Use Cases

ID:	**0**
Title:	Player's turn
Description:	This describes what happens during a player's turn. It describes how the player interacts with the frontend, and what the backend does in response.
Primary Actor:	Player
Preconditions:	It is the player in question's turn, they have three actions left.
Postconditions:	The player has used a maximum of three actions.
Main Success Scenario:	1. The GameView presents the player with options to travel, restock, fight fascism, and play a card. 2. The player chooses one of these options. 3. The GameView communicates this choice to the FrontendGameStateController. 4. The FrontendGameStateController makes an HTTP POST request to the "/action" endpoint. 5. The backend WebSocketGameController listens for calls to these endpoints. When the request from the FrontendCatController hits an endpoint, the FrontendCatController performs the specified action. (See separate use cases). 6. The backend GameStateController decrements the number of actions remaining for that Cat's turn 7. Steps 1-5 are repeated two more times (three times total). 8. The player rolls the dice (See "Roll Dice" use case). 9. The backend GameStateController updates the current turn (a new turn begins).
Extensions:	2a. The player chooses not to use the rest of their actions at any point. i. The player rolls the dice (See "Roll Dice" use case). ii. The backend GameStateController updates the current turn (a new turn begins).
Frequency of Use:	Continuous
Status:	Designing
Owner:	Whole team
Priority:	High priority

ID:	**1**
Title:	Restock
Description:	This describes what happens when a player selects the restock action during a turn.
Primary Actor:	Player
Preconditions:	It is the player in question's turn. The player has chosen to restock, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The player has restocked their resist cards and has used an action.

Main Success Scenario:	1. The backend WebSocketGameController receives a request to the “/action” endpoint. 2. The backend GameStateController creates a new RestockAction object. 3. RestockAction takes cards from the resistCardDeck until the Cat’s hand is full.
Extensions:	3a. Resist deck becomes empty at any point during restock i. The backend GameStateController reshuffles the resist deck by replenishing and randomizing card order. ii. RestockAction continues to take cards until the Cat’s hand is full.
Frequency of Use:	Minimum 0 times per turn, maximum 3 times per turn.
Status:	Designing
Owner:	Whole team
Priority:	High priority

ID:	**2**
Title:	Travel
Description:	This describes what happens when a player selects the travel action during a turn.
Primary Actor:	Player
Preconditions:	It is the player in question’s turn. The player has chosen to travel, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The player has moved their Cat to an adjacent planet and has used an action.
Main Success Scenario:	1. The player chooses to travel by clicking the travel button and selecting a planet. This sends an HTTP request to the “/action” endpoint with the selected planet’s position as a query parameter. 2. The backend WebSocketGameController receives this request. 3. The backend GameStateController creates a new TravelAction object. 4. TravelAction gets the Cat’s current planet from CatModel, and then gets the set of adjacent planets from PlanetModel. 5. The planet that has been requested to travel to is in this set of adjacent planets. 6. TravelAction updates the cat/player’s current planet to the target planet. 7. WebSocketGameController sends a success message to the FrontendGameStateController. 8. FrontendGameStateController updates the GameView to display the Cat on its new planet.
Extensions:	5a. The planet that has been requested to travel to is not in the set of adjacent planets i. WebSocketGameController sends a failure message to the FrontendGameStateController. ii. FrontendGameStateController updates the GameView to tell the player they cannot travel to the chosen planet. iii. Player tries again.
Frequency of Use:	Minimum 0 times per turn, maximum 3 times per turn.
Status:	Designing

Owner:	Whole team
Priority:	High priority

ID:	**3**
Title:	Fight Fascism
Description:	This describes what happens when a player selects the “fight fascism” action during a turn.
Primary Actor:	Player
Preconditions:	It is the player in question’s turn. The player has chosen to fight fascism, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The fascism level of the Cat’s current planet is decremented, and the player has used an action.
Main Success Scenario:	1. The backend WebSocketGameController receives a request to the “/action” endpoint. 2. The backend GameStateController creates a new FightFascismAction object. 3. FightFascismAction gets the Cat’s current planet and decrements its fascism level by 1.
Extensions:	3a. The planet’s fascism level is 0 (no more fascist tokens on the planet). i. The WebSocketGameController communicates this to the FrontendGameStateController. ii. The FrontendGameStateController updates the GameView, informing the user that they cannot fight fascism. iii. The player tries another action.
Frequency of Use:	Minimum 0 times per turn, maximum 3 times per turn.
Status:	Designing
Owner:	Whole team
Priority:	High priority

ID:	**4**
Title:	Play a card - Symbol
Description:	This describes what happens when a player plays a symbol card
Primary Actor:	Player
Preconditions:	It is the player in question’s turn. The player has chosen to play a card, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The player has used one of their resist cards, the player has used an action.
Main Success Scenario:	1. The backend WebSocketGameController receives a request to the “/action” endpoint, with the resist card’s position in the player’s hand as a query parameter. 2. The backend GameStateController creates a new PlayCardAction object. 3. PlayCardAction is passed the CardModel from the player’s hand

	4. PlayCardAction verifies that the conditions have been met. 5. The card's effect is executed by PlayCardAction, and the planet's fascism level is subtracted by 2.
Extensions:	4a. The condition is not met. i. PlayCardAction gets the player's current planet from CatModel. ii. PlayCardAction compares the symbol from the card with that of the player's planet. iii. The symbols do not match so the condition is not met. iv. The effect is not resolved, WebSocketGameController notifies the FrontendGameController that no action occurred.
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**5**
Title:	Play a card - "+1 Liberation"
Description:	This describes what happens when a player plays a "+1 liberation" card
Primary Actor:	Player
Preconditions:	It is the player in question's turn. The player has chosen to play a card, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The player has used one of their resist cards, the player has used an action.
Main Success Scenario:	1. The backend WebSocketGameController receives a request to the "/action" endpoint, with the resist card's position in the player's hand as a query parameter. 2. The backend GameStateController creates a new PlayCardAction object. 3. PlayCardAction is passed the CardModel from the player's hand 4. PlayCardAction verifies that the conditions have been met. 5. The card's effect is executed by PlayCardAction.
Extensions:	4a. The condition is not met. i. PlayCardAction gets the player's current planet from CatModel. ii. PlayCardAction checks that the fascism level of that planet is less than or equal to 0 (no fascist tokens on the planet). iii. The planet's fascism level is greater than 0, so the condition is not met. iv. The effect is not resolved, WebSocketGameController notifies the FrontendGameController that no action occurred.

Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**6**
Title:	Play a card - “Heal 1”
Description:	This describes what happens when a player plays a “Heal 1” card
Primary Actor:	Player
Preconditions:	It is the player in question’s turn. The player has chosen to play a card, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The player has used one of their resist cards, the player has used an action.
Main Success Scenario:	1. The backend WebSocketGameController receives a request to the “/action” endpoint, with the resist card’s position in the player’s hand as a query parameter. 2. The backend GameStateController creates a new PlayCardAction object. 3. PlayCardAction is passed the CardModel from the player’s hand 4. PlayCardAction verifies that the conditions have been met. 5. PlayCardAction finds all cats on that planet, and filters out cats without scratches. 6. PlayCardAction communicates this list of cats to FrontendGameStateController. 7. FrontendGameStateController updates the GameView to display the list of cats. 8. The user selects a cat, which triggers an HTTP post request to the “/action” endpoint, specifying the selected cat. 9. The backend WebSocketGameController receives the request 10. The backend GameStateController decrements the number of scratches for the selected
Extensions:	4a. The conditions are not met. i. Nothing happens
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**7**
Title:	Play a card - “Heal 2”
Description:	This describes what happens when a player plays a “Heal 2” card
Primary Actor:	Player
Preconditions:	It is the player in question’s turn. The player has chosen to play a card, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The player has used one of their resist cards, the player has used an action.
Main Success Scenario:	1. The backend WebSocketGameController receives a request to the “/action” endpoint, with the resist card’s position in the player’s hand as a query parameter. 2. The backend GameStateController creates a new PlayCardAction object. 3. PlayCardAction is passed the CardModel from the player’s hand 4. PlayCardAction verifies that the conditions have been met. 5. PlayCardAction gets the player’s current planet from CatModel. 6. PlayCardAction finds all cats on that planet, and filters out cats without scratches. 7. PlayCardAction communicates this list of cats to the FrontendGameStateController. 8. The FrontendGameStateController updates the GameView to display the list of cats. 9. The user selects a cat, which triggers an HTTP post request to the “/action” endpoint, specifying the selected cat. 10. The backend WebSocketGameController receives the request. 11. The backend GameStateController decrements the number of scratches for the selected cat. 12. The user selects another cat, which triggers an HTTP post request to the “/action” endpoint, specifying the selected cat. 13. The backend WebSocketGameController receives the request. 14. The backend GameStateController decrements the number of scratches for the selected cat.
Extensions:	4a. The conditions are not met. i. Nothing happens
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**8**
Title:	Play a card - “-2 Fascist”

Description:	This describes what happens when a player plays a “-2 Fascist” card
Primary Actor:	Player
Preconditions:	It is the player in question’s turn. The player has chosen to play a card, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The player has used one of their resist cards, the player has used an action.
Main Success Scenario:	1. The backend WebSocketGameController receives a request to the “/action” endpoint, with the resist card’s position in the player’s hand as a query parameter. 2. The backend GameStateController creates a new PlayCardAction object. 3. PlayCardAction is passed the CardModel from the player’s hand 4. PlayCardAction verifies that the condition is met. 5. PlayCardAction gets the player’s current planet from CatModel 6. PlayCardAction makes a new FightFascismAction object, and uses it to subtract two from that planet’s facsism level
Extensions:	4a. The conditions are not met. i. Nothing happens
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**9**
Title:	Play a card - “Teleport”
Description:	This describes what happens when a player plays a “Teleport” card
Primary Actor:	Player
Preconditions:	It is the player in question’s turn. The player has chosen to play a card, and the client has communicated this decision to the backend using an HTTP request.
Postconditions:	The player has used one of their resist cards, the player has used an action.
Main Success Scenario:	1. The backend WebSocketGameController receives a request to the “/action” endpoint, with the resist card’s position in the player’s hand as a query parameter. 2. The backend GameStateController creates a new PlayCardAction object. 3. PlayCardAction is passed the CardModel from the player’s hand

	4. PlayCardAction verifies that the condition is met. 5. PlayCardAction gets the player's current planet from CatModel 6. PlayCardAction makes a new TravelAction object, which is then used to set the cat's current planet to the target planet
Extensions:	4a. The conditions are not met. i. Nothing happens
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**10**
Title:	Roll dice
Description:	This describes what happens when a player must roll the dice at the end of their turn.
Primary Actor:	Player
Preconditions:	It is the player in question's turn. The player is done using their actions.
Postconditions:	The player has rolled the dice, the fascism level has increased for each corresponding planet.
Main Success Scenario:	1. The fascism scale is between 7 and 12 (inclusive), so the backend GameStateController determines that 3 dice should be rolled. 2. The backend GameStateController generates 3 random numbers between 1-12. 3. The backend GameStateController increases the fascism levels for the planets at the positions given by the random numbers. 4. Using WebSockets these numbers are communicated to the FrontendGameStateController. 5. FrontendGameStateController updates the GameView to display the correct number of dice to the player. 6. The player clicks to roll the dice. 7. The FrontendGameStateController updates the GameView to display the 3 previously generated random numbers to the screen. 8. The GameView is updated by the FrontendGameStateController on the next turn cycle.

Extensions:	1a. The fascism scale value is between 1 and 6 (inclusive). i. The player rolls 2 dice instead of 3. 2a. Any of the randomly generated numbers are between 9 and 12 (inclusive). i. The backend GameStateController increases the fascism levels for the planets at the positions given by the random numbers. ii. The backend GameStateController automatically draws a GalaxyNewsCard for the player - see galaxy news card use cases.
Frequency of Use:	Once per turn
Status:	Designing
Owner:	Whole team
Priority:	High priority

ID:	**11**
Title:	Galaxy news card drawn - Effect_A
Description:	This describes what happens when a player draws a fascism scale card
Primary Actor:	Player
Preconditions:	The backend GameStateController has automatically drawn a galaxy news card
Postconditions:	The fascism scale has increased by one
Main Success Scenario:	PlayCardAction increases the fascism scale by 1.
Extensions:	none
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**12**
Title:	Galaxy news card drawn - Effect_B
Description:	This describes what happens when a player draws a planet card
Primary Actor:	Player
Preconditions:	The backend GameStateController has automatically drawn a galaxy news card
Postconditions:	Facism value on the planet and the scratches on each cat on the planet is increased
Main Success Scenario:	1. PlayCardAction increases the fascism value of the planet the player is on. 2. PlayCardAction applies one scratch to each cat on the player's planet
Extensions:	none
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**13**
Title:	Galaxy news card drawn - Effect_C
Description:	This describes what happens when a player draws a discard card
Primary Actor:	Player
Preconditions:	The backend GameStateController has automatically drawn a galaxy news card
Postconditions:	Facism value on planet is increased and cards are removed from players hand
Main Success Scenario:	1. PlayCardAction increases the fascism scale value by 1. 2. PlayCardAction removes all cards from the player's hand.
Extensions:	none
Frequency of Use:	0-3 times per turn
Status:	Done

Owner:	Whole team
Priority:	High

ID:	**14**
Title:	Galaxy news card drawn - Effect_D
Description:	This describes what happens when a player draws an annihilation card
Primary Actor:	Player
Preconditions:	The backend GameStateController has automatically drawn a galaxy news card
Postconditions:	Fascism scale and planet fascism are each increased by one, scratch applied to each cat on planet
Main Success Scenario:	1. PlayCardAction increases the fascism scale value by 1 2. PlayCardAction increases the fascism value of the planet the player is on by 1 3. PlayCardAction applies one scratch to each cat on the player's planet
Extensions:	none
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**15**
Title:	Galaxy news card drawn - Effect_E
Description:	This describes what happens when a player draws a return card

Primary Actor:	Player
Preconditions:	The backend GameStateController has automatically drawn a galaxy news card
Postconditions:	player's cat is on their home planet, has +2 scratches
Main Success Scenario:	1. PlayerCardAction applies scratches to the player 2. PlayCardAction sets the player's current planet to its home planet.
Extensions:	none
Frequency of Use:	0-3 times per turn
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**16**
Title:	Lose the game
Description:	This describes what happens when the game is lost
Primary Actor:	Player
Preconditions:	Either of the following 1. The fascism scale reaches a value of 14 2. The fascism level of 3 different planets has reached 4
Postconditions:	Game is over
Main Success Scenario:	WebSocketGameController communicates "lose" message to FrontEndGameController
Extensions:	none
Frequency of Use:	0-1 times per game
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**17**
Title:	Win the game
Description:	This describes what happens when the game is won
Primary Actor:	Player
Preconditions:	Four planets with unique symbols have reached a fascism level of -4
Postconditions:	Game is over
Main Success Scenario:	WebSocketGameController communicates “win” message to FrontEndGameController
Extensions:	none
Frequency of Use:	0-1 times per game
Status:	Done
Owner:	Whole team
Priority:	High

ID:	**18**
Title:	Complete a meowssion
Description:	This describes what happens when a meowssion is completed
Primary Actor:	Player
Preconditions:	The conditions described in the meowssion have been met
Postconditions:	A new Meowssion is selected
Main Success Scenario:	1. A bonus card is drawn and its effect is resolved
Extensions:	1a. i) A meowssion has already been completed ii) The gameStateController records that tracks this, allowing the game to be won once another win condition is met.
Frequency of Use:	0-1 times per game
Status:	Done
Owner:	Whole team
Priority:	High