stacks(dvonn_connected_stacks)
activate P5
deactivate P5
end


P6 -> P5 : delete_stacks(non_neighboring_stacks_filter)
activate P5
P5 -> P1 : delete_stacks(non_neighboring_stacks_filter)
activate P1
deactivate P5
deactivate P1
[o<-- P6
deactivate P6
```

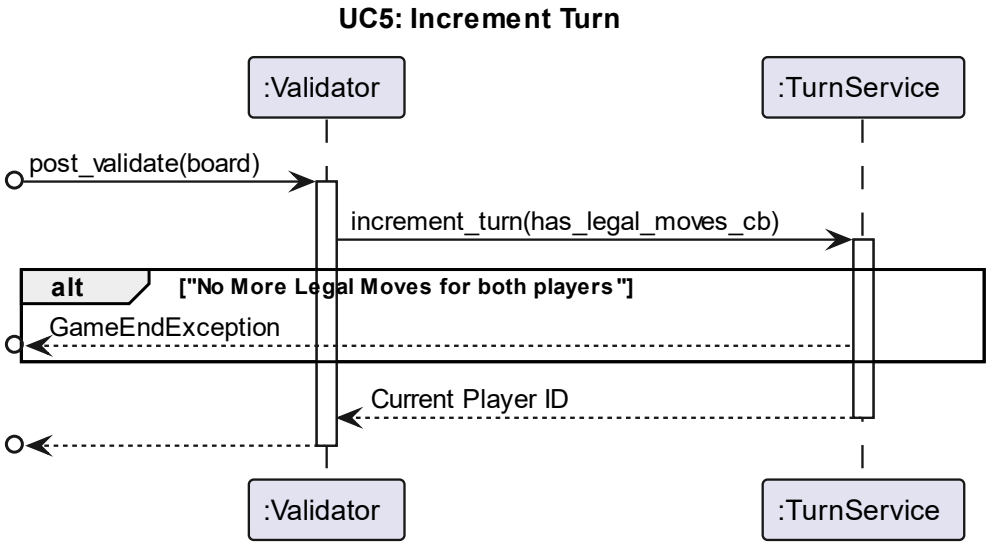
# Use Case 4: Is Player turn




PlantUML Code

```
title UC4: Is Player Turn

participant ":TurnService" as P1
participant ":Board" as P2
participant ":BoardState" as P3

[o-> P1 : is_turn(board, {x,y})

activate P1
P1 -> P2 : get_stacks(current_turn_filter)
activate P2

P2 -> P3 : get_stacks(current_turn_filter)
activate P3
deactivate P3
deactivate P2

alt Stack is not the current turn's stack
[o<--P1 : IllegalTurnException
end

[o<--P1
```

# Use Case 5 Sequence Diagram

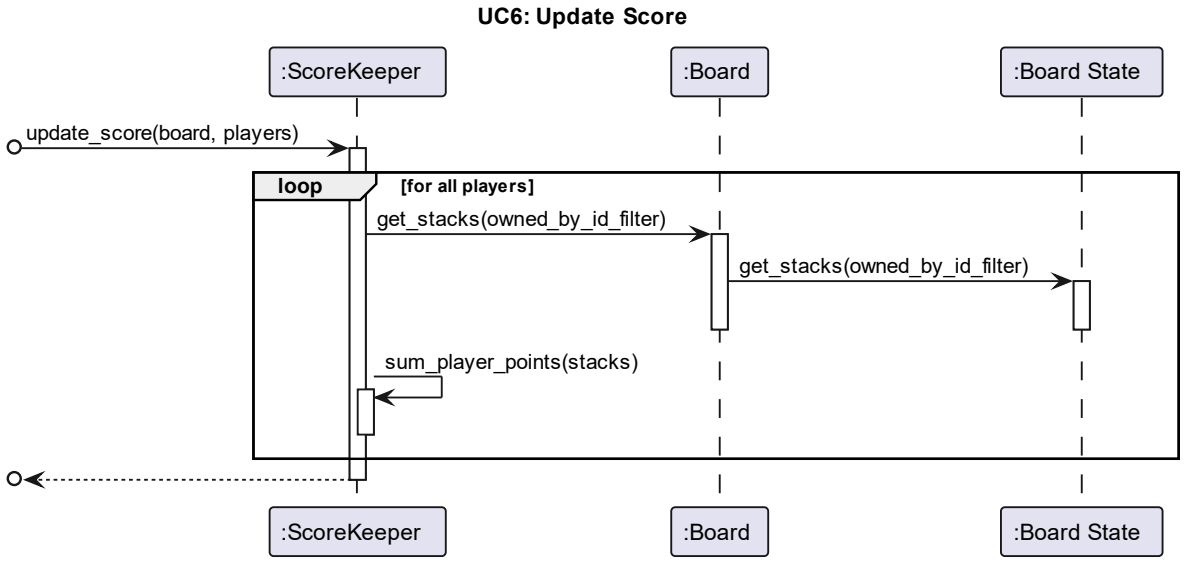



PlantUML Code

```
title UC5: Increment Turn

participant ":Validator" as P2
participant ":TurnService" as P3

[o->P2 : post_validate(board)
activate P2

P2-> P3: increment_turn(has_legal_moves_cb)
activate P3

alt "No More Legal Moves for both players"
[o<--P3 : GameEndException
end

P3 --> P2 : Current Player ID
deactivate P3
[o<-- P2
deactivate P2
```

# Use Case 6: Update Score

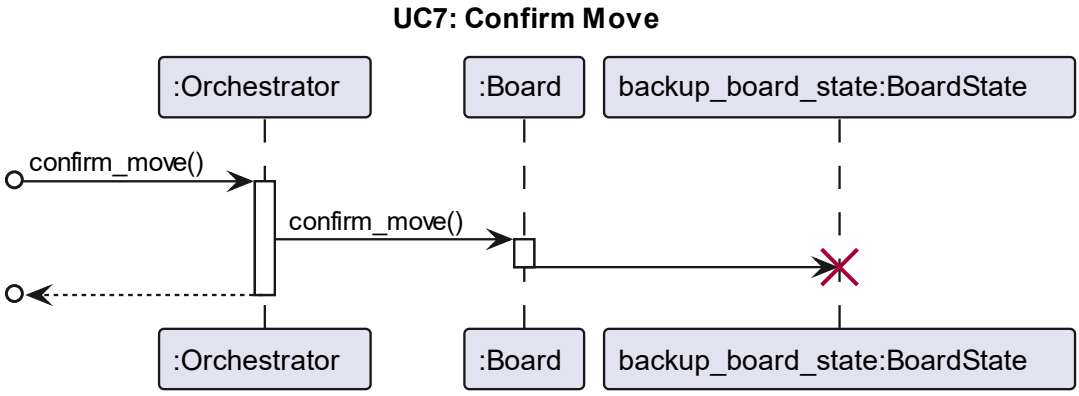
## PlantUML Code




```
title UC6: Update Score

participant ":ScoreKeeper" as P1
participant ":Board" as P2
participant ":Board State" as P3

[o->P1 : update_score(board, players)

activate P1
loop for all players
P1 -> P2 : get_stacks(owned_by_id_filter)

activate P2
P2 -> P3 : get_stacks(owned_by_id_filter)
activate P3
deactivate P3
deactivate P2

P1->P1 : sum_player_points(stacks)
activate P1
deactivate P1
end
```

```
[o<--P1
deactivate P1
```

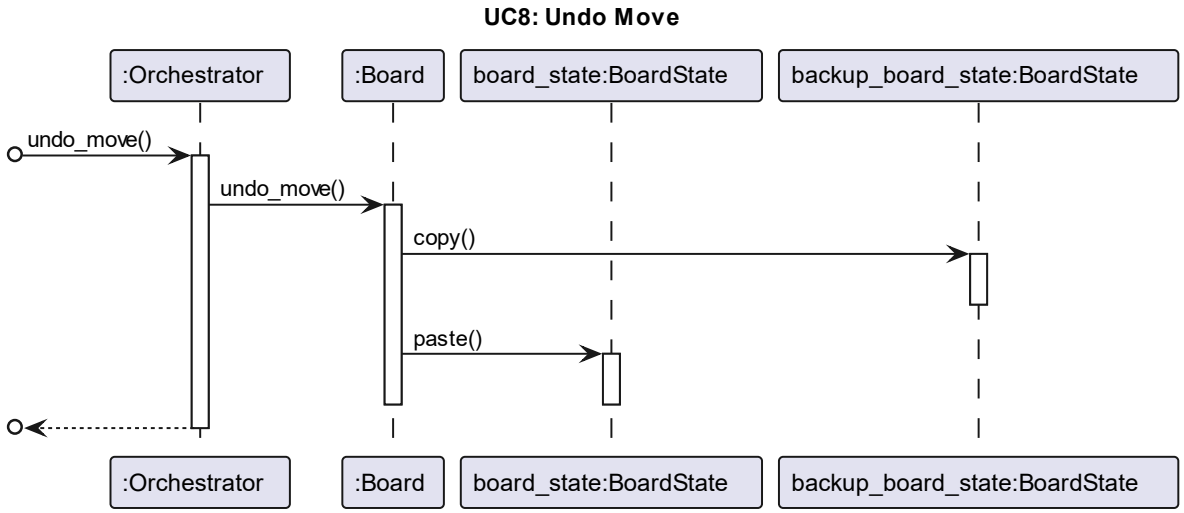
# Use Case 7: Confirm Move




PlantUML Code

```
title UC7: Undo Move

participant ":Orchestrator" as P1
participant ":Board" as P2
participant "backup_board_state:BoardState" as P3

[o-> P1 : undo_move()
activate P1
P1 -> P2 : undo_move()
activate P2
P2 -> P3
deactivate P2
destroy P3
[o<-- P1
deactivate P1
```

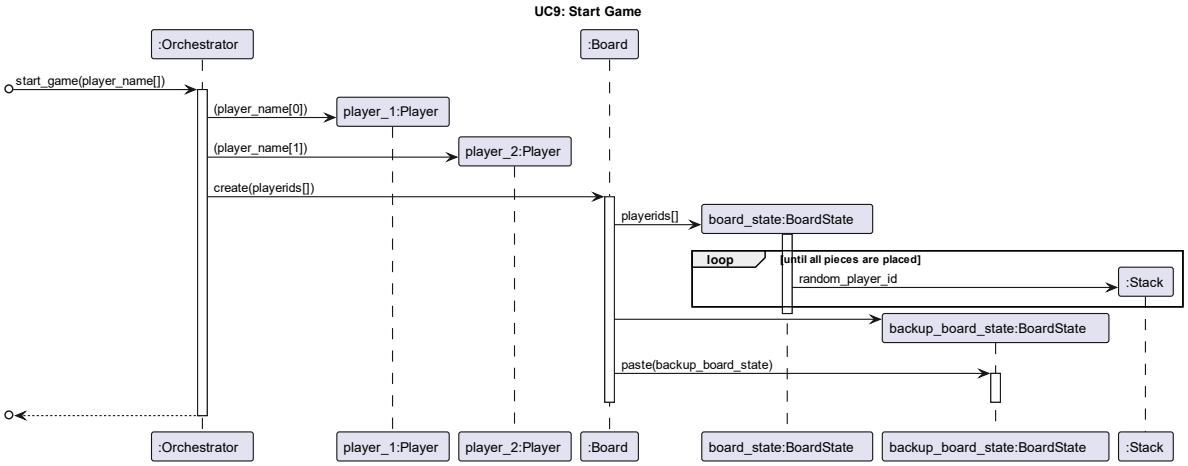
# Use Case 8: Undo Move




PlantUML Code

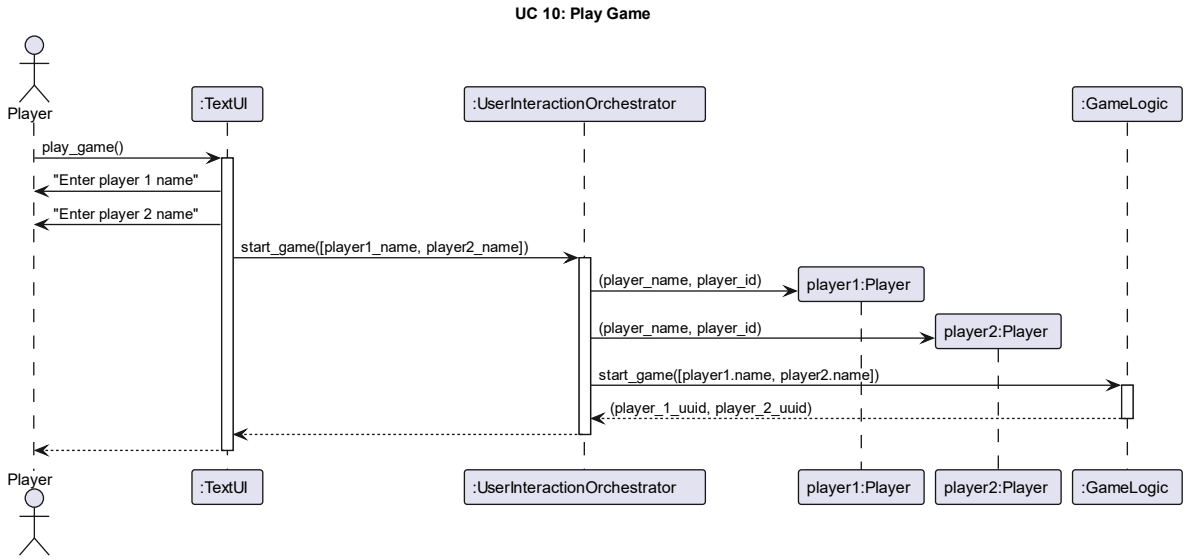
```
title UC8: Confirm Move

participant ":Orchestrator" as P1
participant ":Board" as P2
participant "board_state:BoardState" as P3
participant "backup_board_state:BoardState" as P4

[o-> P1 : confirm_move()
activate P1
P1 -> P2 : confirm_move()
activate P2
P2 -> P4: copy()
activate P4
deactivate P4
P2 -> P3 : paste()
activate P3
deactivate P3
deactivate P2
[o<-- P1
deactivate P1
```

# Use Case 9 Sequence Diagram




```
title UC9: Start Game

participant ":Orchestrator" as P1
participant "player_1:Player" as P2
participant "player_2:Player" as P3
participant ":Board" as P4
participant "board_state:BoardState" as P5
participant "backup_board_state:BoardState" as P6
participant ":Stack" as P7

[o-> P1: start_game(player_name[])
activate P1

P1 -> P2 ** : (player_name[0])
P1 -> P3 ** : (player_name[1])
P1 -> P4 : create(playerids[])
activate P4
P4 -> P5 ** : playerids[]
activate P5
loop until all pieces are placed
P5 -> P7 ** : random_player_id
end
deactivate P5
P4 -> P6 **
P4 -> P6 : paste(backup_board_state)
activate P6
deactivate P6
deactivate P4
[o<-- P1
deactivate P1
```

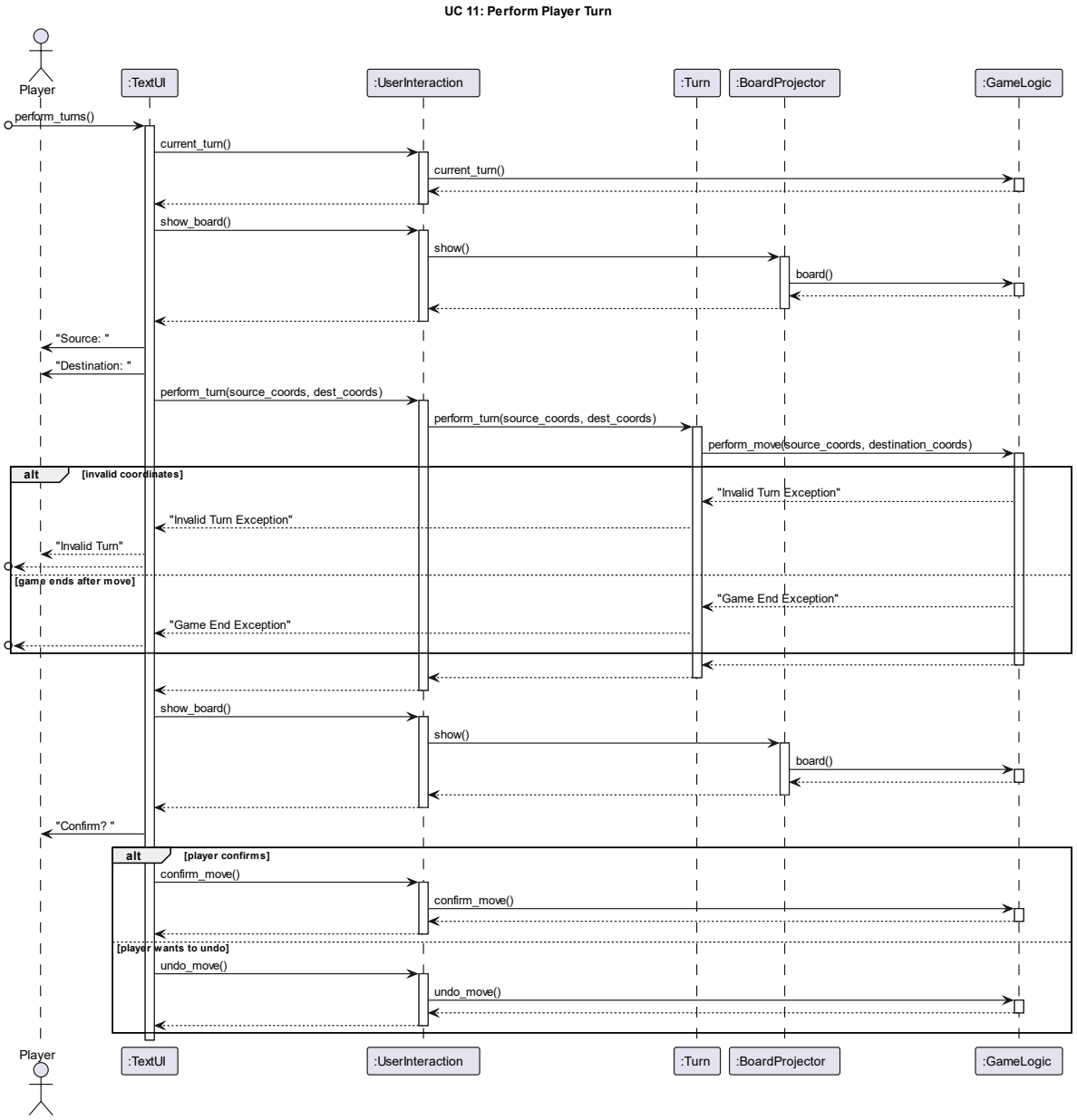
# Use Case 10 Sequence Diagram




```
title UC 10: Play Game

Actor Player
autoactivate on

Player -> ":TextUI" : play_game()

autoactivate off
":TextUI" -> "Player" : "Enter player 1 name"
autoactivate on


autoactivate off
":TextUI" -> "Player" : "Enter player 2 name"
autoactivate on

":TextUI" -> ":UserInteractionOrchestrator" : start_game([player1_name,
player2_name])
":UserInteractionOrchestrator" -> "player1:Player" ** : (player_name,
player_id)

":UserInteractionOrchestrator" -> "player2:Player" ** : (player_name,
player_id)

":UserInteractionOrchestrator" -> ":GameLogic" : start_game([player1.name,
player2.name])
":GameLogic" --> ":UserInteractionOrchestrator" : (player_1_uuid,
player_2_uuid)
```

```
deactivate ":GameLogic"
":UserInteractionOrchestrator" --> ":TextUI"
":TextUI" --> Player
```

# Use Case 11 Sequence Diagram

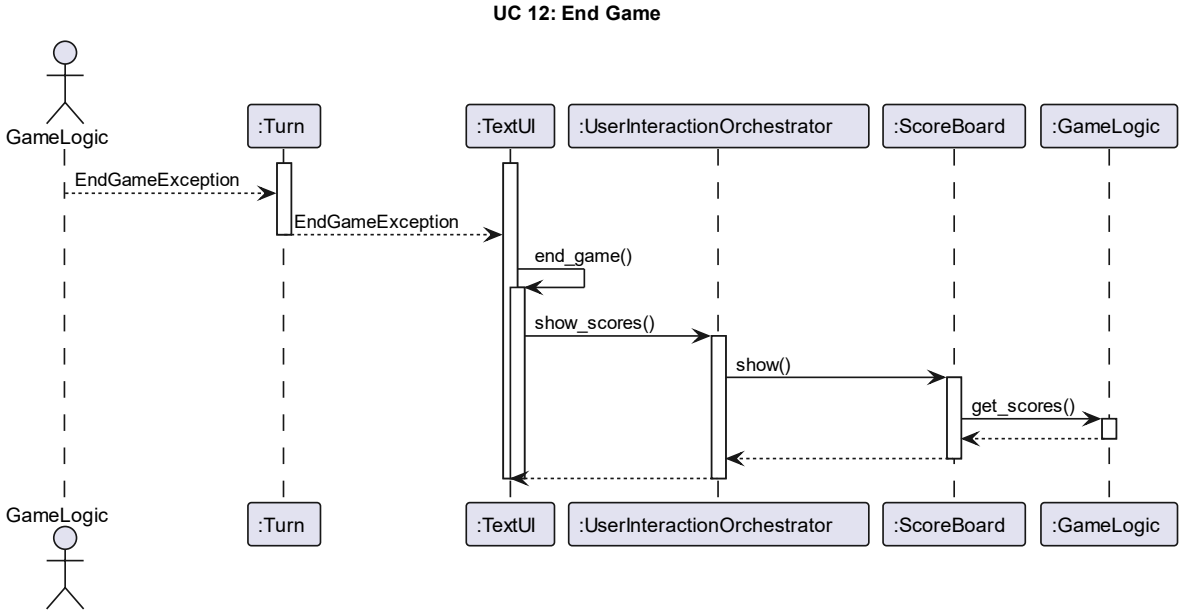



```
title UC 11: Perform Player Turn

Actor Player
autoactivate on

participant ":TextUI"
participant ":UserInteraction"
participant ":Turn"
participant ":BoardProjector"
participant ":GameLogic"
```

```
[o->":TextUI" : perform_turns()
":TextUI" -> ":UserInteraction" : current_turn()
":UserInteraction" -> ":GameLogic" : current_turn()
":GameLogic" --> ":UserInteraction"
":UserInteraction" --> ":TextUI"
deactivate ":GameLogic"
deactivate ":UserInteraction"

":TextUI" -> ":UserInteraction" : show_board()
":UserInteraction" -> ":BoardProjector" : show()
":BoardProjector" -> ":GameLogic" : board()
":GameLogic" --> ":BoardProjector"
":BoardProjector" --> ":UserInteraction"
":UserInteraction" --> ":TextUI"

autoactivate off
":TextUI" -> "Player" : "Source: "
":TextUI" -> "Player" : "Destination: "
autoactivate on

":TextUI" -> ":UserInteraction" : perform_turn(source_coords, dest_coords)
":UserInteraction" -> ":Turn" : perform_turn(source_coords, dest_coords)
":Turn" -> ":GameLogic" : perform_move(source_coords, destination_coords)

alt invalid coordinates
autoactivate off
":GameLogic"-->":Turn" : "Invalid Turn Exception"
":TextUI"<--":Turn" : "Invalid Turn Exception"
":TextUI" --> Player : "Invalid Turn"
[o<--":TextUI"

else game ends after move
":GameLogic"-->":Turn" : "Game End Exception"
":TextUI"<--":Turn" : "Game End Exception"
[o<--":TextUI"
end

autoactivate on
":GameLogic" --> ":Turn"
":Turn" --> ":UserInteraction"
":UserInteraction" --> ":TextUI"

":TextUI" -> ":UserInteraction" : show_board()
":UserInteraction" -> ":BoardProjector" : show()
```

```
":BoardProjector" -> ":GameLogic" : board()
":GameLogic" --> ":BoardProjector"
":BoardProjector" --> ":UserInteraction"
":UserInteraction" --> ":TextUI"

autoactivate off
    ":TextUI" -> "Player" : "Confirm? "
autoactivate on

alt player confirms
    ":TextUI" -> ":UserInteraction" : confirm_move()
    ":UserInteraction" -> ":GameLogic" : confirm_move()
    ":GameLogic" --> ":UserInteraction"
    ":UserInteraction" --> ":TextUI"
else player wants to undo
    ":TextUI" -> ":UserInteraction" : undo_move()
    ":UserInteraction" -> ":GameLogic" : undo_move()
    ":GameLogic" --> ":UserInteraction"
    ":UserInteraction" --> ":TextUI"
end
```

# Use Case 12 Sequence Diagram




```
title UC 12: End Game

autoactivate on
Actor GameLogic

activate ":Turn"
activate ":TextUI"
GameLogic --> ":Turn" : EndGameException

":Turn" --> ":TextUI": EndGameException
":TextUI" -> ":TextUI" : end_game()
":TextUI" -> ":UserInteractionOrchestrator" : show_scores()
":UserInteractionOrchestrator" -> ":ScoreBoard" : show()
":ScoreBoard" -> ":GameLogic" : get_scores()

":ScoreBoard" <-- ":GameLogic"
":UserInteractionOrchestrator" <-- ":ScoreBoard"
":TextUI" <-- ":UserInteractionOrchestrator"
deactivate ":TextUI"
deactivate ":TextUI"
```

# Class Descriptions

## Model Classes:

### Orchestrator

- Constructor:
  - **Orchestrator(List<Strings>? player_names)**
- Instance Variables:
  - validator: Validator
  - board: Board
  - Players: List<Player>
  - Move_service: MoveService
  - turn_service: TurnService
  - score_keeper: ScoreKeeper
- Methods
  - **performMove(Tuple src, Tuple dest):** Performs players move
  - **findLegalMoves(player_id):** Finds legal moves for player
  - **startGame():** Initializes services and creates the Board
  - **get_scores():** Gives the player list
  - **confirm_move():** Confirm move on the *Board*
  - **undo_move():** Undo move on the *Board*

### Validator

- Constructor:
  - **Validator(List<UUID> player_ids)**
- Instance Variables:
  - turn_service: TurnService
  - disconnect_service: DisconnectService
  - move_service: MoveService
- Methods
  - **validate(Tuple<Int>, Tuple<Int>, Board board):** Checks if move is valid
  - **postValidate(Board board):** Sets up next turn

### Board

- Constructor:

    - **Board()**
- Instance Variables:
    - board_state: BoardState
    - backup_board: BoardState
- Methods
    - **create():** Creates new board and duplicated backup
    - **update(Tuple original_coords, Tuple new_coords):** Moves stack from *original_coords* to *new_coords* on the main *board_state.*
    - **getStacks(Function filter):** Gets a set of stacks from BoardState as described in filter
    - **deleteStacks(Function filter):** Deletes a set of stacks from BoardState as described in filter
    - **confirm_move():** Delete old *backup_board* and duplicate the main *board_state* into the backup.
    - **undo_move():** Restore old *backup_board* by duplicating it into *board_state.*

## BoardState

- Constructor:
    **BoardState(List<UUID> player_ids)**
- Instance Variables:
    - stack_arrangement: Array<Array<Stack>>
- Methods
    - **getStacks(Function filter):** Gets a set of stacks as described in filter
    - **update(HashSet<Tuple, Stack> original_coords, HashSet<Tuple, Stack> new_coords)**: Moves one stack from one coordinate to another
    - **deleteStacks(Function filter):** Deletes a set of stacks as described in filter
    - **paste(BoardState board_state):** It copies the provided *board_state's* instance variables into itself.

## Stack

- Constructor:
    - **Stack(Int num_pieces, Bool is_DVONN, UUID? owner)**
- Instance Variables:
    - id: UUID
    - owner: UUID?

    - num_pieces: Integer
    - is_DVONN: Bool
- Methods
    - **num_pieces():** Gets the number of pieces in the Stack.
    - **is_DVONN():** Check whether the stack contains a DVONN piece.
    - **owned_by(UUID player_id):** Checks whether the stack is owned by the player specified ID.

## Player

- Constructor:
    - **Player(String name)**
- Instance Variables:
    - name: String
    - points: Integer
    - id: UUID
- Methods
    - **name():** Gets the name
    - **points()**: Gets the points
    - **id():** Gets the id

## DisconnectService

- Constructor:
    - **DisconnectService()**
- Methods
    - **remove_disconnected_stacks(Board board):** Removes Stacks disconnected from all DVONN stacks.

## TurnService

- Constructor:
    - **TurnService(List<UUID> player_ids)**
- Instance Variables:
    - currentTurn: UUID
    - player_ids: List<UUID>
- Methods

- **is_turn(Board board, Tuple<Int> coords):** Checks if stack belongs to the player whose turn it is
- **currentTurn():** Returns current turn's player ID.
- **increment_turn(Function has_legal_moves_cb):** Changes turn to next player if they have legal moves

## MoveService

- Constructor:
  - **MoveService()**
- Methods
  - **validate_move(Board board, Tuple src, Tuple dest, Function curr_turn):** Checks if move is legal
  - **retrieveLegalMoves(Board board, UUID player_id):** Retrieves legal moves for a player

## ScoreKeeper

- Constructor:
  - **ScoreKeeper()**
- Methods
  - **update_score(List<Players> players, Board board):** Updates scores of all players

# View Classes

## TextUI

- Contructor:
  - **TextUI()**
- Instance Variables:
  - orchestrator: UserInteractionOrchestrator
- Methods:
  - **play_game()**
    - The only public method
    - Called by the player at the start of the game
    - Calls start_game()
    - Calls perform_turns() in a loop, until a GameEndException is raised.
    - Once the exception is raised, calls end_game()
  - **perform_turn()**
    - Gets the current turn from the UserInteractionOrchestrator
    - Displays the board using the UserInteractionOrchestrator
    - Asks the user for source and destination coordinates
    - Calls perform_turn() on the UserInteractionOrchestrator with the given coordinates
    - Displays the board again
    - Asks for confirmation from the user
    - Calls confirm_move or undo_move depending on the response

## UserInteractionOrchestrator

- Contructor:
  - **UserInteractionOrchestrator()**
- Instance Variables:
  - game_logic: Orchestrator
  - players: HashSet<UUID, Player>
  - board_projector: BoardProjector
  - turn_service: Turn
- Methods:
  - **start_game(List<String> names)**
    - Initializes the players given the names and calls start_game on the model
  - **perform_turn(Tuple src, Tuple dest): Bool**
    - Delegates to the Turn class
  - **confirm_move()**
    - Delegates to the Turn class
  - **undo_move()**
    - Delegates to the Turn class
  - **show_board()**
    - Delegates to the BoardProjector class
  - **show_scores()**
    - Delegates to the ScoreBoard class
  - **current_turn()**
    - Calls the current_turn() method on the model

## Turn

- Constructor:
  - **Turn(Orchestrator game_logic)**
- Instance Variables:
  - game_logic: Orchestrator
- Methods
  - **perform_turn(Tuple src, Tuple dest): Bool**
    - An abstraction of the **perform_move()** method on the Orchestrator. Bubbles up exceptions to the caller.
  - **confirm_move()**
    - An abstraction of the **confirm_move()** method on the Orchestrator.
  - **undo_move()**

        - An abstraction of the **undo_move()** method on the Orchestrator.

## BoardProjector

- Constructor:
    - **BoardProjector(Orchestrator game_logic)**
- Instance Variables:
    - game_logic: Orchestrator
- Methods:
    - **show(HashSet<UUID, Player> players)**
        - Fetches the board from the model.
        - Maps each user id on the model's board to a symbol from the players.
        - Displays the baord as outlined in the text based UI section.

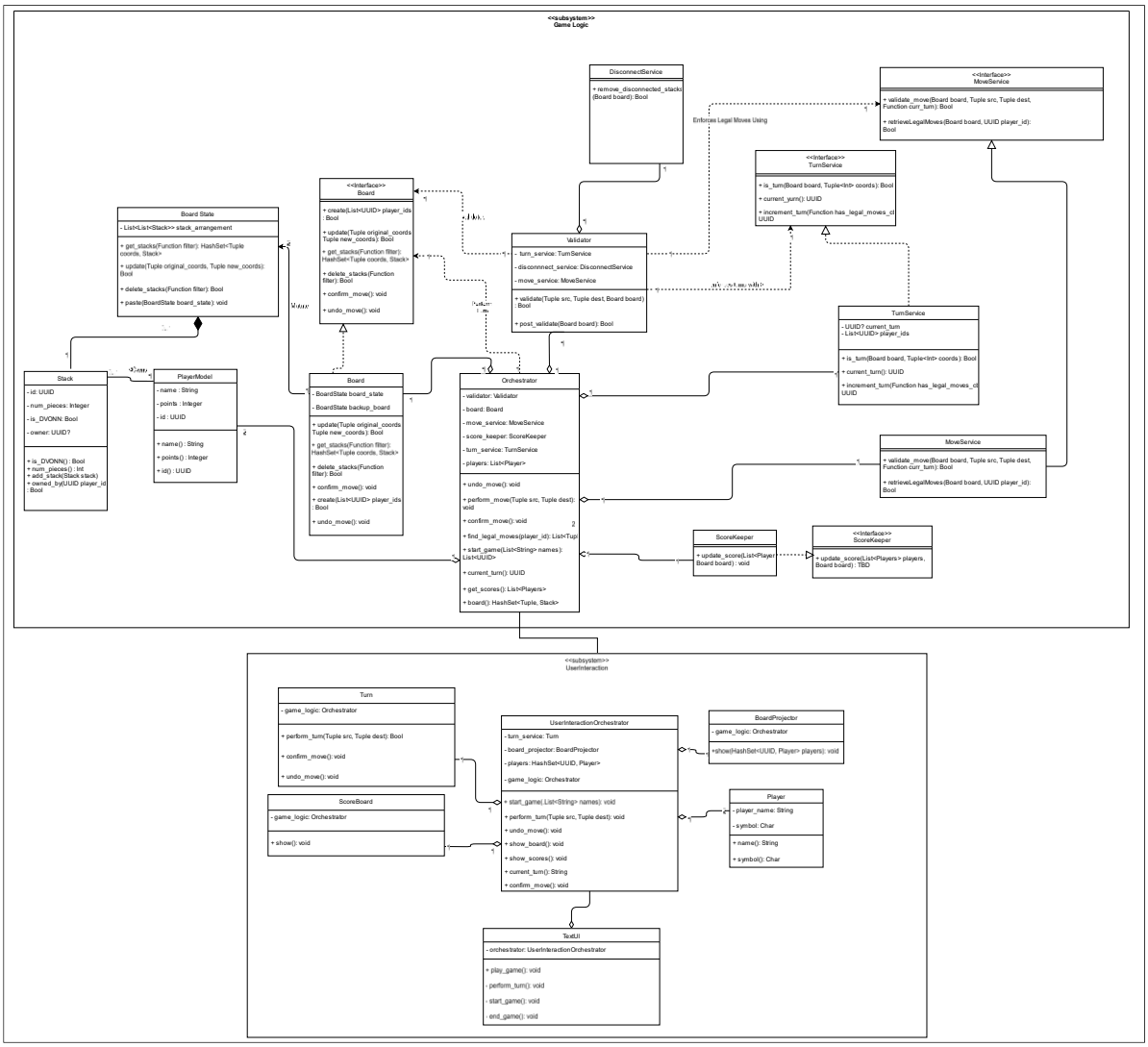
## ScoreBoard

- Constructor:
    - **ScoreBoard (Orchestrator game_logic)**
- Instance Variables:
    - game_logic: Orchestrator
- Methods:
    - **show()**
        - Fetches the scores from the model.
        - Displays the result as outlined in the text based UI section.

## Player

- Constructor:
    - **Player(String name, Char symbol)**
- Instance Variables:
    - player_name: String
    - symbol: Char
- Methods:
    - **name()**
        - returns the name of the player
    - **symbol()**
        - returns the symbol of the player (symbols are according to the UI

# Class Diagram



# Text Based UI:

```
// For demonstrative purposes only:

*********************************************************************************
Welcome to DVONN!

Player 1 name: Great-Player1
Player 2 name: Legendary-DVONN-Master


Pending Great-Player1's turn (x):
Board:
1         x1      x1      o1      o1      x1      o1      d1      o1      x1
2     o1      o1      x1      o1      x1      x1      o1      x1      o1      x1
3 x1      x1      o1      o1      x1      o1      d1      o1      x1      o1      x1
4     o1      o1      d1      o1      x1      x1      o1      x1      o1      x1
5         x1      x1      o1      o1      x1      o1      o1      o1      x1
  1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21

Source: 5,3
Destination: 4,2

Result of move:
1         x1      x1      o1      o1      x1      o1      d1      o1      x1
2     o1      o1      x1      o1      x1      x1      o1      x1      o1      x1
3 x1      x1      o1      o1      x1      o1      d1      o1      x1      o1      x1
4     x2      o1      d1      o1      x1      x1      o1      x1      o1      x1
5                 x1      o1      o1      x1      o1      o1      o1      x1
  1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21

Confirm? Y

---------------------------------------------------------------------------------
Pending Legendary-DVONN-Master's turn(o):

Board:
1         x1      x1      o1      o1      x1      o1      d1      o1      x1
2     o1      o1      o2      o1      x1      x1      o1      x1      o1      x1
3 x1      x1      o1              x1      o1      d1      o1      x1      o1      x1
4     x2      o1      d1      o1      x1      x1      o1      x1      o1      x1
5                 x1      o1      o1      x1      o1      o1      o1      x1
  1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21

Source: 3,5
```

```
Destination: 4,6

You selected an illegal stack. Please try again with a stack you own.

Board:
1         x1      x1      o1      o1      x1      o1      d1      o1      x1
2     o1      o1      o2      o1      x1      x1      o1      x1      o1      x1
3 x1      x1      o1              x1      o1      d1      o1      x1      o1      x1
4     x2      o1      d1      o1      x1      x1      o1      x1      o1      x1
5                 x1      o1      o1      x1      o1      o1      o1      x1
  1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21

Source: 5,5
Destination: 1,3

You selected an illegal placement for your stack. Please try again with a placement that
is number of pieces away from your stack.

Board:
1         x1      x1      o1      o1      x1      o1      d1      o1      x1
2     o1      o1      o2      o1      x1      x1      o1      x1      o1      x1
3 x1      x1      o1              x1      o1      d1      o1      x1      o1      x1
4     x2      o1      d1      o1      x1      x1      o1      x1      o1      x1
5                 x1      o1      o1      x1      o1      o1      o1      x1
  1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21

.
.
.

Source: 5,3
Destination: 4,2

Game Ended...

*********************************************************************************
Great-Player1's score: 25
Legendary-DVONN-Master's score: 45
*********************************************************************************
The winner is.....

Legendary-DVONN-Master!!!!!

Thank you, bye bye
*********************************************************************************
```

# Archive

These are the Initial Object Model and Use Cases. They have not been updated since A1 and are thus outdated.

## Initial Object Model

## Entity List

### Space:

A collection of Slots.

### Slot:

A location for a Stack to be placed on.

### Row:

An adjacent set of Slots on the Board

### Board State:

The position of each Stack.

### Stack:

Holds the number of Pieces inside it and its ownership.

### Piece:

Part of a Stack, and counts for 1 Point.

### DVONN Piece:

A piece that initially does not belong to any Player.

### DVONN Stack:

A Stack containing a DVONN Piece. All Stacks must be connected to a DVONN Stack.

### Legal Stack:

A Stack that can legally move.

### Legal Placement:

A Stack that another Stack can be legally placed on.

### Legal Move:

A combination of a Legal Stack and a Legal Placement.

### Game Master:

Directs the flow of the game (start, player turns and end).

### Invigilator:

Maintains the validity of a given board state by checking for disconnections or finding Legal Moves for a given player.

### Board:

Manages the Board State by handling movements and disconnects.

### Player:

A player interacting with the game, has a score based on the number of pieces in each Stack they own.

### Opponent:

A Player who’s not currently performing a turn.

## Action List

### Game Master:

Performs Player Turn

### Invigilator:

Finds legal placements for player, finds Legal Stacks for Player, and validates board state

### Piece Engine:

Performs moves.

### Game master:

Creates the Board State, starts and ends game, count Player Points

# Use Cases

## UC1: Perform Player Turn

**Description:** Handles the operations involved in a player's turn.
**Primary Actor:** Game Master
**Initiating Event:**

- Opponent completes their turn

**Basic Flow:**

1. Opponent confirms Legal Move
2. Game Master prompts Invigilator to find Legal Stacks of the Player whose turn it is. (UC8)
3. Game Master prompts Invigilator to find Legal Moves of the Player whose turn it is. (UC2)
4. Player selects a Legal Stack.
5. Player selects a Legal Placement.
6. Piece Engine performs the Legal Move. (UC4)
7. Player confirms the Legal Move.

**Alternate Flows:**

1a. Invigilator finds no Legal Stacks.
    1a.2. Game Master begins the other Player's turn.
3a. Invigilator finds no Legal Moves.
    2a.1. Game Master checks the end of the game. (UC5)
6a. Player doesn't confirm
    6a.1. Piece engine resets the Board State to pre-move state.

## UC2: Find Legal Moves

**Description:** Finds the Legal Stacks who have at least one Legal Placement.
**Primary Actor**: Invigilator
**Basic Flow**:

1. Game Master prompts Invigilator for Legal Moves.
2. Invigilator finds the number of Pieces in a Legal Stack.
3. Invigilator finds another stack that is the same number of spaces away as the number of pieces in the Legal Stack
4. Invigilator creates a Legal Placement from the Stack.
5. Invigilator creates a Legal Move from the Legal Stack and Legal Placement.
6. Invigilator repeats steps 2-4 once for each direction.
7. Invigilator repeats steps 1-5 once for each Legal Stack.

**Alternate Flows**:

2a.There is no Stack that is the same number of spaces away as the number of pieces in the Legal Stack in the given direction.

2a.1. Invigilator repeats step 2 with the next direction.

## UC3: Remove Disconnected Stacks

**Description:** Ensure a valid game state by checking for any Stacks not connected to a DVONN Stack.
**Primary Actor**: Piece Engine
**Basic Flow**:

1. Game Master prompts Piece Engine to Remove Disconnected Stacks.
2. Piece Engine prompts Invigilator to find connected stacks
3. Invigilator finds all DVONN Stacks.
4. Invigilator finds all Stacks directly connected to a given DVONN Stack.
5. Invigilator finds all Stacks connected directly to a directly connected Stack.
6. Invigilator repeats step 3 until no more Stacks can be found.
7. Invigilator repeats steps 2-3 once for each DVONN Stack.
8. Piece engine removes Stacks not found in steps 2-5.
9. Piece engine updates Board State.

**Alternate Flows**:

3a. Invigilator already found Stack while checking another DVONN Stack.

2a.1. Invigilator skips current Stack, as it knows that Stack is connected.

4a. Invigilator already found Stack while checking another DVONN Stack.

3a.1. Invigilator skips current Stack, as it knows that Stack is connected.

8a. All Stacks are connected.

6a.1. Piece Engine exits flow early.

## UC4: Perform Legal Move

**Description:** Handles the movement of a Legal Stack onto a Legal Placement.
**Primary Actor:** Piece engine
**Basic Flow:**

1. Game Master prompts Piece Engine to perform Legal Move.
2. Piece Engine places the Legal Stack's Pieces on top of the Legal Placement's Pieces.
3. Piece Engine removes the Legal Stack.
4. Piece Engine updates the Board State.
5. Piece Engine Removes Disconnected Pieces. (UC3)

**Alternate Flows:**
2a. The Legal Stack and Legal Placement have different ownership.
    2a.1. Piece Engine updates the ownership of the Legal Placement.

## UC5: End Game

**Description:** Declare a winner if the game has ended.

**Primary Actor**: Game Master

**Basic Flow**:

1. Player confirms Legal Move.
2. Game Master prompts Invigilator to check end game conditions.
3. Invigilator finds Legal Stacks of the Opponent. (UC8)
4. Invigilator finds Legal Moves of the Opponent. (UC2)
5. Invigilator counts Points for the Player. (UC6)
6. Invigilator counts Points for the Opponent. (UC6)
7. Invigilator informs Game Master each Player's Points.
8. Game Master declares Player with the most Points as the winner.
9. Game Master prompts Players to start a new game

**Alternate Flows**:

4a. Invigilator finds Legal Move for Opponent:
    4a.1. Game Master prompts Opponent for the next move. (UC1)
8a. Players have equal Points:
    8a. 1. Game Master prompts both Players there is a draw.

## UC6: Count Player Points

**Primary Actor**: Invigilator

**Description**: Sums the player's Points based on their Stacks.

**Basic Flow:**

1. Game Master prompts Invigilator to count Player Points.
2. Invigilator finds all Player Stacks.
3. Invigilator starts to count Points from zero.
4. Invigilator measures the height for each Stack.
5. Invigilator adds stack height for each Stack to Points.
6. Invigilator assigns Points to players

**Alternate Flow:**
N/A

## UC7: Create the Board

**Description:** Organizes the pieces on the board to enable the start of the game.

**Primary Actor**: Piece Engine

**Basic Flow**:
1. Player prompts Piece Engine for new Board State
2. Piece Engine creates a Space enough for 11x5 Pieces.
3. Piece Engine places 9 Pieces offset from the left by two Slots on the first Row.
4. Piece Engine places 10 Pieces offset from the left by one Slot on the second Row.
5. Piece Engine places 11 Pieces with no offset on the third Row.
6. Piece Engine places 10 Pieces offset from the left by one slot on the fourth Row.
7. Piece Engine places 9 Pieces offset from the left by two slots on the fifth Row.

**Alternate Flow:**
N/A

## UC8: Find Legal Stacks For Player

**Description:** Finds all stacks that can be legally moved for a given player, meaning they are not completely surrounded.
**Primary Actor:** Invigilator
**Basic Flow:**

1. Game Master prompts Invigilator to find Legal Stacks for Player
2. Invigilator collects all Stacks for a given Player.
3. Invigilator counts the other Stacks around a given player Stack.
4. Invigilator creates a Legal Stack from the given player Stack.
5. Invigilator repeats steps 2-3 once for each player Stack.

**Alternate Flow:**
3a. The given player Stack is completely surrounded.
    3a.2. Invigilator returns to step 2 with the next Stack.