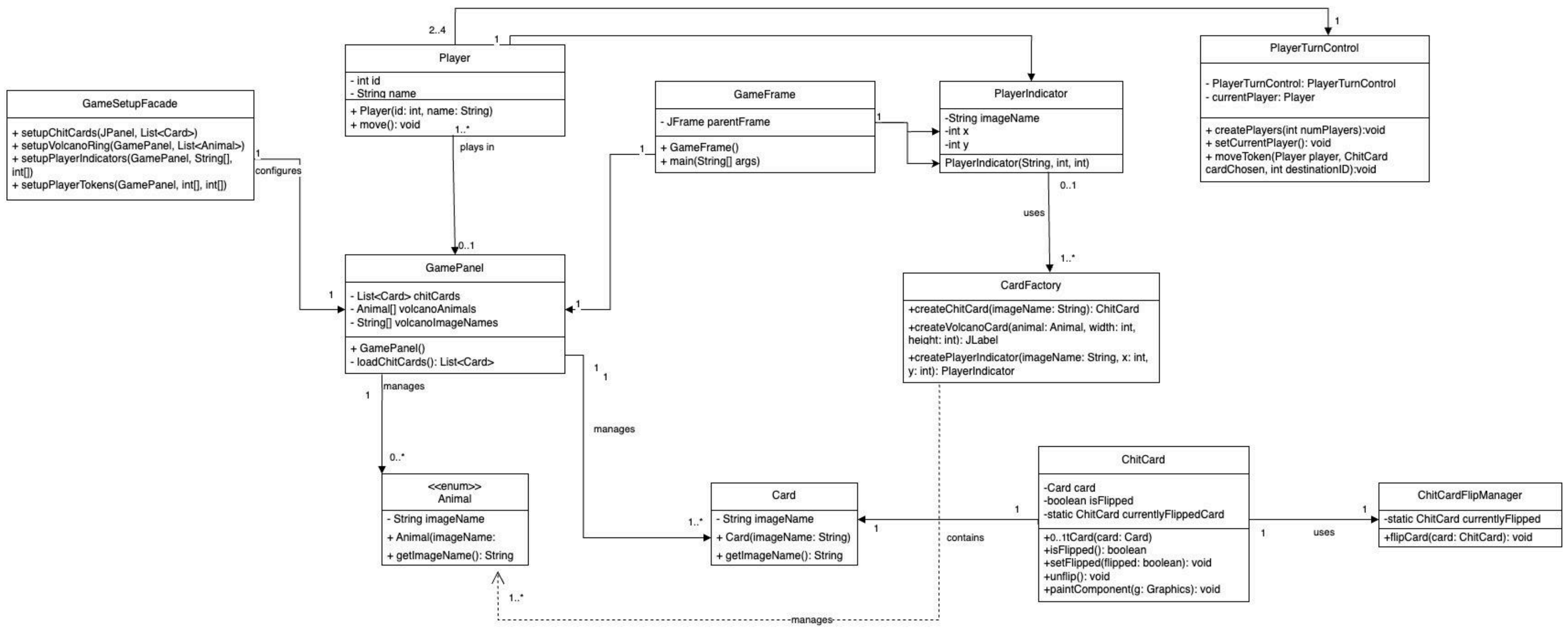
# SPRINT 2 - UML
# (CLASS & SEQUENCE)

**1.1 Class Diagram**



Player
- int id
- String name
+ Player(id: int, name: String)
+ move(): void
2..4
1
1..*
plays in
0..1
PlayerTurnControl
- PlayerTurnControl: PlayerTurnControl
- currentPlayer: Player
+ createPlayers(int numPlayers):void
+ setCurrentPlayer(): void
+ moveToken(Player player, ChitCard cardChosen, int destinationID):void
GameSetupFacade
+ setupChitCards(JPanel, List<Card>)
+ setupVolcanoRing(GamePanel, List<Animal>)
+ setupPlayerIndicators(GamePanel, String[], int[])
+ setupPlayerTokens(GamePanel, int[], int[])
configures
GameFrame
- JFrame parentFrame
+ GameFrame()
+ main(String[] args)
PlayerIndicator
-String imageName
-int x
-int y
PlayerIndicator(String, int, int)
uses
0..1
1..*
GamePanel
- List<Card> chitCards
- Animal[] volcanoAnimals
- String[] volcanoImageNames
+ GamePanel()
- loadChitCards(): List<Card>
CardFactory
+createChitCard(imageName: String): ChitCard
+createVolcanoCard(animal: Animal, width: int, height: int): JLabel
+createPlayerIndicator(imageName: String, x: int, y: int): PlayerIndicator
manages
0..*
<<enum>>
Animal
- String imageName
+ Animal(imageName:
+ getImageName(): String
1..*
manages
1..*
Card
- String imageName
+ Card(imageName: String)
+ getImageName(): String
contains
ChitCard
-Card card
-boolean isFlipped
-static ChitCard currentlyFlippedCard
+0..1tCard(card: Card)
+isFlipped(): boolean
+setFlipped(flipped: boolean): void
+unflip(): void
+paintComponent(g: Graphics): void
uses
ChitCardFlipManager
-static ChitCard currentlyFlipped
+flipCard(card: ChitCard): void

## 1.2 Sequence Diagram

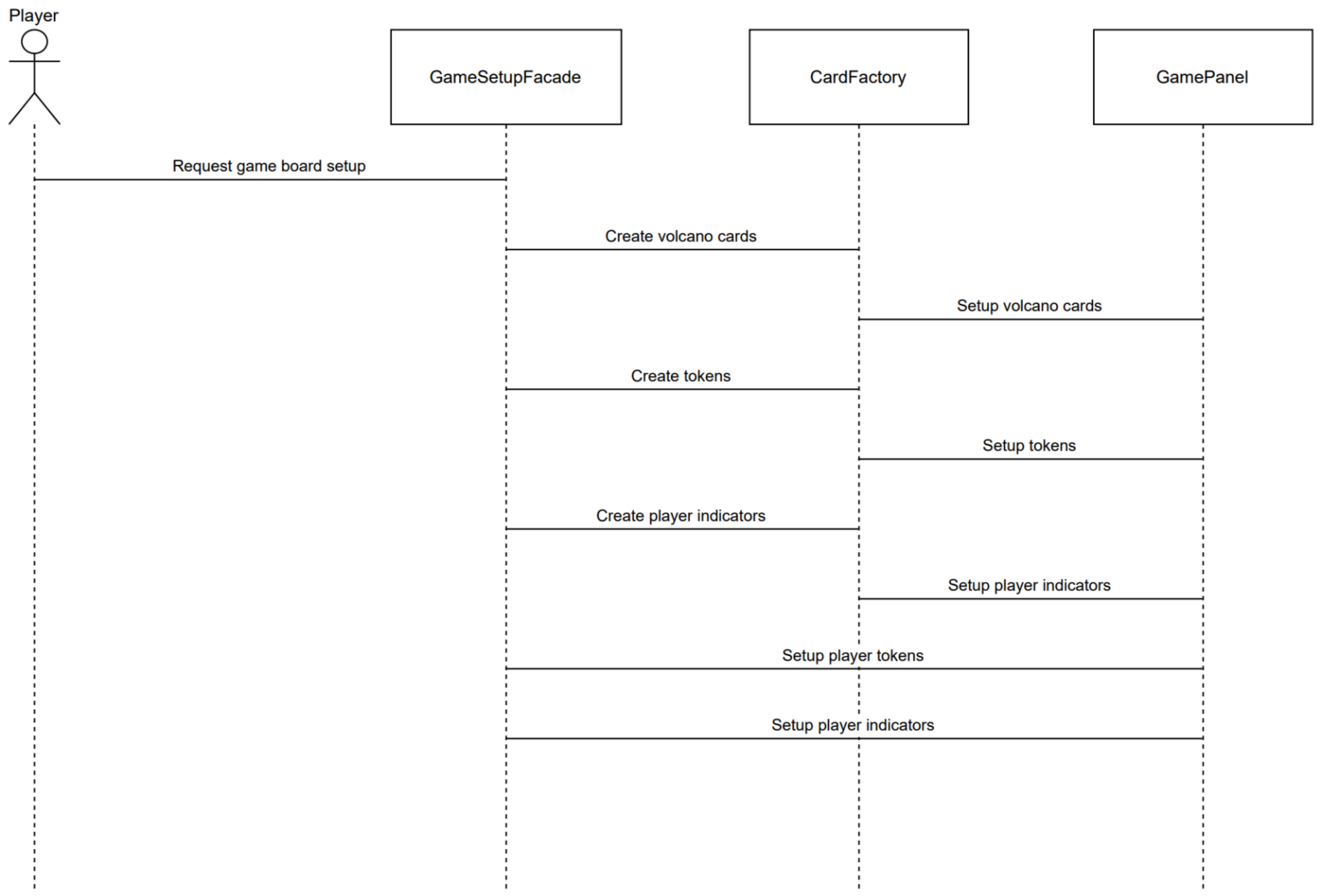
### 1.2.1 Setup of Game Board with Volcano Cards, Tokens and Player Indicator

Firstly, the game board is initialised within the GamePanel class, which sets the preferred dimensions and layout for the game interface. The GameSetupFacade class plays a main role in the setup process, particularly for the volcano cards, tokens, and player indicators.

For the volcano cards, the CardFactory class is utilized by the GameSetupFacade to create volcano cards. These cards are generated using specific animal types, widths, and heights, on the game board. The CardFactory also handles the loading and scaling of card images from the resources directory, contributing to the visual richness of the game.

In parallel, player tokens are set up on the game board using the setupPlayerTokens method from the GameSetupFacade. These tokens are represented as JLabels with distinct player numbers, strategically positioned based on game logic and design. This setup ensures that each player has a clear starting position and visual representation during gameplay.

Additionally, player indicators are established to mark the starting positions of players. These indicators, created through the *setupPlayerIndicators* method, utilize specific image names to differentiate players. The positions of these indicators are calculated to align with player tokens, providing a cohesive and intuitive visual layout on the game board.


Player
GameSetupFacade
CardFactory
GamePanel
Request game board setup
Create volcano cards
Setup volcano cards
Create tokens
Setup tokens
Create player indicators
Setup player indicators
Setup player tokens
Setup player indicators

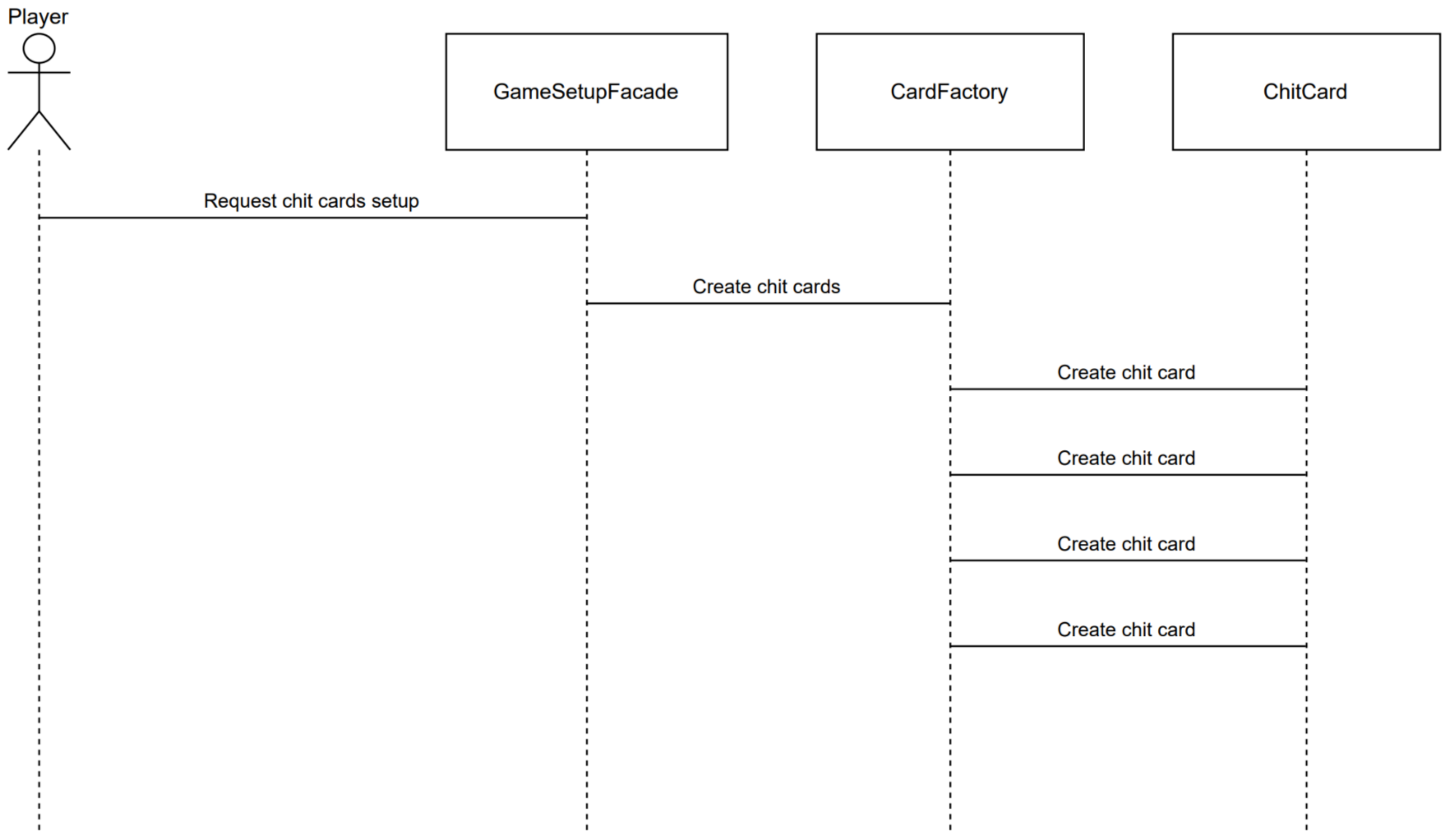
### 1.2.2 Setting up of the ChitCards

The setup of Chitcards in the Fiery Dragons game involves the creation and placement of interactive game cards known as Chitcards. This process is arranged by the GamePanel class and the GameSetupFacade.

To begin with, the GamePanel class initializes a list of Chitcards, which are loaded with images corresponding to various game elements such as dragons, bats, salamanders, and spiders. These Chitcards are crucial components of the game's mechanics and provide interactive elements for players during gameplay.

The GameSetupFacade class plays a big role in setting up ChitCards on the game board. Using the setupChitCards method, the facade arranges the ChitCards in a grid layout within a designated panel, ensuring a visually organized and accessible arrangement for players. This setup process includes shuffling the Chitcards to introduce an element of randomness and challenge to the gameplay experience.

MONASH University

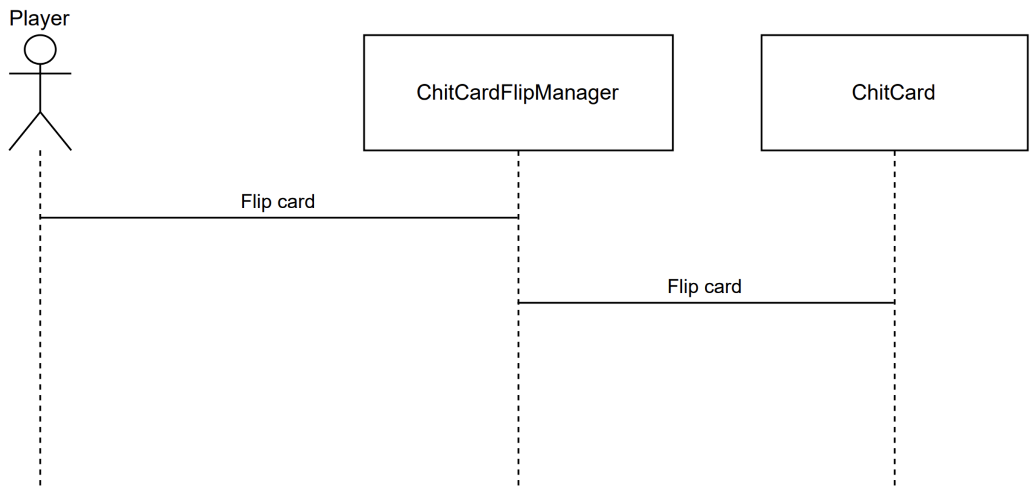


## 1.2.3 Flipping of ChitCard Functionality

This functionality is managed by the ChitCard class and the ChitCardFlipManager. When a player interacts with a Chitcard by clicking on it, the ChitCard class handles the flipping action through its *setFlipped* and *unflip* methods. These methods toggle the state of the Chitcard between face-up and face-down, revealing or concealing the associated image based on player actions.

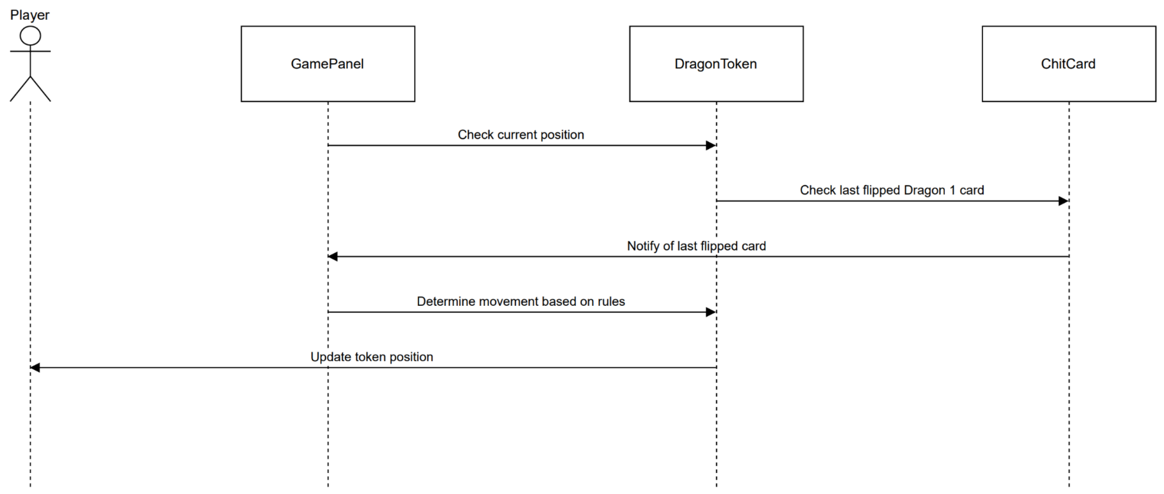
The ChitCardFlipManager class oversees the flipping process to ensure that only one Chitcard can be flipped at a time across the game interface. When a player flips a Chitcard, the manager checks if there is already a flipped Chitcard. If so, it unflips the previously flipped card before allowing the new card to be flipped, maintaining a single active Chitcard for player interaction.

This flipping mechanism adds an element of memory and strategy to the gameplay, as players must remember the positions and images of Chitcards to progress effectively in the game.



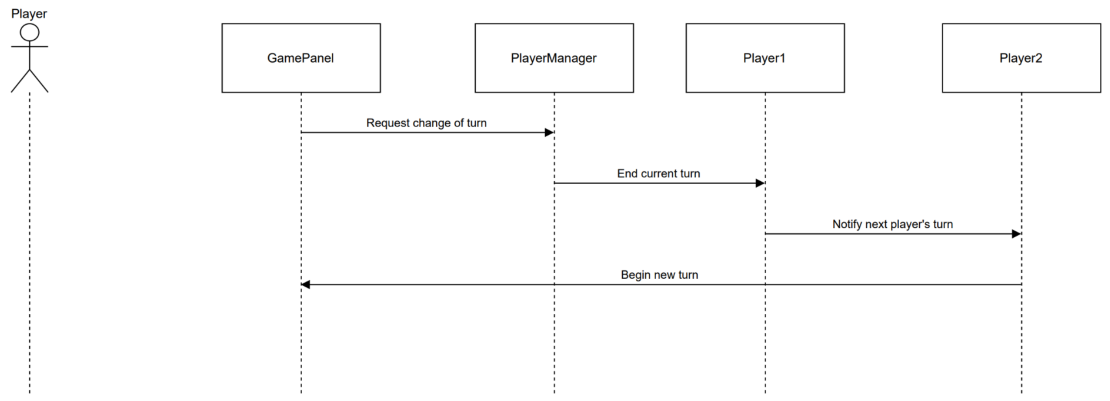
## 1.2.4 Movement of Dragon Tokens

When a player interacts with the game panel, the system checks the current position of the dragon tokens. It then communicates with the DragonToken component to inquire about the last flipped Dragon 1 card. The ChitCard component responds to this query, providing information about the last flipped card to the game panel. Based on predefined game rules, the game panel determines the movement of the dragon tokens. Subsequently, the DragonToken component updates the player's token position, reflecting the movement on the game interface.


Player
GamePanel
DragonToken
ChitCard
Check current position
Check last flipped Dragon 1 card
Notify of last flipped card
Determine movement based on rules
Update token position

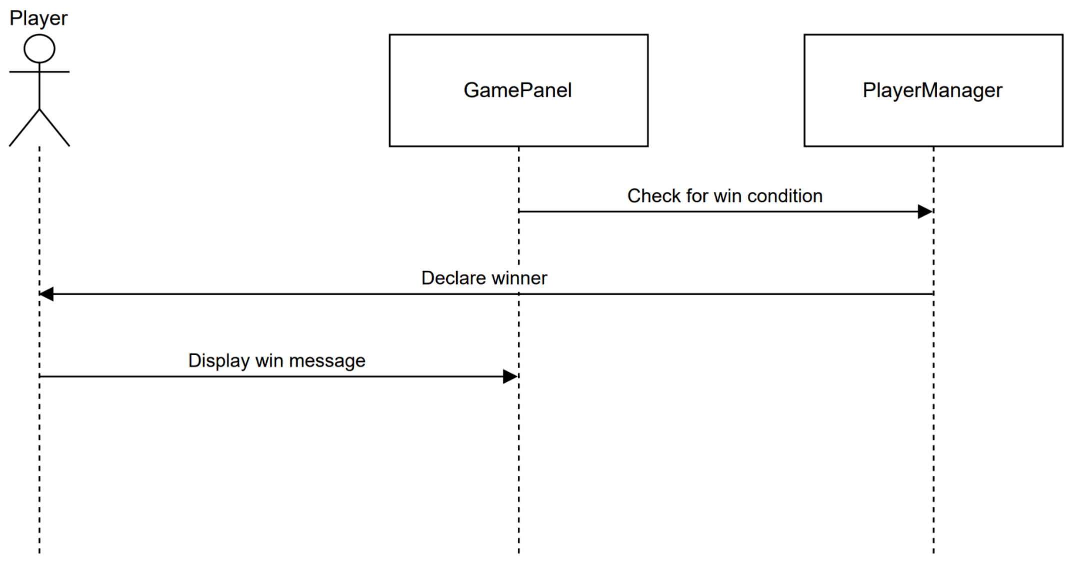
### 1.2.5 Change of Turn to the Next Player

During gameplay, when a turn change is required, the game panel sends a request to the PlayerManager component to facilitate the transition. The PlayerManager communicates with the current player (e.g., Player 1) to conclude their turn. Player 1 then notifies the next player (e.g., Player 2) about their upcoming turn. Player 2 acknowledges this notification, and the game panel initializes the new turn for Player 2, allowing them to proceed with their actions and decisions.



## 1.2.6 Winning the Game

When the player reaches their cave, the PlayerManager component analyzes the game state and determines the winner, signalling this outcome to the respective player. The player receives a notification of their victory, and the game panel displays a congratulatory message, indicating the successful conclusion of the game.


Player
GamePanel
PlayerManager
Check for win condition
Declare winner
Display win message