nerd^5

ConKUeror: D1

“When to use iterative development? You should use iterative development only on projects that you want to succeed.” – Martin Fowler

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.

https://github.com/altunh/2023S_nerd5

Team Members

- **Altun Hasanli** (Team Leader) (ahasanli19@ku.edu.tr)
  🏢 *Department of Sarcasm*
- **Ömer Nadir Civelek** (ocivelek19@ku.edu.tr)
  ⚡ *Department of Devil’s Advocacy*
- **Mert Balçık** (mbalcik20@ku.edu.tr)
  💡 *Department of Ideation*
- **Kazım Toprak Güler** (kguler20@ku.edu.tr)
  🎨 *Department of Media & Visuals*
- **Zindan Kurt** (zkurt18@ku.edu.tr)
  💬 *Department of Punctuality*

Contents

⚔️ Presentation

🎮 Specification

⚙️ Design

➗ Logical Architecture

↗️ Diagrams

⚔️

Presentation

ConKUeror: Koç University Dominion

Competitive multiplayer strategy game inspired (i.e. stolen) from RISK.

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!

ConKUeror is played on a political map of the world. The map is divided into territories and grouped into continents. Players control armies of playing pieces and try to capture territories from other players. The goal of the game is to occupy every territory on the map and eliminate the other players.
So why not gather your friends and start playing today?

Players battling for dominance — *generated by AI.*

- **Build an army to clash against your foes!** You'll get to recruit and train soldiers from various parts of the world, each with their own unique strengths and weaknesses. It's up to you to build the perfect army that can conquer any challenge that comes your way.
- **Use diplomacy to gain allies and fight to the death for blood and honor!** You'll have to use your wit and charm to forge alliances with other players, but make no mistake, when it comes down to it, it's every player for themselves. You'll have to be ruthless and fight to the death for your honor and the glory of your kingdom.
- **Command your troops on the battlefield!** You'll be the one giving the orders on the battlefield, directing troops to attack, defend, or retreat. It's up to you to make quick and strategic decisions that will turn the tide of battle in your favor.
- **Engage in glorious combat and all-out war!** Get ready for epic battles that will test your skills and your nerves. You'll have to outsmart your opponents, use cunning strategies, and make bold moves if you want to emerge victorious.
- **Protect your allies & conquer your enemies!** The world of ConKUeror is a dangerous place, and you'll need to do everything in your power to protect your allies and defeat your enemies. But be warned, betrayal is always a possibility, and you'll need to stay on your toes if you want to stay one step ahead of your opponents.
- **Use strategy to lead your army to victory!** ConKUeror is a game of strategy, and you'll need to use all your cunning and tactical skills to lead your army to victory. You'll have to outthink, outmaneuver, and outplay your opponents if you want to come out on top.
- **Play with friends!** ConKUeror is a game that's meant to be played with friends. Gather your buddies, form alliances, and take on the world together. Who knows, you might just conquer it all!

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

Features

- **ConKUeror** is designed to be played on a single device by 2-6 players, much like a board game.
- **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 9 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

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

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.

Enabling/Disabling Territories & Continents —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 and no player can station troops on them.

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.

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

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.

“*veni, vidi, vici”, roman saying — I came, I saw, I conquered*

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.

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.

Vision

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

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.

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

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.

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

*"Software is like s**. It's better when it's free."*
— Linus T., creator of Linux

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.

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.

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.

ConKUeror Glossary

- **ConKUeror: Koç University Dominion** — 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.
- **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 map 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).
- **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 map. 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.

🎮

Specification

ℹ️ The following is a full Phase 1 specification — design and implementation requirements for the game **ConKUeror.** Some parts of the specification might be subject to change during the next stage/phase of development.

User Interface: The View

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

*Login screen of **ConKUeror.***

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.
- **Building screen**: Lets the user to add players and configure the map.
- **Game screen**: The screen with the world map and the user panel.
- **Pause screen**: Pauses the game, shows a leaderboard of the players in the game with their army and territory counts. Also has options to go to the help screen, save and 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.

Game screen of **ConKUeror: Koç University Dominion.** Shows the map, menu options, the phase panel with user actions and a list of all players.

Map Grid

The Map consists of **6 continents** and **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

Asia (7)

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

North America (9)

- Alaska
- Greenland
- Northwest
- Alberta
- Ontario
- Quebec
- Western America

- Anatolia
- Persia

Africa (6)

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

- Eastern America
- Central America

Oceania (4)

- Indonesia
- New Guinea
- Western Australia
- Eastern Australia

South America (4)

- Venezuela
- Peru
- 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.

Pause screen for **ConKUeror.**

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 `App` — or as we call it, the **System** — is the controller of the MVC pattern. In our diagrams, we actively use the name, **System** 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

Screen for adding players in **ConKUeror**.

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, Green, Yellow, Orange, Purple. There are also 30 different avatars to choose from.*

Screen for customizing the player in **ConKUeror.**

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

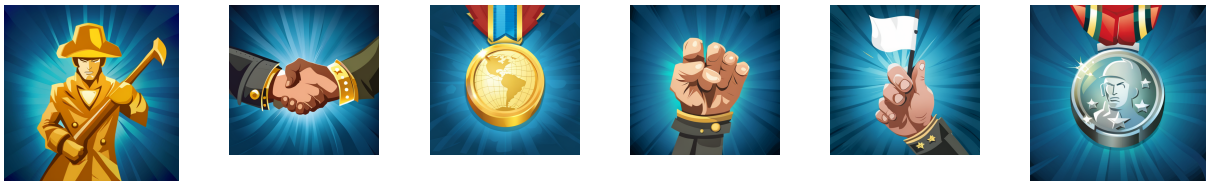
At any point during their round, if players have a complete set of 3 army cards, they can exchange them using the following rules:

- **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 Cards



Some of the icons for different chance cards in the game.

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.

Design

Design Principles

Ease of Use

The player needs an easy-to-use UI and intuitive controls to interact with game objects. Keep in mind that **ConKUeror** is a computer game based on the board game RISK and, by design, is a strictly game-controlled environment. Because the game has many moving parts, a simple help page is not sufficient for users to understand it. Therefore, intentional use of attention-grabbing icons is necessary 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 applicable and relevant. This is a principle that we actively utilize in designing the game.

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

Adaptability

Creating a captivating game environment requires effective system messages and announcements. For instance, when the game presents a chance card, the screen blurs out to focus the player's attention on the card's content. The message also includes big, relevant buttons for that chance card and a "skip" option. The panel always displays relevant information based on the current game setting and phase. During a player's turn, the panel shows easy-to-see buttons to exchange armies or use army and territory cards, but only if the player satisfies the requirements for doing so. This ensures that the user sees only the options they can use at that moment, rather than all of the game options at once.

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

Configurability

The game can adapt to different numbers of players and user preferences. The game map can be modified according to the players' needs. Additionally, depending on the number of players, ConKUeror is designed to utilize different rule sets to provide a fair experience for all players.

Reliability

Like any other application, ConKUeror is not immune to bugs and errors. If an error occurs, the system should display a message on the screen that includes the error, as well as a button to send the error to the developer. Additionally, the game should be saved and restarted to prevent the loss of user data.

To ensure security and reliability, the game should be built using one of the most popular and advanced programming languages in the development world: Java. This makes it highly reliable and secure, allows for system and

architecture-independent game development, and eliminates unexpected behavior.

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

Performance

As mentioned, the game should be built using one of the most potent and high-performing languages available, utilizing a fast and flexible core library to make the game run smoothly even on low-end machines. Performance is also a priority for the user interface and animations. As a 2D board game, ConKUeror utilizes the reliable Java library Swing to implement all custom UI objects, animations, and user interactions, minimizing the risk of unnecessary resource usage.

Cross-Platform Design

Java's cross-platform capabilities, made possible by its support for JVM, allow it to run on any operating system. This is particularly helpful, as it would enable the development of a network gaming option in the future. Players with different machines and operating systems could play together and enjoy the same gaming experience.

Modularity

The game should be packaged in a way that makes it easy to extend and customize. This would allow for easy additions of further chance cards and army types, new countries and continents, or even different game maps and themes.

Constraints & Limitations

ConKUeror uses only Java standard libraries and Swing. The custom UI components are also designed using these same libraries.

Implementation Principles / Patterns

Patterns used

- **Object-Oriented Programming:** Inheritance & Polymorphism
- **Model-View Separation:** MVC Pattern
- **Gang of Four (GoF):** Singleton, Simple Factory, Observer, Strategy
- **GRASP:** Creator, Expert, Controller, Low Coupling, High Cohesion

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`, 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 controller acts as the intermediary between the model and the view. App 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.

Controller

The controller pattern is actively used in **ConKUeror** to manage the application and game state. Both the `App` and the `Game` classes act as an intermediary between the model and the view, responsible for managing the game flow and providing the necessary data to the UI components. The `Game` class actively utilizes the **Observer Pattern** to ensure that the UI is updated whenever the game state changes.

The `Game` class also manages the various game components, such as the game deck, the game map, and the player objects. It ensures that these components are updated and managed correctly, and that the game logic is executed correctly.

Expert Pattern

The expert pattern is a design pattern used in software engineering that assigns responsibilities to the objects that have the necessary information to fulfill them. In **ConKUeror**, the controller and the specific `Game` instance uses the expert pattern to delegate responsibilities to the game objects that have the necessary information to perform specific tasks.

For example, when a player attacks an enemy territory, the game needs to determine the outcome of an attack and update the state. Instead of having the `Game` class handle all of these factors, the responsibility is instead delegated to the `Round` class, which in turn uses the `Dice` and `Deck`, which have all of the necessary information to determine the outcome of the attack.

Singleton

Singleton is a design pattern that ensures only one instance of a class can be created and provides a global point of access to that instance. This pattern is actively used in **ConKUeror** to ensure that there is only one instance of a class that manages the game state and flow, also allowing this class to be accessed by all other classes that require this information.

This is especially useful since it ensures a single instance of a class, which can be useful for managing global resources and state, and provides a global point of access to that instance, which simplifies the code and makes it easier to manage dependencies.

However, using the Singleton pattern alone is not thread-safe, which means that if multiple threads try to access the Singleton instance simultaneously, it can lead to race conditions and other issues. To address this, **ConKUeror** uses the **Bill Pugh Singleton pattern**, also known as the initialization-on-demand holder idiom, which is thread-safe and ensures that only one instance of a class is created even in a **multi-threaded environment**.

Simple Factory

The **Simple Factory pattern** is used to create the game map in **ConKUeror**. For example, the `DefaultWorldMap` class uses `createMapItem()` in reference to its adjacency matrix to create the different territories and continents for the map.

Creator Pattern

The Creator Pattern is 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. In **ConKUeror**, this pattern is used to create army objects. For example, the `ArmyCreator` interface defines a method called `createArmy()` that returns an `Army` object, and the `InfantryCreator` and `CavalryCreator` classes implement this interface to create infantry and cavalry armies, respectively. This allows for a more flexible and modular design, as new army types can be easily added by creating new classes that implement the `ArmyCreator` interface.

For example, chance cards utilize this pattern to add a new type of armies called the `Mercenary` using the `MercenaryCreator`.

Observer Pattern

The Observer pattern is 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. In **ConKUeror**, the Observer pattern is used to ensure that the game UI is updated to reflect changes in the game state.

For example, when a player makes a move, the game state changes, and the UI needs to be updated to reflect this change. By using the **Observer pattern**, the game is designed so that the UI is automatically updated whenever the game state changes. The main advantage of the Observer pattern is that **it decouples the subject** (the object being observed) from its observers. This means that changes to the subject do not affect the observers directly, but instead, the observers are notified of the changes and can update themselves accordingly.

However, a large issue with the Observer pattern is that it can lead to a large number of updates being sent to observers, which can be inefficient and lead to performance problems. Additionally, if the subject and observers are tightly coupled, it can be difficult to maintain and extend the system.

To address these, **ConKUeror** sends **updates locally and on-demand**, and utilizes **Low Coupling** and **Strategy.** For example, when a player moves an army from one territory to another, the game only sends an update to the UI for the affected territory, rather than updating the entire game map. Additionally, **ConKUeror** uses the Observer pattern in combination with other design patterns to create a more efficient and modular system. For example, the game uses the **Strategy Pattern** to determine how to handle different types of game events. This allows for more efficient and targeted updates to the game state, reducing the number of updates that need to be sent to observers.

Low Coupling

ConKUeror defines clear interfaces and uses dependency injection to provide the necessary objects to each class, actively utilizing the **Low Coupling pattern**.

For example, the `Game` class defines an interface for the `WorldMap` object, which is implemented by the `DefaultGameMap` class. The `Game` class is then instantiated with an instance of the `GameMap` interface, without knowing which concrete implementation is being used. This ensures that we can at some point add a new game map to the game for variety.

Similarly, the `Player` class defines an interface for the `Army` object, which is implemented by the `Infantry`, `Cavalry`, and `Artillery` classes. This allows for new army types to be added without affecting the `Player` class. `Mercenary` is an example for this.

High Cohesion

In **ConKUeror**, high cohesion is achieved by organizing classes into packages based on their functionality. For example, all classes related to the game map are located in the `game.map` package, while all classes related to the game deck are located in the `game.deck` package. Each package has a well-defined purpose, and all classes within that package are related to that purpose.

Additionally, each class has a single, well-defined purpose. For example, the `Game` class is responsible for managing the game state and flow, while the `Player` class is responsible for managing the player's armies and territories.

To document the purpose of each class and package, we are using java decorators such as `@Category`, `@Description`, `@Label` and IntelliJ decorators such as `@NotNull` and `@Nullable` etc., for annotations.

Strategy

The Strategy pattern is a design pattern that allows the behavior of a class to be changed at runtime by providing different implementations of a particular algorithm.

For example, the `ChanceCard` class defines a `ChanceCardEffect` interface that represents the effect of the chance card, and the `ChanceCard` class has a `setChanceCardEffect()` method that allows the effect to be set at runtime. Different chance cards are then created by implementing the `ChanceCardEffect` interface with different behavior. For example, the chance card *Bombardment* implements the `ChanceCardEffect` interface with a `rollTwo()` method, while *Reinforcements* allows the player to move armies by implementing the `ChanceCardEffect` interface with a `moveArmies()` method.

The `ChanceCard` class also uses the **Singleton pattern** to ensure that there is only one instance of the `ChanceCardEffect` object for each type of chance card.

Strategy pattern also allows us to define different game rules for different stages. `Game` class uses a `GameRules` interface to define the rules for the game, which is implemented by the `GameInitRules` and `GameTurnRules` classes. This allows for a seamless transition between the initial sharing of territories stage to the rounds stage.

The `ChanceCard` class in **ConKUeror** uses the **Strategy pattern** to allow different chance cards to modify the rules of the game. The `ChanceCard` class defines a `ChanceCardEffect` interface that represents the effect of the chance card, and the `ChanceCard` class has a `setChanceCardEffect()` method that allows the effect to be set at runtime.

Adapter Pattern

There is no necessity to use this pattern, unless we decide to support a network gaming option.

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.
- **Object**: An instance of a class.
- **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.
- **Domain Model**: A conceptual model that represents the real-world entities and their relationships in a particular domain.
- **System:** the **ConKUeror** game and its underlying architecture and components, including the domain model, user interface, and game logic.
- **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.

Logical Architecture

Package Hierarchy for `conkueror`

Package: `domain`

- **Package:** `domain.model`
  - **Package:** `domain.model.game`
  - **Package:** `domain.model.map`
  - **Package:** `domain.model.army`
  - **Package:** `domain.model.player`
  - **Package:** `domain.model.deck`
  - **Package:** `domain.model.card`
- **Package:** `domain.controller`
  - **Package:** `domain.controller.app`
  - **Package:** `domain.controller.listeners`

Package: `view`

- **Package:** `view.screens`
- **Package:** `view.components`
- **Package:** `view.graphics`

Description

Main Packages:

- The `domain.model` package contains classes that define the game logic and mechanics, including the game itself, the game map, the players, and the decks and cards used in the game.
- The `domain.controller` package contains classes that handle user input and manage the game state, including the main application and event listeners.
- The `view` package contains three sub-packages: `view.screens`, `view.components`, and `view.graphics`. These packages contain classes that handle rendering and displaying the game to the user, including different screens, individual UI components, and graphics rendering.

Domain Model Packages

The `domain.model` package contains classes that define the game logic and mechanics, including the game itself, the game map, the players, and the decks and cards used in the game.

- `domain.model.game`: This package contains classes related to the game itself, such as the `Game` class, which manages the game state and controls the flow of the game.

- `domain.model.map`: This package contains classes related to the game map, such as the `Map` class, which defines the layout of the game map and the various territories on it.
- `domain.model.army`: This package contains classes related to armies, such as the `Army` class, which represents a group of units that can be moved around the game map and used to attack other players.
- `domain.model.player`: This package contains classes related to the players, such as the `Player` class, which represents a single player in the game.
- `domain.model.deck`: This package contains classes related to decks and cards, such as the `Deck` and `Card` classes, which define the different types of cards that can be used in the game.
- `domain.model.card`: This package contains classes related to individual cards, such as the `ArmyCard`, `ChanceCard` and `TerritoryCard` classes.

Domain Controller Packages

The `domain.controller` package contains classes that handle user input and manage the game state, including the main application, event listeners, and network communications.

- **Package:** `domain.controller.app`: This package contains classes related to the main application, such as the `App` class, which serves as the entry point for the game and manages the overall application state.
- **Package:** `domain.controller.listeners`: This package contains classes that listen for user input and handle events, such as the `InputListener` class, which listens for mouse and keyboard events.

User Interface Packages

The `view` package contains three sub-packages: `view.screens`, `view.components`, and `view.graphics`. These packages contain classes that handle rendering and displaying the game to the user, including different screens, individual UI components, and graphics rendering.

- **Package:** `view.screens`: This package contains classes related to different screens that the user can interact with, such as the `MenuScreen` and `GameScreen` classes.
- **Package:** `view.components`: This package contains classes related to individual UI components that are used throughout the game, such as buttons, text fields, and menus.
- **Package:** `view.graphics`: This package contains classes related to graphics rendering, such as the `MapGraphics` and `Animation` classes, which define the appearance and behavior of various game elements.

Module Directory Structure

- `conkueror/src`: directory with the three main packages — the source code — `domain.model`, `domain.controller`, `view`
- `conkueror/assets`: all the assets for the game — avatars, buttons, fonts, background images etc.

↗️

Diagrams

Due to pdf format restrictions, we can't include the 4 types of diagrams we are required to submit here, namely, **Package**, **Class**, **Sequence** and **Communication.** *We instead have to include them here for your review:*

📦 Package Diagrams

🎲 Class Diagrams

🔁 Sequence Diagrams

💬 Communication Diagrams

The source code for ConKUeror can be found at over our Github repository, under the main branch.

https://github.com/altunh/2023S_nerd5

Thank you for your attention!

Sincerely, Nerd^5.