ConKUeror: R1M1

Introducing Nerd^5

Nerd^5 — A COMP302 project team comprised of passionate and dedicated individuals who are committed to creating innovative and cutting-edge software solutions. With their combined knowledge and expertise, they strive to push the boundaries of what is possible in the world of technology, exploring new horizons and breaking new ground. Lead by *the self-proclaimed leader* **Altun Hasanli**, and with the expert advice of **Professor Attila Gursoy** and our TA, **Damla Ovek**, they are determined to make a lasting impact on the world by transforming ideas into reality.

Team Members

- **Altun Hasanli** (Team Leader) (ahasanli19@ku.edu.tr)
  🏢 *Administrative Department*
- **Ömer Nadir Civelek** (ocivelek19@ku.edu.tr)
  ⚡ *Agility Department*
- **Mert Balçık** (mbalcik20@ku.edu.tr)
  💡 *Idea Department*
- **Kazım Toprak Güler** (kguler20@ku.edu.tr)
  🎨 *Visuals Department*
- **Zindan Kurt** (zkurt18@ku.edu.tr)
  💬 *Communications Department*

ConKUeror Github Link
https://github.com/altunh/2023S_nerd5

Contents of this document

- Game Specification
  - ConKUeror Glossary
  - Technical Glossary
  - Design Principles
  - Constraints
  - Implementation Principles / Patterns
  - User Interface (View)
  - System (Controller)
  - Domain Model
- User-System Interactions
  - Use Cases
  - System Sequence Diagrams
  - Operation Contracts

Vision

“*Victory comes from finding opportunities in problems.*”
— Sun Tzu, Art of War

Team Nerd^5 gather to dicuss the game — **ConKUeror.** *Generated by AI.*

Le Début

Inception

ConKUeror is a competitive multiplayer strategy game that offers an exciting opportunity for strategy gaming enthusiasts to experience the thrill of devising plans to conquer and invade countries in a virtual multiplayer world. The game combines fun and challenge, allowing players to battle it out for world domination, while challenging their strategic thinking and decision-making skills, making it a fun and educational experience for all ages. **ConKUeror** can be played across multiple platforms including *Windows*, *MacOS*, and *Linux*, making it accessible to a wider audience. In today's world where gaming has become a popular way of entertainment for people of all ages and backgrounds, **ConKUeror** presents a very appealing opportunity for all sorts of interest groups, *or stakeholders*, to capitalize on a growing market of gamers who enjoy strategy games that require critical thinking and tactical planning.

ConKUeror allows players to conquer countries, invade territories, and send soldiers to defend their borders. The game offers a unique and immersive gaming experience that is both challenging and entertaining. With a wide range of customizable features and gameplay options, **ConKUeror** is designed to keep players engaged

and entertained for hours on end. From new players to seasoned strategy-game veterans, **ConKUeror** offers something for everyone.

Medieval generals planning an invasion — *generated by AI.*

Parties intéressées

Interested parties

Our stakeholders include **players**, **game developers**, **investors**, and **advertisers**. The players are the primary stakeholders who participate in the game and provide feedback on the game's features and performance. The game developers are responsible for designing, developing, and maintaining the game. Investors provide financial support for the development and marketing of the game, while advertisers sponsor the game and advertise their products or services to the players.

- **Players**: The players' primary interests are to have a fun and engaging gaming experience while improving their strategic thinking and decision-making skills. They want the game to be challenging, exciting, and fair.
- **Game Developers, i.e. Nerd^5**:
  The game developers' primary goal is to design and develop a high-quality game that meets the players' expectations. They want the game to be user-friendly, visually appealing, and bug-free.
- **Investors**: The investors' primary goal is to see a return on their investment. They want the game to be successful and generate revenue through in-game purchases, sponsorships, and advertising.
- **Advertisers**: The advertisers' primary goal is to reach the game's players and promote their products or services. They want the game to have a large player base and a high engagement rate.

Folks of the last supper (Jesus in the middle) amazed by the mobile version of ConKUeror — generated by AI.

Proposition d'affaires

Business Proposition

We envision a multitude of ways we can generate revenue using **ConKUeror,** of course without violating the core principles of our game, including but not limited to marketplaces like Steam, in-app purchases and non-invasive in-app advertisement.

“*Great results can be achieved with small forces.”*
— Sun Tzu, Art of War

Game Marketplaces

With the amount of work and dedication poured into the game, it would definitely be a sin not to seek revenue from the game, especially if it allows our team to further improve the game. The marketplace of choice for desktop gaming is Steam, which is popular among all players. Being offered on Steam, it would definitely help garner a lot of attention that we most certainly need. We think it will be a valuable offer for gamers of all ages. If we decide to pursue this option, we aim to choose a fair price, to provide best value for the buck.

*“Software is like s**. It’s better when it’s free.”*
— Linus T., creator of Linux

In-app Purchases

The game is based around a single game session, where all the action happens. However, we are considering the possibility of adding daily rewards, bonuses, custom characters and all sorts of different add-ons to monetize the game. We could, for example, have extra characters that you can purchase with game currency, which you purchase with your credit card. This is an especially popular monetization method for mobile games, which means we won't be short of business ideas, if decide to create a mobile version for *iOS* and *Android*.

In-app Advertisements

Advertisements would be very out-of-place, in games center around a fun and long-term gameplay experience such as **ConKUeror.** It's an approach that we do not favor, but still would consider, if it will yield the same revenue as the above two, and won't spoil the game experience.

Introducing the game

Introducing ConKUeror

ConKUeror is a fun and exciting strategy game inspired by the popular game *RISK*. In this game, players compete to conquer the world by launching daring attacks, defending themselves on all fronts, and sweeping across vast continents with boldness and cunning. However, be careful! Your opponent may strike and take away your progress just when the world is within your grasp!

Players battling for dominance — *generated by AI.*

The game is played on a political map of the world, divided into territories and grouped into continents. Players control armies of playing pieces with which they attempt to capture territories from other players, with results determined by dice rolls. Players may form and dissolve alliances during the course of the game, adding to the excitement. The goal of the game is to occupy every territory on the board and, in doing so, eliminate the other players.

So why not gather your friends and start playing today?

Troops on different territories, ready to attack — *generated by AI.*

Game Design

- **ConKUeror** is designed to be played on a single device by 2-6 players. *We do not support network gaming as of yet.*
- **ConKUeror** can either be played by a single player against the computer, or by multiple human players on the same device. The user can specify exactly how many human and computer players there will be.
- **ConKUeror's** design allows it to be cross-platform, meaning *Windows*, *MacOS* and *Linux* users can get the same gaming experience.

An army piece and die — *generated by AI.*

Player rearranging armies — *generated by AI.*

Players in medieval costumes over the board game battling for dominance — *generated by AI.*

“alea iacta est”, roman saying — *the die has been cast.*

Game Objects

World Map

Game is built around a world map, consisting of 6 continents divided into 39 territories. Each continent is a different color and contains 6 territories. Some territories share a border between them, and some are connected by what’s called a sea-line. Yet, some territories are separated by a sea, but not connected by a sea-line. Armies are stationed directly on the territories for attack/defense.

Game Die

The game uses a six-sided die to determine the outcomes of attacks and who goes first in the first round of the game.

Armies

Players battle for world domination using armies denominated by three distinct types: *Infantry*, *Cavalry*, and *Artillery*. Armies can be exchanged at any point in the game.

Game Deck

There are three kinds of cards in the game: *territory cards*, *army cards*, and *chance cards*. Chance cards are shuffled and form a separate deck. Territory and army cards are shuffled together, making the count of decks in the game two. At the beginning of any turn, the player picks a chance card, and at the end of any turn in which the player has captured at least one territory, they earn one territory or army card from the deck.

Territory Cards

There are as many territory cards as the number of territories in the game. During the game, the aim of collecting territory cards is to conquer a continent without attacking. When the player collects all territory cards of a continent, the player conquers all territories of that continent without a need to attack them.

Army Cards

Just like the armies, there are three types of army cards: *infantry*, *cavalry*, and *artillery*. For each player in the game, there are three infantry, two cavalry, and one artillery card. Players can trade army cards to gain an additional army if they have a set of 3 cards. They can place the new armies on any of their own territories.

Chance Cards

There are 10 unique chance cards, varying in the opportunities and losses they create for the players. These cards add randomness to the game, occasionally disrupting players' plans to test their flexibility.

Gameplay

Game Setup

Adding Players — in order to create a game environment, the user specifies the number of human and computer-controlled players. The user chooses a distinct username and color for all the players.

There is a fixed world map in the building screen. The user can enable/disable the continents and the territories. Disabled territories/continents cannot be owned, attacked, etc. Simply, they are not involved in the game. The users can build bridges (draw lines) between non-adjacent territories.

Enabling/Disabling Territories & Continents — users can also make changes to the map by enabling or disabling territories and entire continents. This means, they are not playable and no player can station troops on them.

Medieval armies on borders, preparing for attack — generated by AI.

Simulation of the world map in **ConKUeror** — generated by AI.

How the game starts

At the beginning of the game, depending on the number of players, each player gets a number of infantry pieces as indicated in the game specification. Each player rolls a die to determine who goes first, and the first player places exactly one army on a territory of his choice. After this, players take turns to place one army onto unoccupied territories until all territories are claimed. After this, the first round of the game begins, and the game continues until a player has reached complete world domination.

A single round of the game

Onset: Chance Cards

At the beginning of a round, the game might randomly decide to give the player a **chance card** from the deck. The chance card is applied immediately, unless the user decides to skip, in which case, he won't get it back.

"dura lex, sed lex", roman saying — *the law is harsh, but it is the law.*

During the round, each player completes the following steps *in this exact order*:

Phase 1: Draft

The player is given a number of armies depending on the number of territories he occupies, and they place it on the territories they own.

"*veni, vidi, vici", roman saying — I came, I saw, I conquered.*

Phase 2: Attack (Optional)

- The player has the option to attack territories adjacent (or connected by a sea-line) to one of their own. To attack a territory,

the attacking territory must have at least as much matching types of armies as the defending territory. For example, you cannot attack Cavalry pieces with Infantry pieces.

- To launch an attack, the player sends over some of their army pieces to the defending territory (the armies are sent only from the attacking territory). The player can attack and conquer as many territories, however much as he wishes.
- The outcome of an attack is decided by dice roll. Each player rolls a die, and the one with the highest roll wins. In case if the attacker loses, he loses 2 army pieces, Once the player defeats the last remaining army on a territory, he conquers it by moving army pieces there. *However, at least 1 army must be left on the attacking territory for defense.*
- At the end of a turn, player draws one card for each territory he conquered. As specified, player might decide to exchange army cards for armies, or territory cards for territories, or even infantry pieces with equivalent cavalry and artillery pieces. This finalizes the attack step before the fortify option.

Phase 3: Fortify (Optional)

As a final optional move, the player is allowed to move armies from exactly one of their territories to another territory.

Game Specification

> ℹ️ The following is a preliminary (Phase 1) specification, design and implementation requirements for the game **ConKUeror.** Some parts of the specification might be subject to change during the next phase of development.

ConKUeror Glossary

Game

- **ConKUeror**: Name of the game being specified, a strategy game inspired by RISK, in which players battle to conquer the world.
- **World Map**: The political map of the world on which the game is played, divided into territories and continents.
- **Grid**: A tool used to create the game map by dividing it into smaller sections or cells (continents or territories in this case).
- **Continent**: A group of territories on the game map that are geographically connected and share a common color.
- **Territory**: A single area on the game map that can be occupied or defended by players using armies.
- **Reachability**: A condition that ensures all territories on the game map are accessible by players, connecting territories either by the principle of adjacency or sea-line.

- **Sea-line:** An over-sea connection between two territories separated by a water body, making all territories accessible to the player, no matter where they are.
- **Die**: A tool used to determine the outcome of battles between players' armies. Also used to decide which player starts first.
- **Player:** Actors in **ConKUeror**, possessing the ability to occupy territories, fortify positions, trade army units, and play cards, making strategic decisions to win during a single round of the game. A typical **ConKUeror** game consists of 2-6 players (including game-controlled players).
- **Initial Sharing of Territories and Army Placement**: The process by which players initially claim territories and place armies onto them at the beginning of the game.
- **Game Loop:** The game loop in **ConKUeror** consists of a series of rounds, with each round comprising three phases: Draft, Attack (optional), and Fortify (optional). The game continues until a player has achieved complete world domination by occupying every territory on the board and eliminating all other players.
- **Round (or Turn):** A game state that allows each player to choose game actions, wherein after every player, the player to the left goes next, and chooses game actions until they decide they have reached their goals for that moment.
- **Draft:** A game phase in which a player receives a number of army pieces depending on the amount of territories they own, and the player is asked to station them on own territories.
- **Fortify**: A game phase in which a player strengthens their position by moving armies between adjacent territories.
- **Attack**: A game phase in which a player attempts to capture an opponent's territory by engaging in battles.
- **Defense:** A game rule that allows the armies of a territory under attack to defend itself, not directly triggered by the player occupying the territory under attack.
- **Winning Condition:** A condition, asserting that for a player to win and the game to be over, the player needs to conquer the whole map (all territories), and thus, eliminate all opponents (including the computer).

App/Interface

- **Login Screen**: A screen where players enter their unique nicknames before starting the game.
- **Help Screen**: A screen that provides information on game objects, features, and instructions on how to play.
- **Building Mode**: The initial phase of the game in which players set up the game board and configure the map.
- **Running Mode**: The main phase of the game in which players take turns moving armies, attacking and conquering territories, and fortifying their positions, and eliminating opponents.
- **Pause/Resume**: A feature that allows players to temporarily halt the game and continue it later.
- **Game Over Screen**: A screen displayed when the game ends, showing the winner.

Game Objects

- **Armies**: The game pieces used by each player, denominated by three different types, representing their military forces, used to occupy and defend territories, and engage in battles.

- **Infantry**: The basic army unit.
- **Cavalry**: Army unit worth 5 Infantry.
- **Artillery**: Army unit worth 10 *Infantry* or 2 *Cavalry*.
- **Cards**: Elements that influence gameplay, including territory, chance, and army cards.
- **Territory cards**: Cards for each territory on the game board. If a player collects cards for all territories of a continent, he can conquer the whole continent without needing to attack.
- **Army cards**: Cards for each type of different army units, namely infantry, cavalry and artillery. Used to gain an additional army if a player has a set of 3 such cards. Different combinations of these cards can be traded to gain different army units, determined by specific table.
- **Chance cards**: Cards that introduce random events and opportunities during a player's turn.
  See the "Chance Cards Mechanics" section for a list of all chance cards.

Technical Glossary

- **Java**: A popular programming language used for developing a wide range of applications.
- **JVM**: Java Virtual Machine, a virtual machine that executes Java bytecode.
- **Java Standard Libraries**: A set of libraries included with Java that provide a wide range of functionality.
- **Java Swing**: A GUI toolkit for Java used for developing desktop applications.
- **Class**: A blueprint or template for creating objects.
  Or equivalently `domain object templates` in the realm of domain modeling.
- **Object**: An instance of a class.
  Or equivalently `domain object` in the realm of domain modeling.
- **OOP**: Object-Oriented Programming, a programming paradigm that focuses on objects and their interactions.
- **Model-View separation**: A design pattern that separates the data from the presentation of the data.
- **MVC**: Model-View-Controller, a design pattern used in software engineering that separates the application into three interconnected components: the model, the view, and the controller.
- **Model**: The part of the MVC pattern that represents the application's data and business logic.
- **View**: The part of the MVC pattern that represents the presentation of the data to the user.
- **Controller**: The part of the MVC pattern that handles user input and interacts with the model and the view.
- **Creator pattern**: A design pattern used in software engineering that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.
- **Observer pattern**: A design pattern used in software engineering that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- **Domain Model**: A conceptual model that represents the real-world entities and their relationships in a particular domain.

- **User:** A user is an individual who interacts with a system, such as a computer program, website, or application. In the context of this document, a user would refer to someone playing or interacting with the **ConKUeror** game.
- **Use Cases**: Descriptions of how a user interacts with a system to achieve a specific goal.
- **System Sequence Diagrams**: Diagrams that show the interactions between an actor and a system to achieve a specific use case.
- **Operation Contracts**: Formal specifications of the preconditions, postconditions, and invariants of an operation in a software system.

Design Principles

The infamous cover of the book, Structure and Interpretation of Computer Programs, but programmers are receiving messages from God — edited by AI.

Ease of Use

The player must have an easy-to-use UI and intuitive controls to interact with game objects. Note that **ConKUeror** is a computer game based on a board game called RISK, and by design, it's a strictly game-controlled environment. For users to understand such a game with a lot of moving parts, a simple help page is not enough. This type of design requires intentional use of attention-grabbing icons to represent objects such as the game deck, cards, chance card effects and so on, as well as a straightforward user panel that displays all possible actions wherever it's applicable and relevant. This is a principle we actively exploit in designing the game.

Adaptability

This type of a game environment requires a lot of captivating system messages and announcements. For example, when the game gives a chance card, the map is blurred out to divert attention to the chance card and its content, the message also shows big and bulky buttons that are relevant to that specific chance card, and also a skip option. The panel always shows what's relevant to the current game setting and phase. For example, during a player's turn, there are easy-to-see buttons to exchange armies, to use army and territory cards only when you satisfy the requirements to do such an exchange), so that the user sees only what he can do, rather than seeing all game options at once.

God is pondering whilst writing his first computer program — generated by AI.

Configurability

The game adapts under different number of players and different user preferences — game map can be modified to suit the needs of the players, and depending on the number of players, **ConKUeror** is designed to utilize different rule sets to provide a fair experience for the end user.

Reliability

Like any other application, **ConKUeror** is also not spared of bugs and errors. In case, if an error occurs, system should show a message with

the error on screen, with a button to send the error to the developer, save the game and restart it, so that no user data is lost mid-game.

For security and reliability reasons, the game should be built using one of the most popular and advanced languages in the developer world — Java. This makes it extremely reliable and secure to write system and architecture-independent game, and eliminates unexpected behavior.

Performance

As mentioned, the game should be built using one of the most potent and high-performing languages out there, utilizing a fast and flexible core library, making the game fast even on the poorest of machines. Performance is also a priority when it comes to the User Interface and animations. Being a 2D board game, **ConKUeror** uses another reliable java library — swing — to implement all the custom UI objects, animations, and user interactions, minimizing the risk of unnecessary use of resources.

Game map with noticeably distinct colors for visual aid — generated by AI.

Cross-Platform Design

Use of Java also helps it to be cross-platform and run on any OS that supports JVM. This is an especially helpful factor, since it would make it possible to develop a network gaming option in the future, allowing players with different machines and different OSs play together and enjoy the same gaming experience.

Modularity

The game should be packaged packaged in such a way as to make it easy to extend and customize. This would make it, for example, easy to add further chance cards and army types, new countries and continents, or even different game maps and themes.

Constraints

ConKUeror utilizes only java standard libraries and swing. The UI is also designed with custom components using the very same libraries.

Implementation Principles / Patterns

Object-Oriented Programming

ConKUeror is being developed with the object-oriented programming (OOP) principles in mind. This means that the code is organized into objects that represent the game's entities, such as the game map, the game deck, and the player objects. OOP is also aligned with our “Modularity” principle, which means that the code is easier to read, maintain, and extend.

Some common OOP patterns used in the development of **ConKUeror** include **inheritance** and **polymorphism**. Inheritance is used to create subclasses of the army objects, such as the Infantry, Cavalry, and Artillery objects. **Polymorphism** is used to allow the game to handle different types of objects in a consistent way, such as when determining the outcome of an attack. Use of OOP principles

allows for a more organized and efficient development process, resulting in a better game for the end user.

Model-View Separation

In the development of **ConKUeror**, we utilize the Model-View-Controller (MVC) design pattern. This pattern separates the application logic into three interconnected components: the model, the view, and the controller.

- **Domain Model: Model + Controller**
  Our model and controller are combined in one layer we call the **Domain Model,** as specified in the section with the same name. All the underlying data and game logic — the model — are implemented in game objects, while the `Game` object of the domain is the single controller that acts as the intermediary between the model and the view. `Game` manages game flow, player turns, game state transitions, and so on, actively utilizing the **Observer Pattern**, as specified above.
- **User Interface: View**
  The view is the UI abstraction layer, as specified in the next section, responsible for rendering the model data to the user interface, including components such as the world map, the cards, armies and interactive components like the buttons and inputs.

By separating the model from the view and the controller, we have created a modular and flexible design that allows for easier maintenance and extension of the game.

Creators

Special factory methods should be used across the domain model, to handle object creation at runtime, which are triggered by `Game` as part of the game loop.

Observers

Special checks should be performed across the domain model, which in turn notify `Game` about state changes, triggering UI updates.

User Interface: The View

As mentioned, the user interface is an abstraction layer separate from the Domain Model in accordance with the Model-View separation principle.

App Screens

These are the five different screens of the application:

- **Login screen**: Appears before the game starts. Has options to either create a new game or continue the previous game.
- **Game screen**: The screen with the world map and the user panel.
- **Menu screen**: When opened, pauses the game. Has the following options: Help, Save, Save & Exit.
- **Help screen**: Shows a list of most important rules with the option to go back to the Menu Screen.
- **Game Over screen**: Appears when the game ends, shows the winner, and lists other players as runner-ups in order of when they lost the game. Has options to play again or quit to Login Screen.

World Map

The map shows all the territories and the continents, and the sea-lines that connect them.

- **Color:** Initially all territories with no owner are gray. But when conquered, the territory changes to the color of the player.
- **Armies:** Numbers are shown on each territory, indicating the armies stationed on them.

World Map of **ConKUeror** based on our specification. 39 territories in total.
Doesn't include the number of armies, or the game panel. Designed by Nerd^5.

Map Grid

The Map consists of **39 territories**. The following is a list of all continents and territories in **ConKUeror:**

Europe (9)

- Great Britain
- Iceland
- Scandinavia
- Northern Europe
- Western Europe
- Eastern Europe
- Southern Europe
- Anatolia
- Persia

Asia (7)

- Russia
- Mongolia
- Japan
- China
- India
- Middle East
- Siam

Africa (6)

- North Africa
- Egypt
- East Africa
- Congo
- South Africa
- Madagascar

Oceania (4)

- Indonesia
- New Guinea

North America (9)

- Alaska
- Greenland

South America (4)

- Venezuela
- Peru

- Western Australia
- Eastern Australia
- Northwest
- Alberta
- Ontario
- Quebec
- Western America
- Eastern America
- Central America
- Brazil
- Argentina

Map Actions

When clicked on a territory, depending on the phase and current action different options are shown.

- In the draft phase, it shows an input prompt to specify how many armies to deploy.
- In the attack phase, it shows all possible territories that can be attacked from there.
    - After an opponent territory is selected, you are asked to specify a number of armies (the input starts from 2).
    - After a territory is conquered, it prompts you to specify how many soldiers to send to the newly conquered territory.
- In the fortify phase, it shows possible territories where troops can be moved.

User Panel

The user panel is a layer of different buttons and inputs on top of the world map. It is designed to align with our “Adaptability” principle. The user panel is divided into the following sections:

- **Game Menu**: Pause/Resume and other options of the Menu Screen.
- **Players List:** all players are shown with their names and avatars in turn order.
- **Turn Control (dynamic):** shows the current phase of the turn, and has a button to go to the next phase
- **Game Notifications (dynamic):** used to notify the user of possible actions
    - In the draft phase, shows how many armies are left to place on the map.
    - Shows an icon when there’s a complete set of territory cards, indicating a possibility of exchange. When clicked, conquers the continent for the user and prompts to add armies to the territories of the continent.
    - Shows an icon when there’s a complete set of army cards, indicating a possibility of exchange. When clicked, prompts user to select a territory to do the exchange.
    - Shows if there are active “Chance Effects” applied for that turn.

Game Notifications

During an illegal or disallowed move (or action), a small message indicator appears at the top right corner of the screen, indicating the appropriate warning message.

Avatars and other visuals

All UI components, including buttons, dialogs, inputs and also avatars are designed in comic-book style, with big and bulky sizes to align with our “Ease of Use” principle.

Here are some of the custom avatars we created using AI. In total there will be 20 different avatars to choose from.








System: The Controller

As specified, the `Game` object — or as we call it, the **System** — is the controller of the MVC pattern. In our diagrams, we actively use the name, **System,** to in describing the interactions between the user and the application. This abstraction layer handles game states, the game loop, as well as, all other user-application interactions, delegating all logical tasks to the individual **Model Objects.**

Different Game Modes

- **Building Mode**: allows the user to add players and make changes to the map (i.e. enable/disable territories and continents).
- **Running Mode**: main part of the game where turns take place.

Adding Players

In the building mode,

- User adds players and specifies whether each player is played by a person or the computer. *However, at least 1 player must be computer-controlled.*
- User chooses an avatar, username and a color for the player. *There are 6 colors to choose from: Blue, Red, Yellow, Purple, Green, Magenta. There are also 20 different avatars to choose from.*

Game-controlled Players

Game-controlled players are represented just as any other player, except that they have pre-programmed ways of interacting with the game objects, defined by straightforward set of rules and constraints.

Enabling/Disabling regions

In the building mode, you can enable/disable territories or entire continents on the world map. These continents or territories will not be playable in any way, and **any interaction with them is disallowed**.

Different Game Actions

There are various, what we call “game actions” that determine game functionality, changing how the controller behaves. These actions are the interactions that the controller is “expecting” the user to make, divided into different types of “sub-actions”:

- **Action: Next Phase**
  - During a Draft Phase
  - During an Attack Phase
- **Action: Territory Selection**
  - **Sub-action:** Own Territory
    - During a Draft Phase
    - During an Attack Phase
    - During a Fortify Phase
    - After an army card exchange
    - During a relevant chance card: Diplomatic Immunity, Revolt, Reinforcements, Mercenaries
  - **Sub-action:** Opponent Territory
    - During an Attack Phase
    - During a relevant chance card: Nuclear Strike, Revolution, Sabotage
- **Action: Army Number Selection**
  - During a Draft Phase
  - During an Attack Phase
    - For an attack to place
    - After a territory is conquered
  - During a Fortify Phase
- **Action: Dice Roll**
  - During an Attack Phase
  - During a relevant chance card: Sabotage, Mercenaries, Reinforcements
- **Action: Army Exchange Outcome Selection**
  - When you select army exchange in a certain territory
- **Action: Army Card Exchange Outcome Selection**
  - When you select to do army cards exchange

Initial Armies

When the game starts, depending on the number of players, each player gets equal armies, using the following rules:

- **2 players** ⇒ 40 Infantry
- **3 players** ⇒ 35 Infantry
- **4 players** ⇒ 30 Infantry
- **5 players** ⇒ 25 Infantry
- **6 players** ⇒ 20 Infantry

Army Exchange

At any point during their round, players can exchange their armies using the following rules:

- **1 Cavalry** ⇒ 5 Infantries
- **1 Artillery** ⇒ 2 Cavalries or 10 Infantries

Round Phase Mechanics

During a turn of the player, there are 3 phases to be completed: Draft, Attack, Fortify. The last two are optional, but need to be signaled by the user to skip.

- During a draft, the player places all armies given to him (and can continue only if he places all armies). At any time, he can select a territory to do army exchanges and army card exchanges.
- During an attack, the player first chooses his own territory, and then an opponent's territory. If he is allowed to perform the attack, he chooses the number of armies to send over to opponent's territory. As before, he can select a territory to do army exchanges and army card exchanges.
- During a fortify phase, the player first chooses a territory to send armies from, a territory to send armies to, and then the number of armies to send. After this, the player's turn is automatically concluded.

Game Dice

During an attack phase or a relevant chance card, a dice roll is expected to take place (if there is no chance card effect to roll two dice).

Attack Mechanics

During the attack phase, the player first chooses an own territory:

- If own territory has less than 3 armies, the territory is not selected, and a notification is sent to the user.

And then an opponent's territory:

- If the opponent territory doesn't satisfy the "**Army Match"** requirement, system sends a notification to the user.

Army Match Requirement: *the attacker's territory must have at least as much armies of each type as the defender.*

And then the user specifies the number of armies to send:

- The number of armies to send cannot be less than 2.

During the attack, two die are rolled for the attacker and the defender.

- If the attacker rolls a smaller number, he loses two armies that he sent over.
- If the defender rolls a smaller number, he loses one army that defends the territory.

If the final attack, wipes out all the armies on a territory, the player proceeds to conquer the territory, by specifying the number of armies.

Game Deck

At the beginning of the game, a deck consisting of army and territory cards are shuffled and added to the game. At the end of each round, if a player conquers a territory, a random card is drawn.

Army Cards
The number of army cards is determined by the system using the following formula:
[Number of Army Cards] = [Number of Players] x (3 Infantry + 2 Cavalry + 1 Artillery)

Territory Cards
There are 39 territory cards, each denoting a unique territory on the map.

If the territory and army cards end before the game ends, they are shuffled and used again. Therefore, a player who has conquered a continent using territory cards may lose that continent to another player later in the game.

Chance Cards
There are 10 unique chance cards, each with a unique feature.

Army Card Exchange

At any point during their round, if players have a complete set of 3 army cards, they can exchange them using the following rules:

- **3 Infantry Cards** ⇒ 1 Cavalry
- **2 Infantry Cards + 1 Cavalry Card** ⇒ 2 Cavalry
- **2 Infantry Cards + 1 Artillery Card** ⇒ 2 Artillery
- **1 Infantry Card + 2 Cavalry Cards** ⇒ 1 Cavalry + 1 Artillery
- **1 Artillery Card + 2 Cavalry Cards** ⇒ 3 Artillery

Territory Card Exchange

There are 39 territory cards for each territory on the map, when a set of territory cards for a specific continent is completed, system sends a notification to the user, about a possible exchange. When a player decides to use the complete set, the whole territory is conquered by the player, and the user is prompted to station troops on them. The game cannot continue, until the user decides to send at least one army to each newly conquered territory.

Chance Card Mechanics

Chance cards are given by the system randomly. The user holds the right to skip them if necessary. The following are the 10 unique types of chance cards:

- **Revolt**: Play this card on your turn to remove all armies from one of your territories and add them to another.

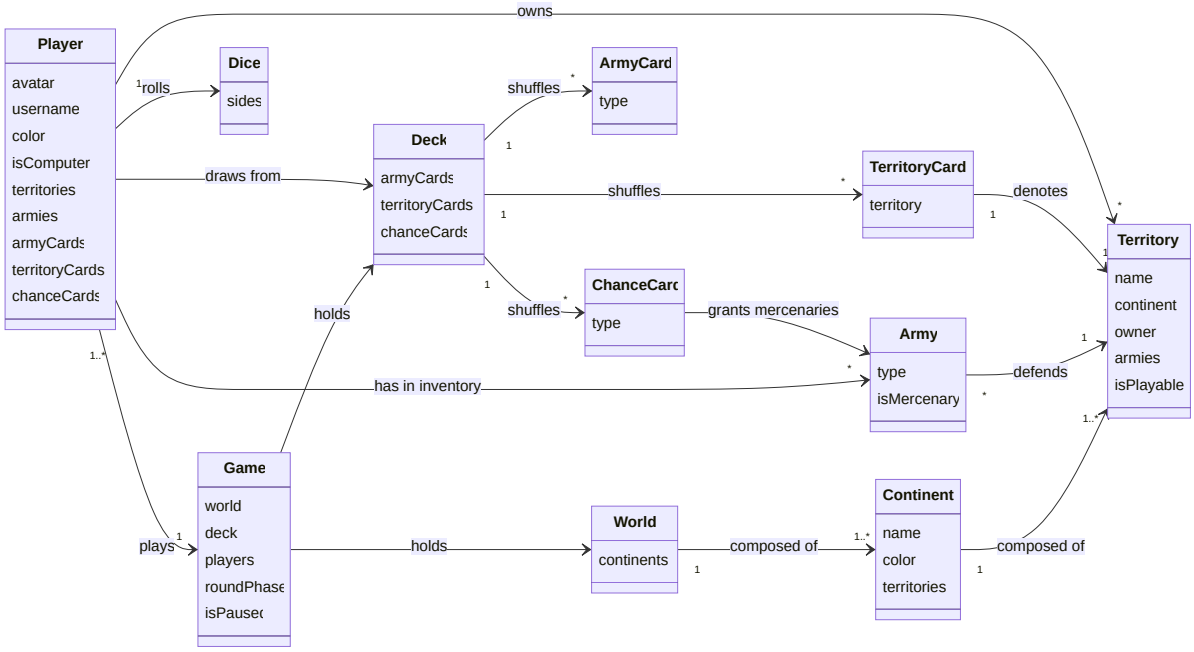
- **Nuclear Strike**: This card allows you to wipe out all armies in one territory, regardless of how many there are, but at the cost of destroying one of your own territories as well (chosen randomly by the game).
- **Sabotage**: This card allows a player to choose one territory belonging to another player and remove a certain number of armies from it. The number of armies removed is determined by a roll of dice.
- **Diplomatic Immunity**: This card allows a player to protect one of their territories from attack for one turn. No other player can attack that territory during that turn.
- **Mercenaries**: This card allows a player to hire a group of mercenaries to fight for them. Number of mercenaries are determined by a single dice roll. The mercenaries are added to one of the player's territories, and act as extra armies for that turn. These armies are on the same level as Infantry pieces, they cannot be moved, nor exchanged.
- **Revolution**: This card allows a player to incite a revolution in one of their opponent's territories. The opponent loses control of the territory, and the player gains control of it.
- **Draft**: This card allows you to draw two additional army cards at the end of your turn.
- **Bombardment**: You can use this card to attack a territory with a barrage of artillery, allowing you to roll two dice instead of one for that attack.
- **Reinforcements**: This card allows a player to add a certain number of armies to one of their territories. The number of armies added is determined by a roll of dice.

Because these cards vary in their mechanics and communication with game objects, each is handled by the system separately. UI handles them in different ways as well (as specified in the UI section).

For example, the “Mercenaries” card creates special armies with `isMercenary = true` attribute, as shown on the Domain Model Diagram.

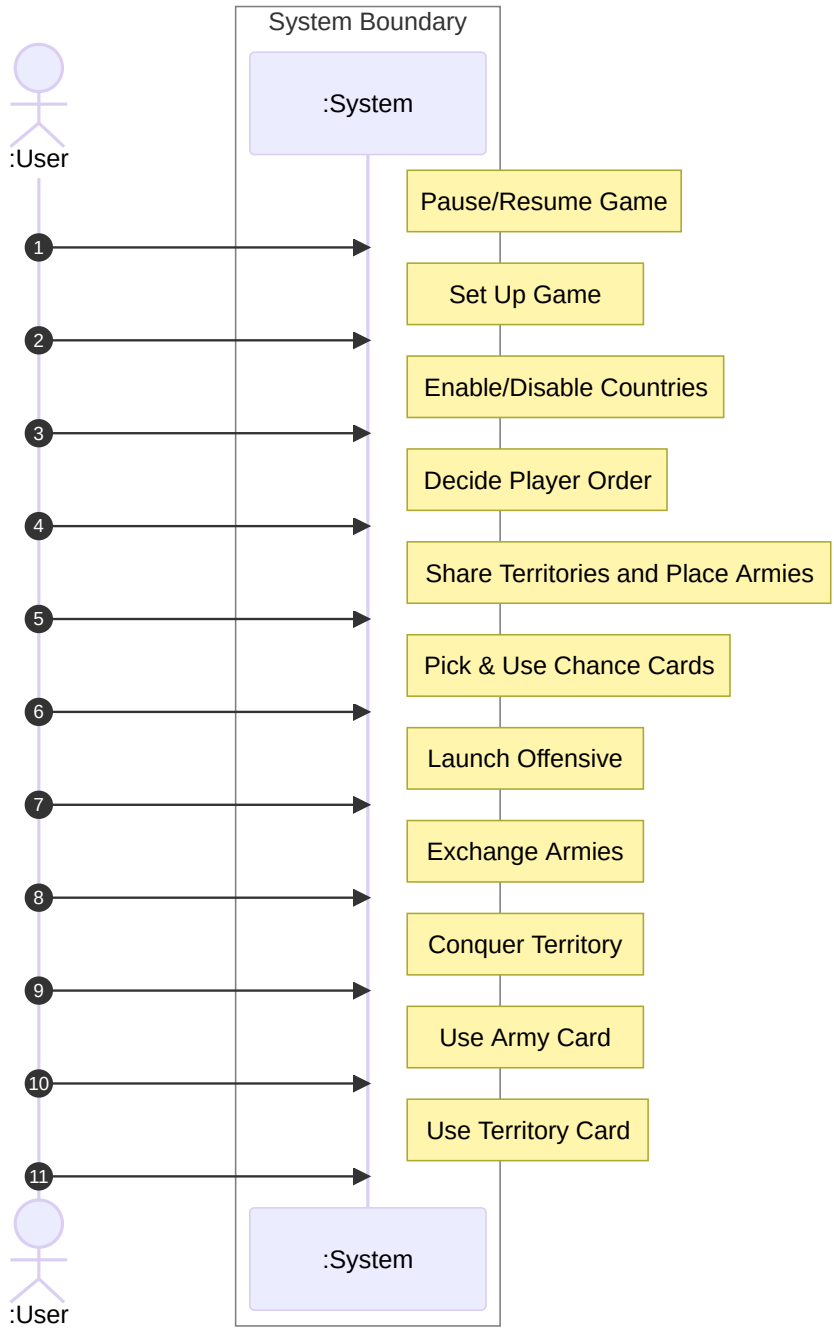
Domain: The Model

The following is a rough diagram of the necessary domain model objects. In the next phase, this will be replaced by a full and detailed class diagram.



User-System Interactions

Use Case Diagram



Use Cases

Share Territories and Place Armies Initally

Use Case Name: Share Territories and Place Armies Initally

Use Case ID: UC-1

Scope: ConKUeror-Phase 1

Level: User Goal

Primary Actor: System

Stakeholders and Interests:

Player: Wants to share equally territories and armies related to number of players.

Developer: Wants to protect equality and balance in the game by sharing armies correctly.

Preconditions:

- Game should have just started.

Postconditions:

- Each Player should have equal number of armies.
- All enabled Territories must be claimed by players.
- All Armies should be placed.

Main Success Scenario:

1. Each Player have replaced their armies on the board
2. The Game begins so now it is in running mode so Turn phases will start and it will be on the player who got highest number
3. The Player who got the highest number at the beginning get access to do an action and so the action starts.

Extensions:

***a.** Inadequate number of available territories:

1.System shows an error message stating that there are insufficient territories for the given number of players.

2. Players modify the quantity of territories or players to fulfill the

1a. Unfinished army deployment:

1. System identifies that a player has not positioned all their armies.
2. System asks the player to position the outstanding armies before advancing to the subsequent turn.

2a.Incorrect territory selection:

1. Player tries to position an army on an invalid or already claimed territory.
2. System exhibits an error message and instructs the player to pick a valid territory.

Technology and Data Variations List:

- Player uses mouse

Frequency of Occurrence:

- In every start phase of the game. So this will be per game once.

Pause/Resume Game

Use Case Name: Pause/Resume Game

Use Case ID: UC-2

Scope: ConKUeror-Phase 1

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:

Player: Should be able to Pause (stop) and Resume (continue) the game anytime.

Preconditions:

- Game must be open
- Game must be already set up.

Postconditions:

- Game stops or resume

Main Success Scenario:

1. Game succesfully stops and the player who has the turn wait the resume response.
2. Game succesfully resumes and the player who had the turn continues to play.

Extensions:

***a.** The game application crashes or unexpectedly closes:

1. Notify player(s) that the game state cannot be restored due to no save functionality upon relaunch.

1**a.** Player accidentally presses Pause button while the game is already paused:

1. Ignore additional pause requests.

2a. Player attempts to resume while another player is paused:

1. Notify player they cannot resume until the player who paused the game resumes.

Frequency of Occurrence:

- In every phase of the game players should be able to game and also resume it.

Game Setup

Use Case Name: Game Setup

Use Case ID: UC-3

Scope: ConKUeror-Phase 1

Level: User-goal

Primary Actor: System

Stakeholders and Interests:

Player: Should be able determine their name and number of total players, and also chose a color.

Developer: Should associate the color with the certain player and also present available player counts.

Preconditions:

- Game should have just started.

Postconditions:

- Every player has a unique color and name attached with their army.
- Number of player should be fixed

Main Success Scenario:

1. Game should show appropriate conditions like max and min number of players and also shows possible colors which is not already taken.
2. Game succesfully resume and the player who had the turn continues to play.

Extensions:

*__a__. Invalid input for player name, color, or count:

1. Display an error message and prompt the user to correct the input.
2. Do not allow the game to start until valid inputs are provided.

*__b.__ A player attempts to choose a color that is already taken:

1. Notify the player that the color is unavailable.
2. Prompt the player to select a different color.

*__c.__ The maximum number of players has been reached:

1. Notify the player that no more players can be added.
2. Disable the option to add more players.
3. Also may set the count to a fix number

Special Requirements:

- Players should be able to easily navigate and understand the setup options.
- The game should store player preferences, such as names and colors, for future sessions.
- Tutorial for help button occurs which shows the detail information about the game in a label when pressed.

Frequency of Occurrence:

- The game should have launched succesfully and it should per game and at the start of each game.

Enable/Disable Mode

Use Case Name: Enable/Disable Mode

Use Case ID: UC-4

Scope: ConKUeror-Phase 1

Level: User-goal

Primary Actor: System

Stakeholders and Interests:

Player: Should be able to disable and enable some territories/continents according to their preferences.

Developer: Should determine the limitations and implement the feature to disable and enable territories.

Preconditions:

- Game should be in building mode.

Postconditions:

- The disabled territories will no longer be available for play.
- The disabled territories will not be distributed by the game.

Main Success Scenario:

1. Player requests to enable or disable certain territories.
2. System checks for any limitations or restrictions.
3. System enables or disables the requested territories accordingly.
4. Players continue to play with the updated territory settings.

Extensions:

1-2a. Player requests to disable too many territories or violate any predefined limitations:

    1. System displays an error message notifying the player about the limitation.
    2. System does not apply the changes.

Frequency of Occurrence:

- Occurs when players want to customize the territories during the building mode.

Use Territory Card

Use Case Name: Use Territory Card

Use Case ID: UC-5

Scope: ConKUeror-Phase 1

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to use the territory card to gain bonus armies or special abilities during the game.

Developer: Wants to ensure that the territory card functions as intended and does not cause any issues in the game.

Preconditions:

- Player must have a territory card in their possession.
- Player must be in the reinforcement phase of their turn.

Postconditions:

- Player gains bonus armies or special abilities as specified by the territory card.
- The used territory card is removed from the player's possession.

Main Success Scenario:

1. Player selects the "Use Territory Card" option.
2. System displays a list of available territory cards.
3. Player selects the desired territory card to use.
4. System applies the bonus armies or special abilities as specified by the territory card.
5. The used territory card is removed from the player's possession.

Extensions:

3a. The selected territory card is not valid or has already been used:

1. System displays an error message notifying the player that the card cannot be used.

3b. The player does not have any territory cards:

1. System displays an error message notifying the player that there are no available cards.

4a. The reinforcement phase has ended:

1. System displays an error message notifying the player that the card can only be used during the reinforcement phase.

Frequency of Occurrence:

- Occurs when a player has a territory card and wants to use it during the reinforcement phase of their turn.
- The frequency depends on how often players obtain territory cards during the game.

Exchange Armies

Use Case Name: Exchange Armies

Use Case ID: UC-6

Scope: ConKUeror-Phase 1

Level: User-Goal

Primary Actor: Player

Stakeholders and Interests:

Player: Should be able to change their armies' quantities and qualities for given strategic situation.

Preconditions:

- Game must be on.
- User must have enough armies to do the exchange.

Postconditions:

- User successfully exchanges the armies soldier distribution.

Main Success Scenario:

1. Player decides to change their existing troops.
2. Player either increases its army size sacrificing a superior army troop or player sacrifices army size and changes less significant peace to superior piece.

Extensions:

*__a__. Player cancels the exchange before confirmation.

1. The system cancels the exchange, and the troops remain the same.

2__a__. Player does not have enough troops to exchange.

1. Request is ignored, and troops do not change.

Frequency of Occurrence:

- Every time the user who has to turn wants to exchange the troops in order to distribute its armies or launch an offense.

Decide Player Order

Use Case Name: Decide Player Order

Use Case ID: UC-7

Scope: ConKUeror-Phase 1

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:

Player: All players roll a die the user who gets the highest number starts the round. Once the first player is decided loop continues until player order is determined.

Developer: Wants to ensure that the die roll functions as intended and does not cause any issues in the game.

Preconditions:

- Game must be initialized.
- Number of players must be valid and determined.

Postconditions:

- Player order is initialized.

Main Success Scenario:

1. Each player rolls a die.
2. The player with the highest roll goes first, followed by the second highest, and so on.
3. If more than one player gets the same highest value, they roll the die again until one player gets a greater value.
4. The order of play is established and communicated to all players.

Extensions:

***a**. Player disputes the result of the die roll:

    1. The players discuss the issue and decide to either reroll the dice or come to a mutual agreement.

***b**. Technical issue occurs during the die roll:

    1. The system prompts the players to roll the die again.

4a. Player leaves the game after determining the order:

    1. The system adjusts the order of play accordingly.

Frequency of Occurrence:

- At the beginning of every game
- After game setup is completed.

Launch Offensive

Use Case Name: Launch Offensive

Use Case ID: UC-8

Scope: ConKUeror-Phase 1

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to gain additional land to win the game and crush their foes.

Developer: Wants to ensure that the offence functions as intended and does not cause any issues in the game.

Preconditions:

- Game must be started.
- Offensive and defensive player should be neighbors in the map.
- Offensive player must have armies.
- Armies should be in the border territory between offensive and defensive player.
- Total number of armies on that border territory must be greater in the offensive side at least by one.

Postconditions:

- Offensive player gains the territory if the attack is successful.
- Defensive player loses the territory if the attack is successful.
- The number of armies in the attacking and defending territories is updated based on the outcome of the attack.

Main Success Scenario:

1. Attacking player chooses the territory from which to launch the assault.
2. Attacking player picks the adjacent territory to target.
3. The system verifies if the attacking territory has a minimum of 2 armies and if the territories are directly connected.
4. The attacking player specifies the number of armies involved in the assault (maximum of 3 or the available armies - 1).
5. The defending player designates the number of armies for defense (maximum of 2 or the available armies).
6. The system rolls dice for both attacking and defending players, then compares the results.
7. The system calculates the battle outcome, modifies the number of armies in both territories, and alters the territory control if necessary.

Extensions:

1**a**. Attacking player selects an inappropriate territory for attack or defense:

    1. System shows an error message and requests the player to choose suitable territories.

2a. Attacking player lacks adequate armies for initiating an assault:

    1. System shows an error message and asks the player to choose a territory with enough armies.

Frequency of Occurence:

- Every time the player who has the turn that satisfies the conditions decides to launch an offence.

Conquer Territory

Use Case Name: Conquer Territory

Use Case ID: UC-9

Scope: ConKueror-Phase 1

Level: User Goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to add the territory to its countries lands.

Developer: Wants to meet that goal without any crashes occur.

Preconditions:

- Game must be on.
- User must have made an attack and be victorious.
- There shouldn't be any enemy armies left on the territory

Postconditions:

- The conquered territory gets added to the user.
- The former owner of the territory lost the territory

Main Success Scenario:

1. After a succesful attack user's troops enter the territory.
2. User obtains the territories card for the conquered territory.
3. The player who loses the territory loses armies on that specific territory.
4. Conqueror user choses how many troops to station on the conquered territory
5. Territories color changes to conqueror's color
6. User obtains the conquered territory card.

Extensions:

3**a**. Defense player has existing armies in the territory.

    1. Territory cannot be captured.

5a. Territory does not change colors after conquering occurs.

    1. Game crashes and must be restarted.

Frequency of Occurrence:

- Each time users makes a consecutive successful attack and demolishes enemy troops completely from a territory.

Use Army Card

Use Case Name: Use Army Card

Use Case ID: UC-10

Scope: ConKUeror-Phase 1

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:

Player: Wants to use the army card to gain advantage during the game.

Developer: Wants to ensure that the army card functions as intended and does not cause any issues in the game.

Preconditions:

- Player must have an army card in their possession.
- Player must be in a phase of their turn where the card can be used.

Postconditions:

- Player gains advantage or benefit as specified by the army card.
- The used army card is removed from the player's possession.

Main Success Scenario:

1. Player selects the "Use Army Card" option.
2. System displays a list of available army cards.
3. Player selects the desired army card to use.
4. System applies the advantage or benefit as specified by the army card.
5. The used army card is removed from the player's possession.

Extensions:

3a. The selected army card is not valid or has already been used:

    1. System displays an error message notifying the player that the card cannot be used.

3**b**. The player does not have any army cards:

    1. System displays an error message notifying the player that there are no available cards.

4a. The phase of the player's turn does not allow for the use of army cards:

    1. System displays an error message notifying the player that the card cannot be used in the current phase.

Frequency of Occurrence:

- Occurs when a player has an army card and wants to use it to gain advantage or benefit during their turn.
- The frequency depends on how often players obtain army cards during the game.

Pick & Use Chance Card

Use Case Name: Pick & Use Chance Card

Use Case ID: UC-11

Scope: ConKUeror-Phase 1

Level: User-goal

Primary Actor: Player

Stakeholders and Interests:

Player: Must pick and use a chance card at the beginning of their turn to gain advantage during the game.

Developer: Wants to ensure that the chance card functions as intended and does not cause any issues in the game.

Preconditions:

- There must be chance cards available in the game.
- Player must be in the first phase of their turn where chance cards can be used.
- Player must draw a chance card before being able to use it.

Postconditions:

- Player gains advantage or benefit as specified by the chance card.
- The used chance card is removed from the player's possession.

Main Success Scenario:

1. Player draws a chance card at the beginning of their turn.
2. System displays the drawn chance card.
3. Player selects the "Use Chance Card" option.
4. System applies the advantage or benefit as specified by the chance card.
5. The used chance card is removed from the player's possession.

Extensions:

*__a__. The phase of the player's turn does not allow for the use of chance cards:

    1. System displays an error message notifying the player that the card cannot be used in the current phase.

3a. The drawn chance card is not valid or has already been used:

    1. System displays an error message notifying the player that the card cannot be used.

3__b__. There are no chance cards available:

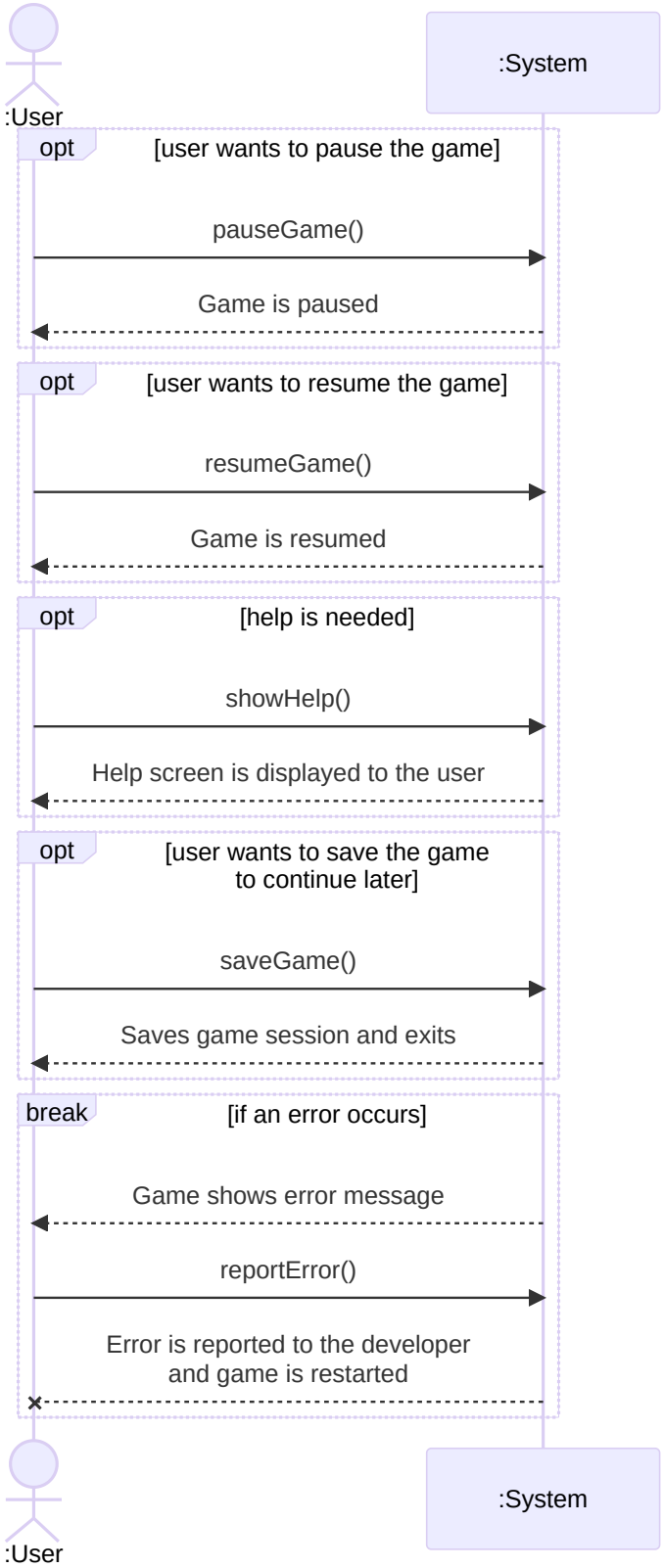
    1. System displays an error message notifying the player that there are no available cards.

Frequency of Occurrence:

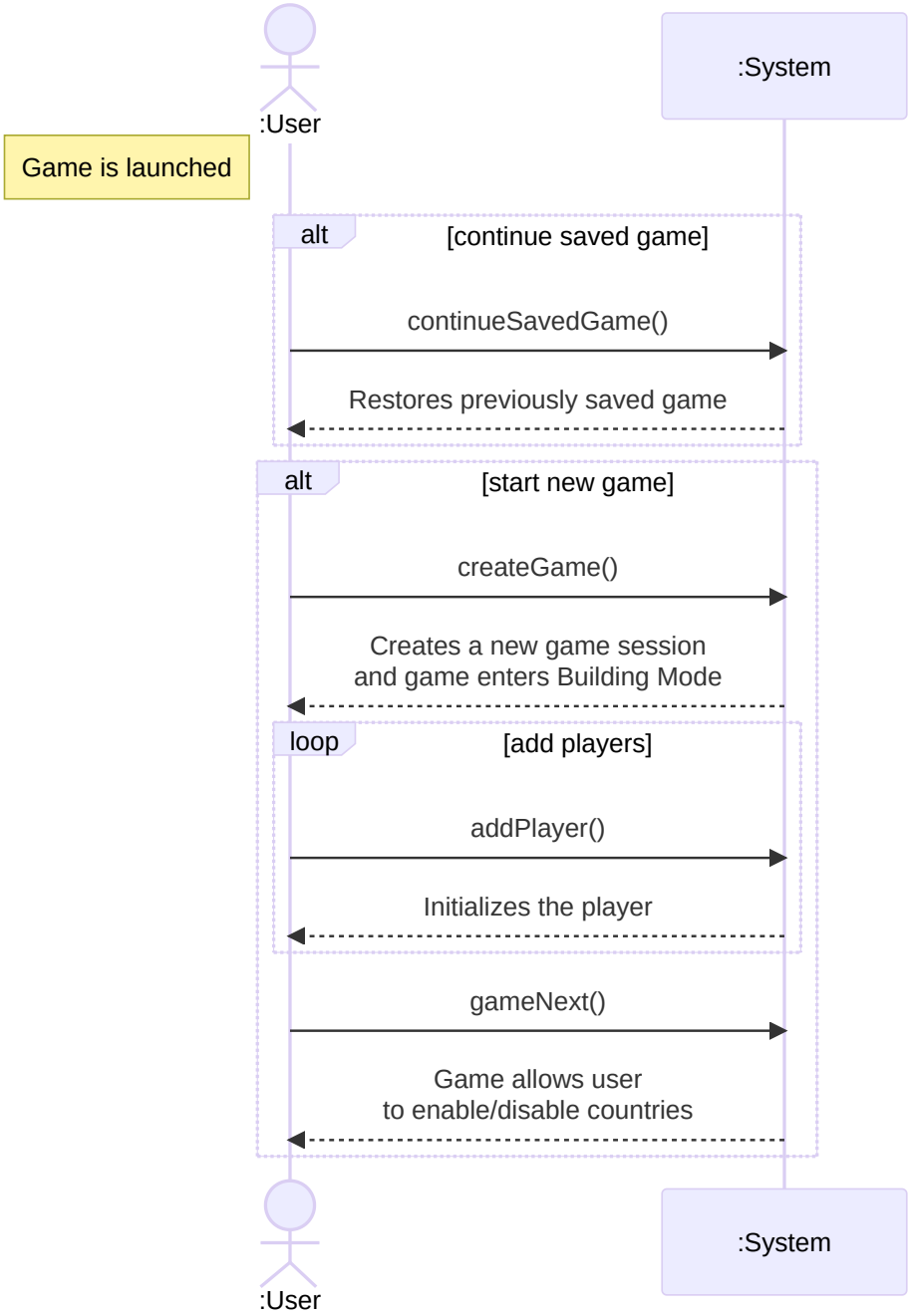
- Occurs at the beginning of each player's turn when they draw a chance card and must use it to gain advantage or benefit during the game.
- The frequency depends on how many players there are in the game and how many chance cards are available.

System Sequence Diagrams

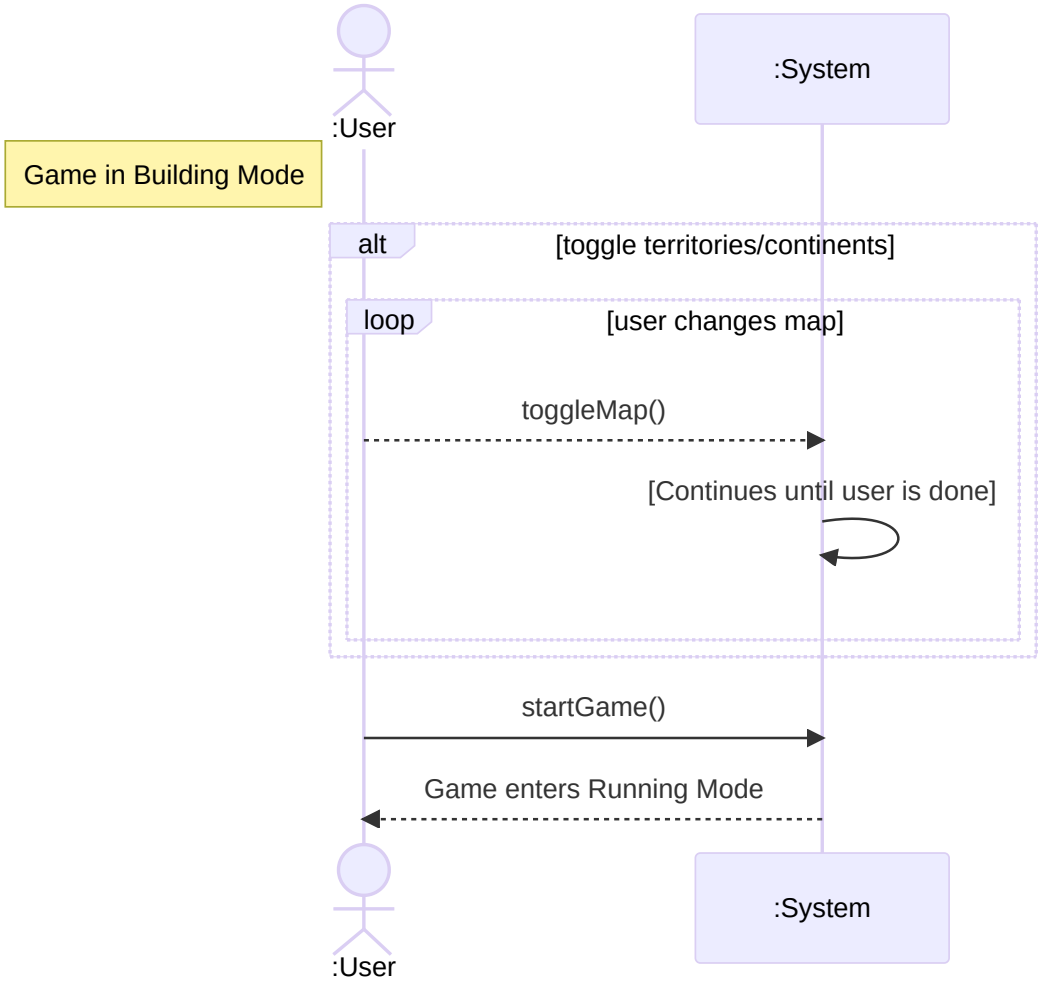
Pause/Resume Game


:User
:System
opt
[user wants to pause the game]
pauseGame()
Game is paused
opt
[user wants to resume the game]
resumeGame()
Game is resumed
opt
[help is needed]
showHelp()
Help screen is displayed to the user
opt
[user wants to save the game
to continue later]
saveGame()
Saves game session and exits
break
[if an error occurs]
Game shows error message
reportError()
Error is reported to the developer
and game is restarted
:User
:System

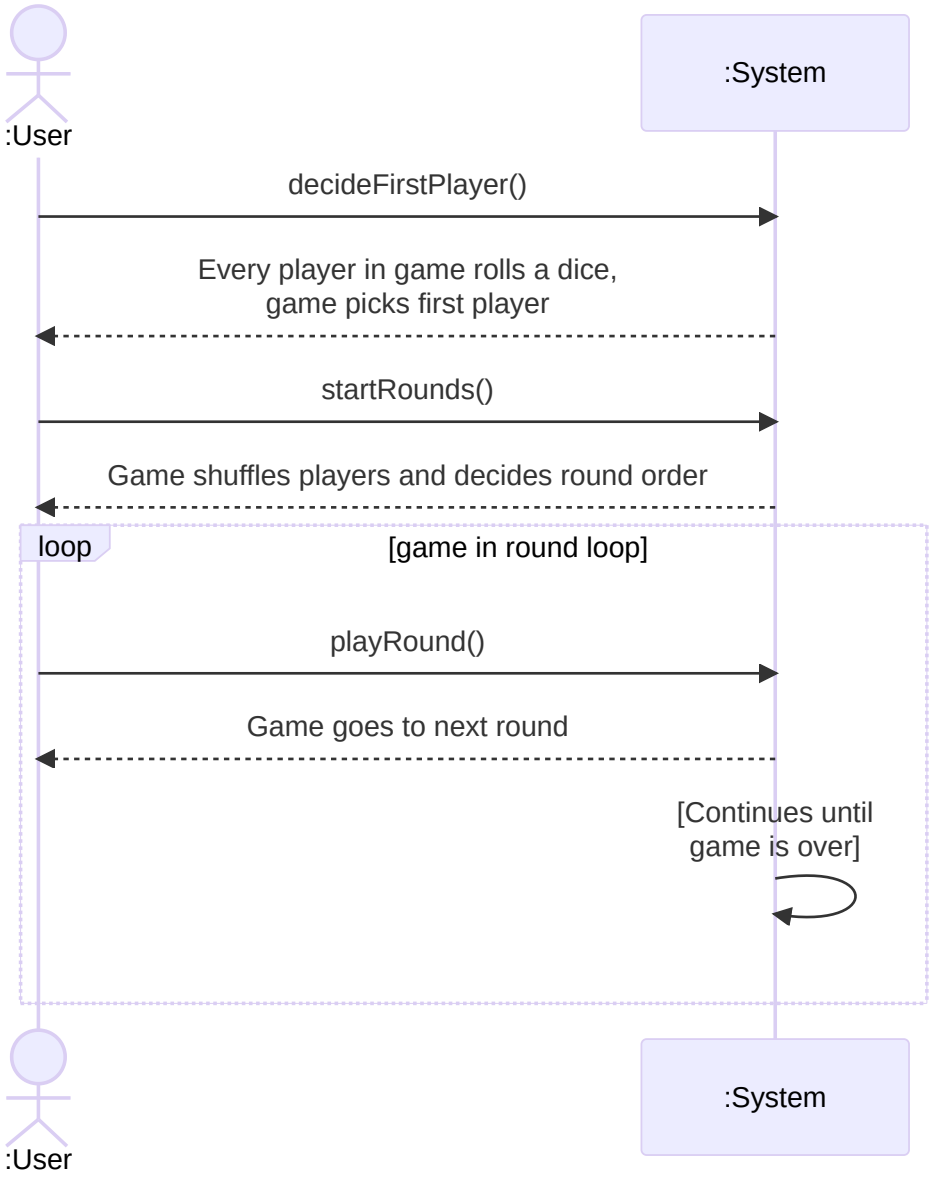
Setup Game



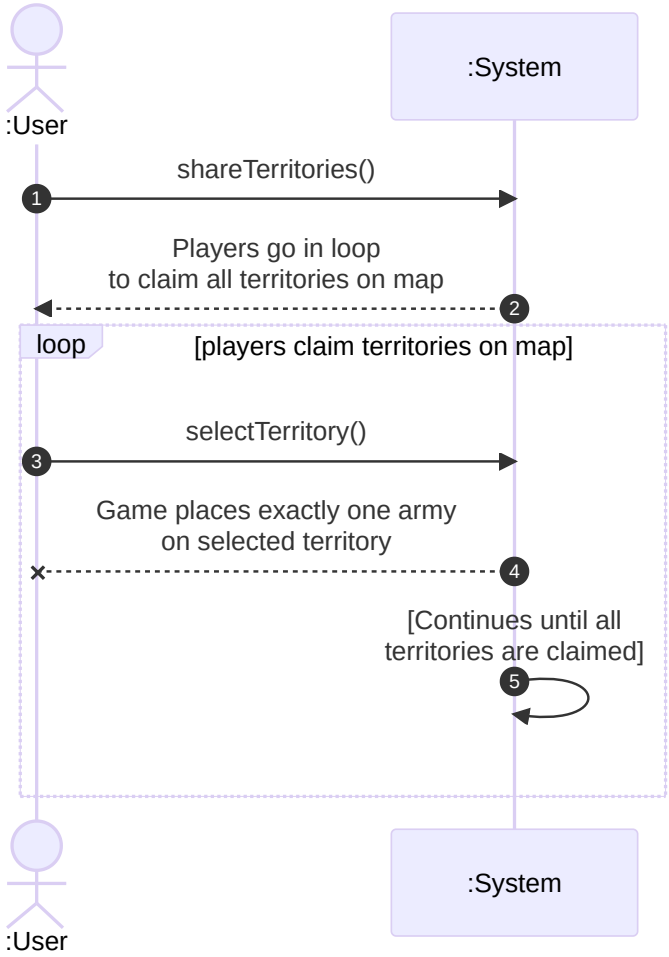
Enable/Disable Countries



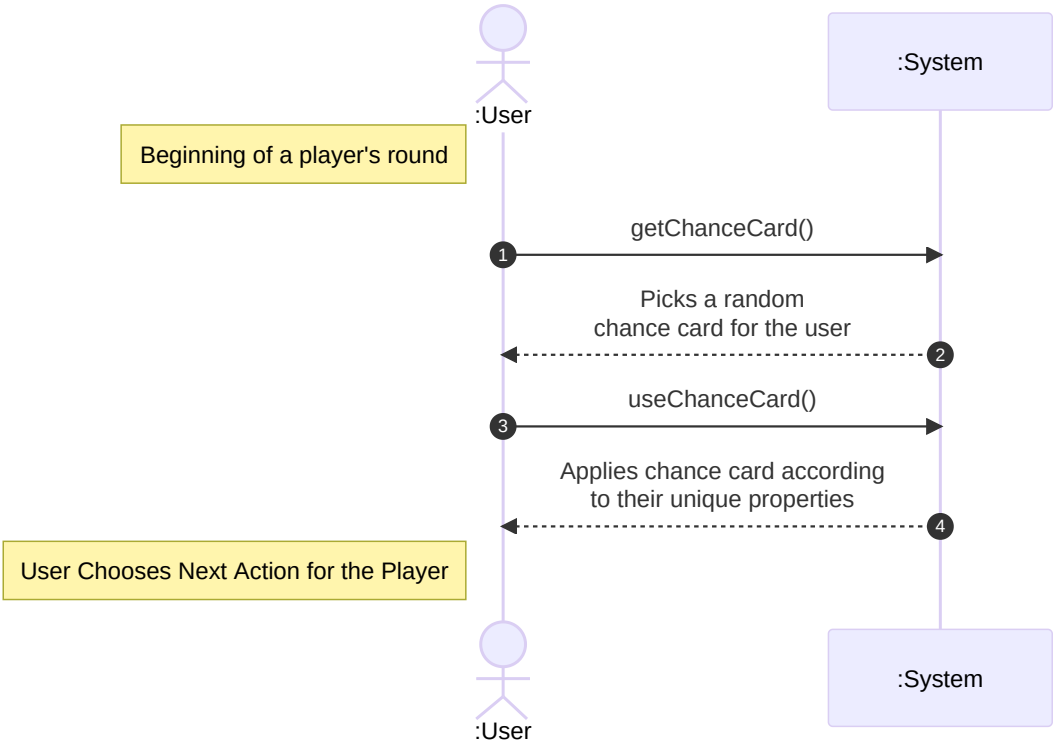
Decide Player Order



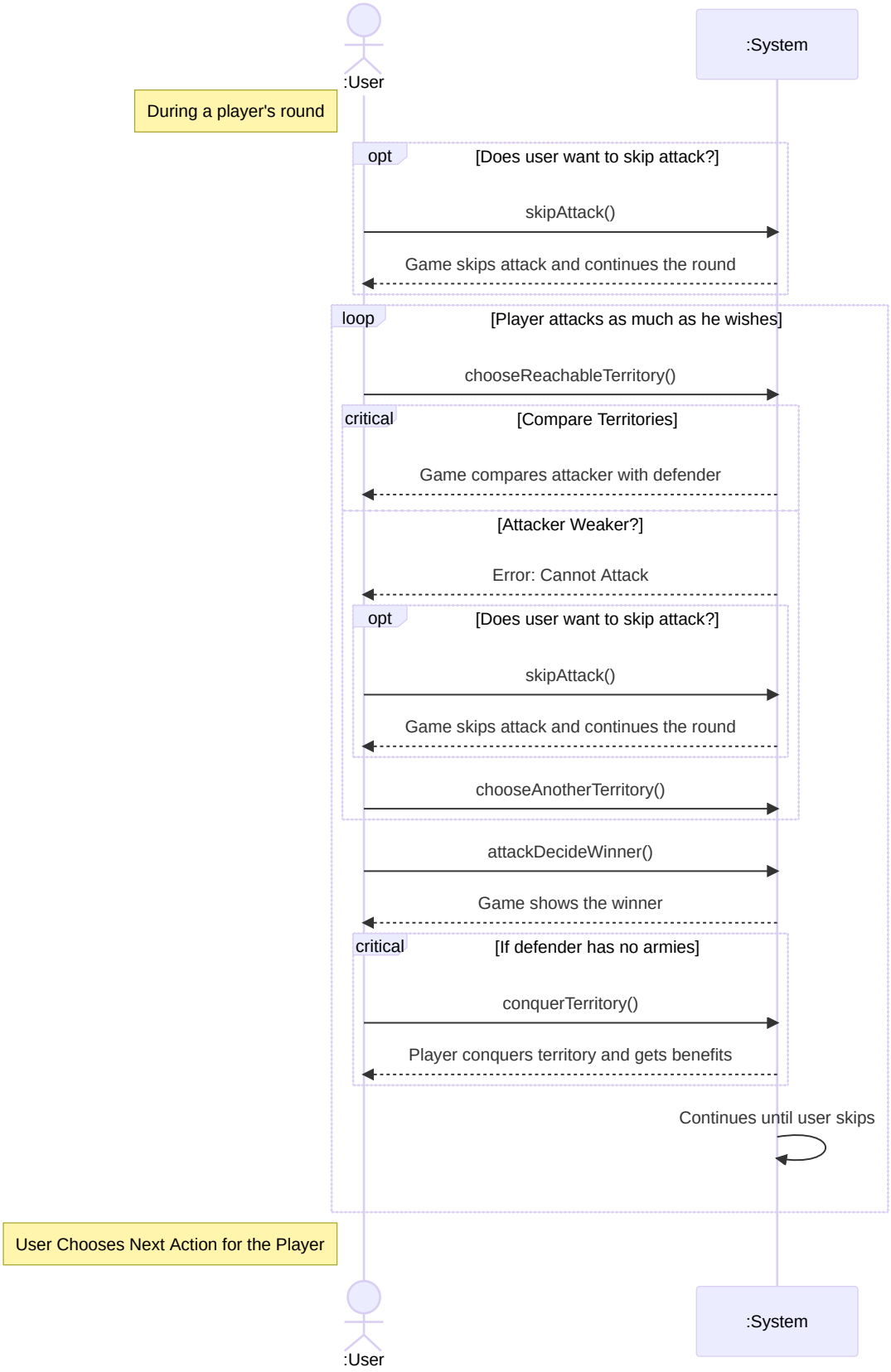
Share Territories and Place Armies



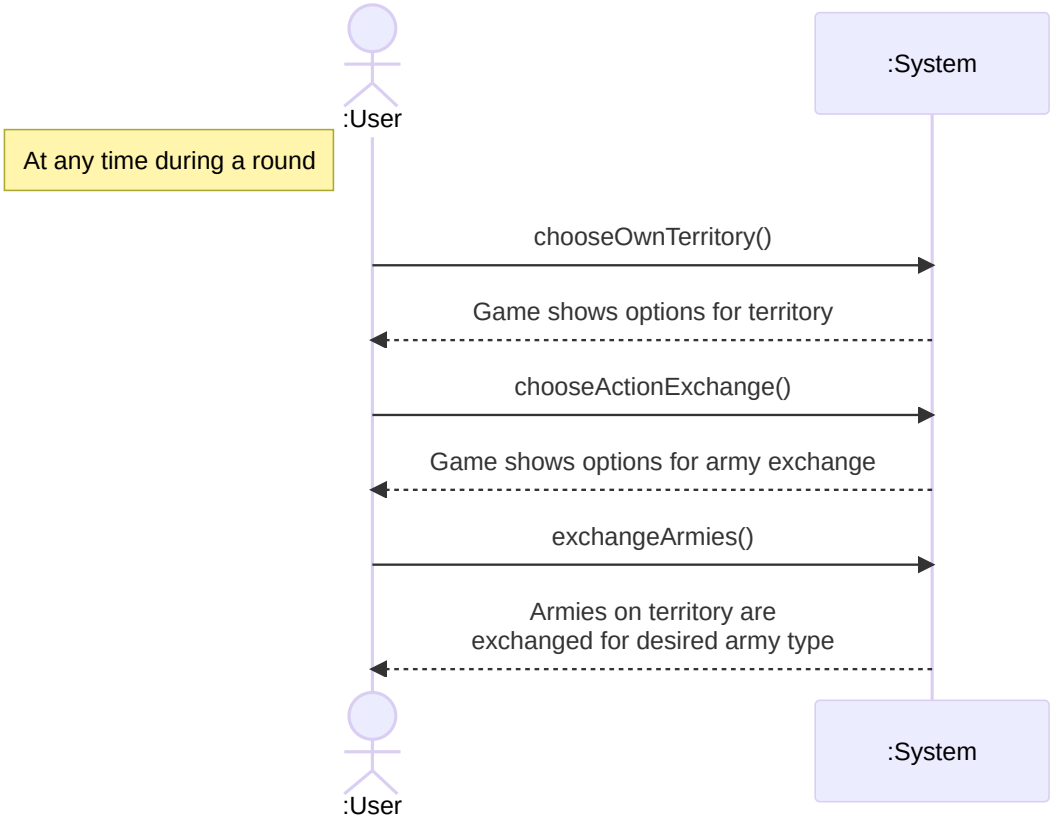
Pick & Use Chance Card



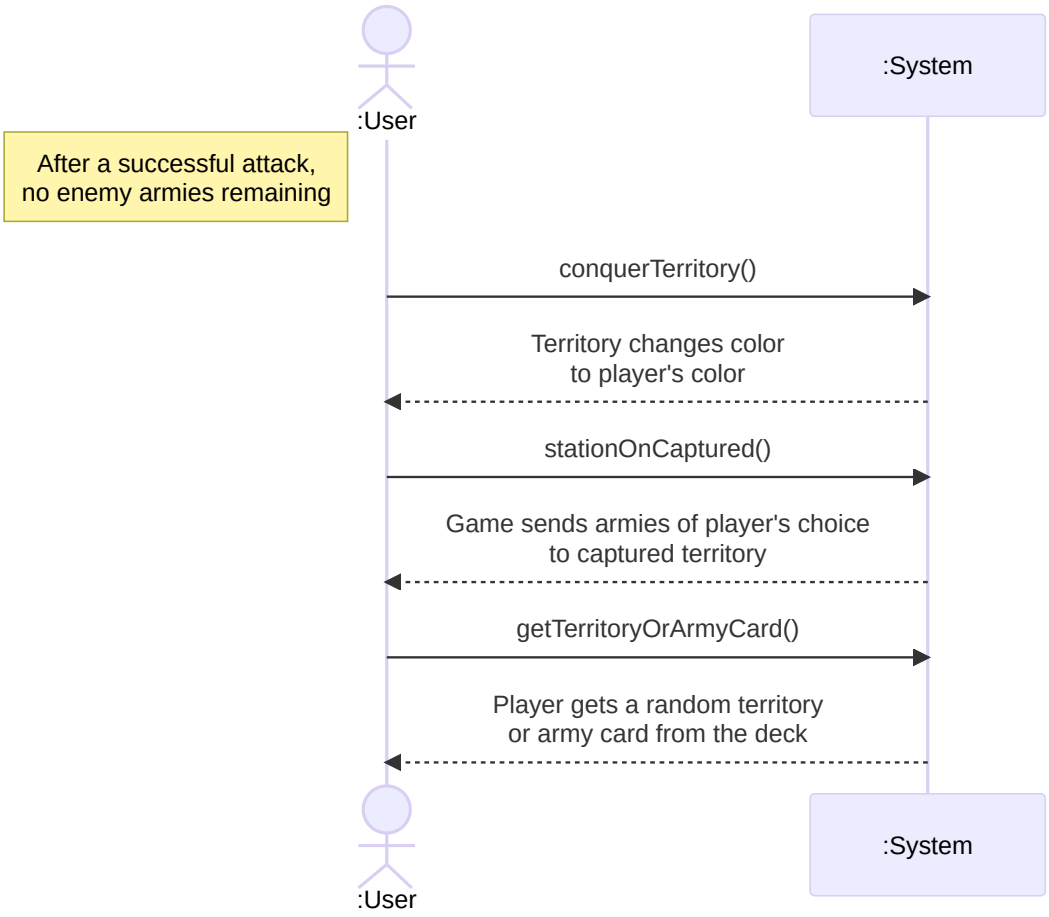
Launch Offensive



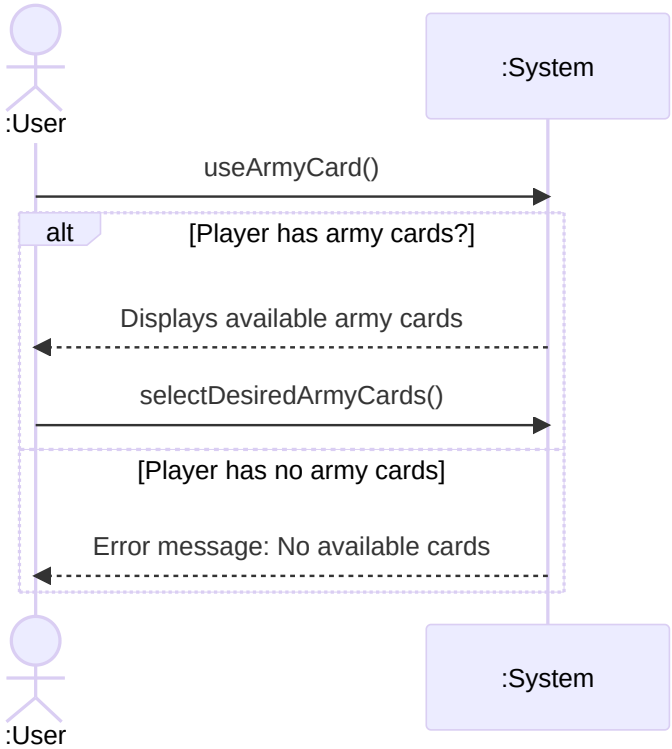
Exchange Armies



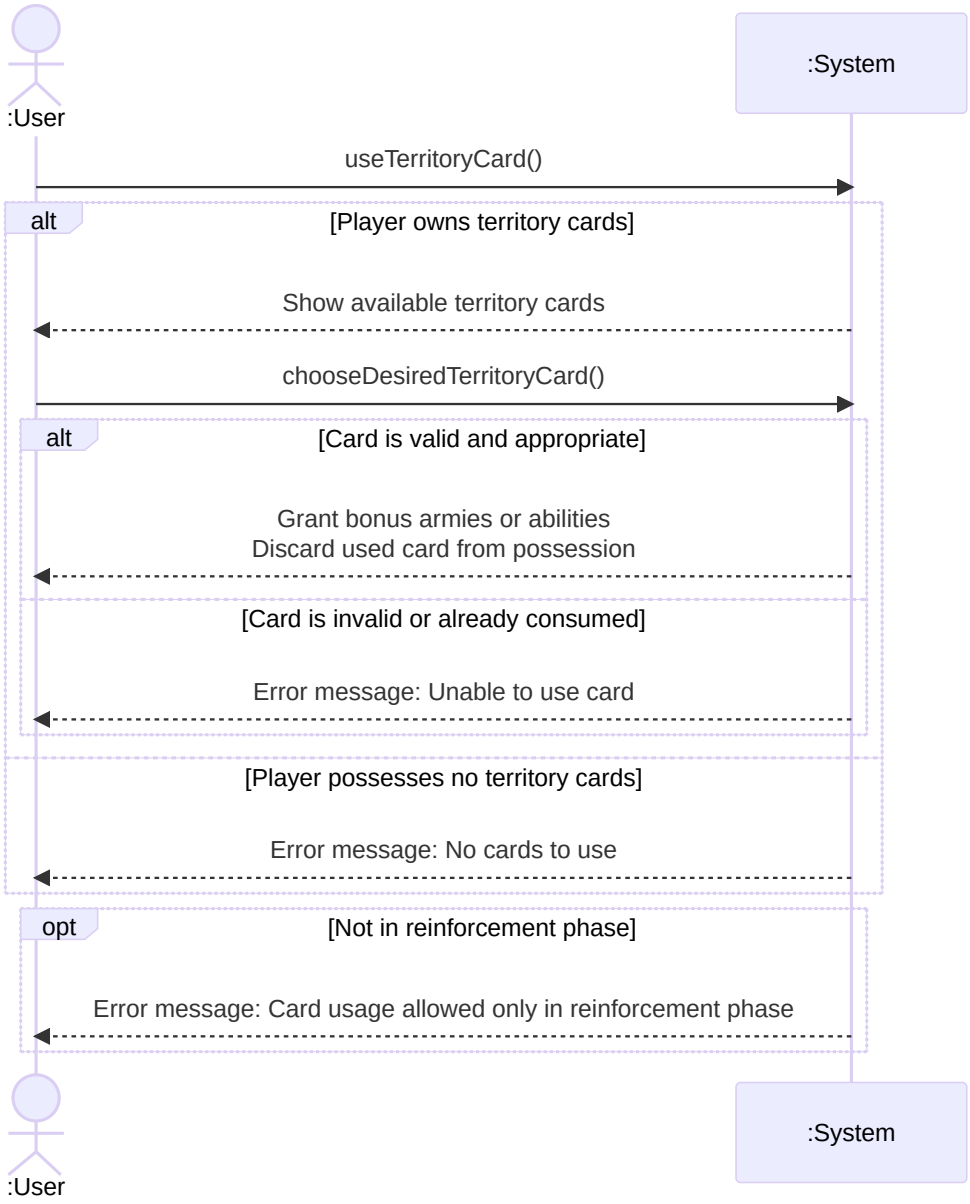
Conquer Territory



Use Army Card



Use Territory Card



Operation Contracts

Operations: *pauseGame()*
Cross-References: Use-Cases: Pause/Resume Game
Pre-Conditions:
- Game must be already set up.
- The previous playing game has been saved.

Post-Conditions:
- No instance was created.
- No association were formed
- continueSaved() method has been called.

Operations: *resumeGame()*
Cross-References: Use-Cases: Pause/Resume Game
Pre-Conditions:
-Game must be already set up.
Post-Conditions:
- A game instance has been created.
- Previous saved game was loaded.

Operations: *setPlayerCount()*
Cross-References: Use-Cases:(Set Game) Game Setup
Pre-Conditions:
- Game should have just started.
- playerList has been completed.
- Players has been added to list
Post-Conditions:
- Number of players should be fixed
- setPlayerCount(PlayerList playerList)
- A PlayerList instance was created as playerList (Instance creation)
- Number of players has been counted by an enhanced for loop. If the number of players was not valid, the playerNumberError instance was created as countError. (Instance creation)
- Count error was given to reportError(Error e) method.

Operations: *createGame()*
Cross-References: Use-Cases:(Set Game) Game Setup
Pre-Conditions:
- Game should not have started.
Post-Conditions:
- Screen displays game.
- createGame()
- A Game instance was created (Instance creation.)
- addPlayer() method was called.
- A playerList was taken from addPlayer() method.
- game.playerList became playerList from addPlayer(). (Attribute modification)
- game.history became current history. (Attribute modification)

Operations: saveGameSeesion(Game game**)**
Cross-References: Use-Cases: Pause/Resume Game, (Set Game)Game Setup
Pre-Conditions:
- Game is paused.

Post-Conditions:
- Current game progress is stored.
- A Game instance was created as game. (Instance creation)
- game was associated with current game instance(Association formed)
- game.History is assigned to history(Attribute modification)

Operations: *reportError(Error e)*
Cross-References: Use-Cases: All use cases
Pre-Conditions:
- Player tries invalid action.
- The corresponding error instance has been created and provided.
Post-Conditions:
- System displays an error message accordingly.
- An Error instance was created as an error. (instance creation) - No attribute modification.
- No association formed. - @tostrig() method of error was returned to the user.

Operations: showHelp((optional) str helpCode)
Cross-References: Use-Cases: All use cases
Pre-Conditions:
- Player tries invalid action.
- The corresponding error instance has been created and provided.
Post-Conditions:
- System displays help options.
- General information expression/attribute was created as genInfo.
- General information was assigned to genInfo. - genInfo printed/displayed.
- A specific request exp./attribute is created as specReq.
- A click or command was taken from the user and assigned to specReq. (This could be quit from help such as “q”)
- The specific request was displayed to the user.
- Process continue until the user quit from help.
- (optional) Additional or more specific requests accepted.

Operations: addPlayer(str name)
Cross-References: Use-Cases: (Set Game) Game Setup
Pre-Conditions:
- Game should have just started.
- Name of the player has been decided and provided.

Post-Conditions:
- A list of players was called a playerList.
- Player instance was created.(Instance Creation)
- Name is assşgned to Player instance. (Attribute modification)

- Player instance has been added to the playerList.
- A boolean or a message returned.(F.e: “Player has been added”)

Operations: rollAllPlayers()
Cross-References: Use-Cases: Decide player order
Pre-Conditions:
- A round should be just starting
Post-Conditions:
- Dice instances were created. (Instance Creation)
- Dice instances are associated with players (Association Formed)
- Dice instances are rolled (Attribute Modification)

Operations: continueSaved(Game game)
Cross-References: Use-Cases: Set Up Game
Pre-Conditions:
- Previous playing game was paused.
- Previous playing game has been saved.
- Game was launched again by the Player.
Post-Conditions:
- A Game instance was created as a game. (instance creation)
- game was associated with saved Game data. (Association formed)
- This game instance attributes were assigned the previous data(attribute modification)

Operations: gameNext()
Cross-References: Use-Cases: Set Up Game
Pre-Conditions:
- Game was launched
- Game instance was created
- Players has been added and initialized
Post-Conditions:
- Country instances has been created (instance creation)
- Country instances were associated with desired countries. (Association formed)
- Country names were became assigned country names(attribute modification)
- Country positions were became assigned country names(attribute modification)

Operations: toggleCountry()
Cross-References: Use-Cases: Enable/Disable Countries
Pre-Conditions:
- Game is in running mode
- A player has the turn

Post-Conditions:
- Clicked territory is marked as toggled (Association Formed)

Operations: startGame(Game game)
Cross-References: Use-Cases: Enable/Disable Countries
Pre-Conditions:
- a Game instance is created
- Game instance is not in running mode
Post-Conditions:
- Game mode is marked as running (Attribute modification)
- Game is in running mode

Operations: decideFirstPlayer(PlayerList playerList)
Cross-References: Use-Cases: Decide Player Order
Pre-Conditions:
- playerList is created
- A round should be just starting
Post-Conditions:
- rollAllPlayers() method have been called
- rolls associated with players get compared - player with the best roll gets first place (association created)
- other players are shuffled (association created)

Operations: startRounds()
Cross-References: Use-Cases: Decide Player Order
Pre-Conditions:
- first player is decided
Post-Conditions:
- first player's turn

Operations: playRound()
Cross-References: Use-Cases: Decide Player Order
Pre-Conditions:
- players' order is decided
Post-Conditions:
- every player played their turn

Operations: shareTerritories()
Cross-References: Use-Cases: Share Territories and Place Armies

Pre-Conditions:
- game just started- players' order is decided

Post-Conditions:
- selectTerritory() method is called for players in loop

Operations: selectTerritory()
Cross-References: Use-Cases: Share Territories and Place Armies
Pre-Conditions:
- territory is toggled and approved

Post-Conditions:
- territory's owner attribute is modified (Attribute modification)
- player's territories list is modified (Attribute modification)
- one Army instance is created (Instance Creation)
- exactly one army is appended to player's armies list (Attribute modification) - exactly one army is appended to player's armies list (Attribute modification)

Operations: chooseActionExchange(Army userArmy)

Cross-References: Use-Cases: Exchange Armies

Pre-Conditions:

- A round has been started

- Player has been ready for choosing the territory

Post-Conditions:

- Army instance army was created. (Instance creation)

- userArmy was associated with army instance.

- Based on the userArmy.quantity and exchangeRate attributes, appropriate exchange options were displayed to use

Operations: chooseOwnTerritory(Territory territory)

Cross-References: Use-Cases: Exchange Armies

Pre-Conditions:

- A round has been started

- Player has been ready for choosing the territory

Post-Conditions:

- A Territory instance has been created.(Instance created)

- Given territory instance has been associated with created territory. (Association formed)

- The territory's availability attribute has been controlled.(Attribute modification/checking)

- If the territory is available, the territory's availability has been changed to taken.(Attribute modification)

- Return a boolean assigned and conditions of exchange.

- If it is not available, an Error instance has been created as terriotoryError.(Instance creation)
- The error has been thrown to the user.

Operations:exchangeArmies(Army givenArmy, Army takenArmy)

Cross-References: Use-Cases: Exchange Armies

Pre-Conditions:

- Armies has been stayed with user.

- Proper armies should be sufficient for exchanging.
- Army object's instances has been created for users(There are many three types of armies)

Post-Conditions:

- Two Army instances have been created. (ınstance creation)
- Armies associated with given and taken armies. (Association formed)
- Exchange conditions of takenArmies are checked with givenArmies.
- If conditions are satisfied, change occurs. givenArmy.quantity and takenArmy.quantity has been changed according to exchange conditions. (Attribute modification)
- Boolean returned for a successful exchange.
- Otherwise, for not sufficient conditions, an Error instance exchangeError has been created, and thrown to the user. (Instance creation)

Operations: doAnotherExchange()

Cross-References: Use-Cases: Exchange Armies

Pre-Conditions:

- player chose Not Enough? option

Post-Conditions:

- exchange army phase is restarted via chooseDesiredArmy() method

Operations: stationOnCaptured()

Cross-References: Use-Cases: Conquer Territory

Pre-Conditions:

- a territory was just captured

Post-Conditions:

- player chooses number of armies to cross to conquered territory from attacker territory
- chosen armies are added to conquered territory's armies list (Attribute modification)
- chosen armies are removed from attacker territory's armies list (Attribute modification)
- getTerritoryOrArmyCard method is called

Operations: getTerritoryOrArmyCard()

Cross-References: Use-Cases: Conquer Territory

Pre-Conditions:

- armies were stationed to conquered territory

Post-Conditions:

- conqueror gets a random territory or army card from the deck (Association Formed)

- territory cards are added to player’s territoryCards list (Attribute Modification)

- army cards are added to player’s armyCards list (Attribute Modification)