SoPra of the dead

Group stage

Table of contents

List of changes

1 Introduction

It is the winter of 2021 and thanks to the numerous vaccinations, COVID-19 finally defeated and normal everyday life is slowly becoming conceivable again. The university sities in the country all switch back to on-site operation one after the other. All weigh in in safety, the students in the lecture halls and the professors at their chairs, when things suddenly get much worse than expected...

A new, much more dangerous virus had quietly crept in and taken over large parts of the world. society into zombies.

2 weeks later...

survivors of the apocalypse have barricaded themselves together at the university and founded a society. The headquarters of the survivors, called the colony, is the canteen building. There is enough space for everyone, including the kindergarten children who are not can take care of themselves or defend themselves. They must also be protected, provided with medical and nourished. Each and every survivor must deal with his or her respective skills to ensure everyone's survival. You have to fight the zombies fight, provide food and find other materials in the area.

And as if that wasn't enough, you also have to fight crises time and again, that haunt the colony...

2. Organizational matters

After the one-week practice phase, the practical part of the software internship begins. Theser is a group phase (~4 weeks) and a subsequent individual phase (~2 weeks) divided.

In the group phase, you will work together with your fellow students in a team of 5-7 people design, implement and test a game. The group allocation takes place by the chair on the first day of the group phase. You will be selected for the group phase in MS Teams added to a team, which serves as your virtual workspace. A Tutor will be at your side and in regular contact with your team.

Voluntary attendance offer

If you have previously indicated that you are interested in the voluntary attendance offer and are assigned to an attendance group, a workroom at the university will be available to your group from Wednesday, 08.09.2021, for the duration of the group phase. Your tutor will let you know which room is available for you.
and agree with you when you can visit your work area for the first time. Please note that proof of a full vaccination, recovery or a negative PoC antigen rapid test is required no later than 24 hours. This proof must be provided daily.

You will use the first one and a half weeks of the group phase for the design, the following approx. two and a half weeks for implementation (including testing). To the Planning the game involves working out rough and detailed designs. You will be drafts to a member of the chair and receive feedback. The implementatization includes the game server with the game logic. In addition, you must have an extensive Develop a test suite that tests as many special cases as possible.

Just like customers in real life, the chair is also very changeable in its requirements for the game to be implemented. Therefore, after the design review there must be a change to the rules of the game. Your design must be flexible enough to the changes in the task without major effort.
The specific changes will then be presented after the draft reviews in a separate document.

After the group phase, you will be given a new task in the individual phase, which must be completed by all participants.
participants must work alone. Further information on the individual phase after the end of the group phase. The prerequisite for the individual phase is successful completion of the group stage.

Note

For the course of the group phase, please also refer to the information in the document "Organizational matters", which was made available to you in the CMS at the beginning of the software internship. It also contains the full timetable and a more detailed description of the hybrid online and face-to-face sessions. Attendance is compulsory during the group phase; please also refer to the "Organizational matters" document for details.

2.1 Draft

In the draft, your group must fulfill the following tasks:

1. you must create a UML class diagram of your system. The class diagram must contain all relevant classes of your implementation, as well as their respective important fields and methods. This should make it possible to recognize which data and which functionality you encapsulate in the respective class. The relations of the classes must be modeled correctly. In the class diagram, all classes must be and all non-trivial methods and their parameters must be listed. To the trivial methods include, in particular, standard getter and setter methods, as well as `hashCode`, `equals` and `toString`.
2. you need to visualize the interaction of your classes using UML sequence diagrams for the following 2 scenarios:
   - Initialization of the game
   - Searching a location with a random encounter where 1 child is also included in the search.

     Game coming

   In the change in requirements after the design review, we will add a third scenario for which you want to analyze the interaction of your classes on the basis of a further UML sequence diagram.
3. you must draw up a detailed schedule and work plan for the implementation phase including testing. This plan specifies who is responsible for which parts of the implementation and who will be responsible for testing which components is responsible. In this plan, pay particular attention to the dependencies between between the various components and the sequence of the next milestones. Don't forget to include writing the integration and system tests in the plan. take into account.

Design review & design approval

In the design review you will receive feedback on your design from a member of the chair. Design. At the design approval stage, your design will be checked by a member of the chair's staff. taken. Both dates are mandatory in presence - please note, that proof of full vaccination, recovery or a negative test result is required. PoC antigen rapid test not older than 24 hours is required.

After the design reviews, we will make changes to the brief. The Changes will be based on the original game. Try to make your design as modular as possible that any changes can be easily integrated into your design. You should however, not all possible changes can be incorporated into the draft in advance.

In both cases, please note that the current status of your draft, which you submit to the chair employees, already 1 hour before your assigned review or meeting time. acceptance date must be in our GitLab *repository* (see also submissions).

2.2 Implementation

In this phase, you will complete the actual implementation of the game simulation in your group. lators. You will also create unit tests, integration tests and system tests. Your game Implementation must meet the following requirements:

1. your implementation must pass our system tests. Our sys- tests are regularly performed on your implementation during the implementation phase. The test results will be sent to you in due course at regular intervals. made available at intervals.

2. your implementation must fulfill the unit tests, integration tests and pass system tests. The unit tests serve as an aid for troubleshooting and Correction. If you have unit tests that your implementation does not pass, you have You either made a mistake when creating the tests or your implementation is still faulty. With your tests, you should achieve the highest possible code coverage. and test scenarios that make as much sense as possible, which can be achieved by finding mutants[1] is secured.

3 We will analyze your implementation with the code analysis tools *PMD* and *SpotBugs*. tersearch. We do not expect to find any problems here. If the tools in If you report problems with your code, we expect you to be able to provide sufficient justification, why you have not fixed them. With the build script that we have created for you you can select these tools yourself from the start of the implementation phase. and check whether your code still has problems.

Simply passing the tests is not enough, your code must also be well structured and follow the concepts of the lecture. In particular, each individual group member must

[1] https://en.wikipedia.org/wiki/Mutation_testing

contribute a significant share to the code of your group. (Make sure you pay attention to correctly configured *git* on your computer so that your commits are also available in our *GitLab* are assigned to you).

Codereview

In the code review, a member of the chair will look at your code together with you, identify weaknesses and make specific suggestions for improvement. As the Codereview is a face-to-face meeting, proof of a complete and accurate Vaccination, recovery, or a negative PoC antigen rapid test not older than 24 hours required.

2.3 Tools required

To take part in the software internship, you will need your own laptop, which you must bring yourself.
you need to bring with you. You must install the following tools on your device:

- *Java 16* (our reference platform works with *OpenJDK 16.0.2*)[2]
- Version management system *git*[3]
- IDE of your choice (we recommend *IntelliJ*)[4]

The *Gradle* build system, which we use, does not need to be explicitly installed on your system. , because we provide you with a project in which a *Gradle wrapper* is already supplied, which takes care of the installation itself. All further Tools that you need do not need to be installed, but are provided by us in the project is already provided preconfigured.

All libraries entered in the *Gradle* configuration *file* may be used. are bound. Other libraries may not be used.

2.4 Levies

We will provide you with a *git* repository in the chair's own *GitLab*[5] for the group phase. available. In order to be able to clone the repository provided or to import it into the repo sitory, you need an *SSH key*. How to generate an SSH *key* and what other technical requirements must be met to use the repository. you have already learned in the practical exercise. In the group phase, the entire group will work on one repository.

23 https://jdk.java.net/16/https://git-scm.com/[4] https://www.jetbrains.com/idea/[5] https://sopra.se.cs.uni-saarland.de/

You will be provided with a separate branch (not `master`) for submitting your design. made available. All design documents and associated representations must be uploaded to the repository. can be pushed. You can also use photos of your diagrams on paper, etc. as long as the resolution is high enough and details are recognizable. The final ver sion of your design must be marked with the specified tag[6] `design approval` , which must be attached to the submitted revision. With the *git* command `git tag <tagname>` you can create a tag for the current commit. Tags must be ex- be pushed explicitly. Please note that your draft must be submitted no later than one hour must be pushed to the repository before the acceptance date assigned to you.

For your implementation, you must then use the master `branch` after design approval. use. There you will be provided with a project with a minimal code framework. Other branches are provided by us in no longer taken into account in the implementation phase.

For the final submission of your implementation, the last commit on the master Branch, which must be received by no later than ~~23:59~~ 19:59 CEST on the day of the code acceptance in the repository has been made. It is your responsibility to check that the push has arrived in the repository and everything is up to date on time. Your submissions must be executable on our reference platform (*Debian 10* with *OpenJDK 16*).

[6] https://git-scm.com/book/en/v2/Git-Basics-Tagging

3. The game

3.1 Game setup

SoPra of the Dead is a cooperative board game in which the players try to play a game of to achieve a jointly set goal before time runs out or nerves are frayed. of the colony are bare.

Analogous to a physical board game, there is a world (the game board, so to speak) in which there **is** the **colony** and various **locations**.

The players control various **characters** (the *survivors* of the crisis), who can perform **actions** each round. For example, you can search for food and materials in the surrounding area, attack zombies and barricade entrances.

Whenever a character might encounter a zombie, there is a certain probability that the zombie probability of contracting an **infection** or injury.

There are various **cards** in the game, e.g. food cards, which add food markers to the Add food supply, or Crisis cards drawn from the Crisis card deck. be.

cards played by players end up in the *waste pile*. This Garbage must be disposed of by the colony so that **morale** does not drop.

Which actions a character can carry out depends on their **abilities**, on the one hand. and, if applicable, how much the action costs him. In this case, the player or the player rolls a number with a so-called *action die*, which is high enough for the character to be able to perform this action.

Each round of the game consists of **two consecutive phases:**

- The *player phase*, in which all players can carry out their actions and a new **crisis** is uncovered.
- The *colony phase*, in which certain points are determined for the entire colony at the end of each round must be dealt with.

The game ends as soon as the common goal has been achieved, the rounds have expired or the morale of the colony drops to 0.

Note

Instead of specifying all game parameters during implementation, we will instead read in a **configuration file** at the start of the game server, which models the game to a large extent. This is used, for example, to specify the goal of the game, the actual game world, but also characters and their abilities.

3.2 Game details

3.2.1 Characters

At the start of the game, each player is given four characters to choose from. must choose characters. All players therefore start with two so-called **survivors** into the game, however, during the course of the game they can (and probably will) additional survivors can be added to the player's party. The number of characters per player is not limited.

There are also helpless **children**. These are not assigned to a player, but exist are simply in the colony and need to be cared for.

The properties of a character are specified in the configuration file:

- Each character has an ID (`identifier`) and a `name`.
- Each survivor has *exactly one* special `ability`. Some special The abilities of survivors require the use of action dice. All Abilities that require this are also indicated. A list of the special skills The different options can be found in Table 1 and Table 2.
- The `attack` **value** indicates how high the number of points of an action dice can be. so that the character can attack a zombie.
- The `search` **value** indicates how high the number of points on an action die is. must be in order for the character `to` search a location.
- Each character has a certain **social** `status` in the colony, which is defined in is specified in the configuration file. The starting player is the player who has the survivor with the highest social status. The character with the lowest social status is always attacked first in zombie attacks. Children have no social al status, and are therefore treated as the weakest members of the colony.

As soon as the last character or survivor of a player has been killed, the player loses this player removes all cards from his hand (the cards are removed from the game, they are returned to the not on the garbage pile). In the event that it is the player's turn, this is immediately is completed. A new survivor is then selected, who is then placed directly in the Colony spawns (as specified in subsection 4.2.3).

Note

Spawning (from the English *to spawn*) is a common term to describe the (re)introduction of a character in computer games.

3.2.2 Colony

All characters spawn in the *colony*, which has unlimited space for the survivors. (incl. children).

From there, the survivors can move to other locations (max. one movement per round) to search for food and materials. The colony has a total of as many inputs as specified in the configuration file. Before each of these inputs , there are three spaces where one zombie can be placed. This means that up to to **three** zombies lurking in front of each entrance.

There is also garbage in the colony. This must be regularly removed by a survivor be emptied in the colony so that morale does not drop. Children never leave the colony, because they can do nothing.

The properties of the colony, which are specified in the configuration file:

- the ID (`identifier`)
- the number of `entrances` (`entrances`)
- The `startCards` represent the starting pile of cards from which all players start. players receive their five cards at the start of the game.

3.2.3 Morality

The colony has a morale value. This is specified in the configuration file and changes in the following cases:

- As soon as a survivor or child dies, morale drops by 1.
- If a crisis is not overcome, morale falls by the amount indicated in the crisis. Value.
- If two more cards than required are contributed to a crisis, morale **increases** at 1.

- Morale decreases by 1 per round for each Hunger Maker in the food supply.
- The colony loses one morale for every 10 undisposed cards in the trash.

If the morale drops to 0, the game is lost.

3.2.4. locations

In addition to the colony, there are other locations that are specified in the configuration file. the. The configuration also indicates *how many survivors* are at the respective location at the same time. location, as well as the number of entrances to the location. In front of each of these entrances, there are also three places where a zombie can be placed. can. This means that up to **three** zombies can be lurking in front of each entrance.

The properties of a location are specified in the configuration file:

- Each location has an ID (`identifier`) and a `name`.
- the number of `entrances (entrances)`
- how many survivors can be at the respective location at the same time (`survivorSpaces`)
- Each location has a pile of `cards` that can be searched.

Each time a location is searched, there is a certain chance of finding a new over- living person. This person then joins the player's group and can still can be used in the same round. However, new characters always spawn in the Colony.

There are the following cases that can occur when searching a location:

- It cannot be searched. This is the case if there are no more cards on the location or the player has no more action dice left with a number ≥ Search value of the character.
- A random encounter takes place (= a new character spawns) instead of the Search.
- There is no random encounter and it is searched normally (= one card drawn).

Therefore, as long as there are cards in the deck, either a search (= a card drawn), or a new character (plus any children) comes into play. How exactly the random encounters are calculated is described in subsection 4.2.6.

3.2.5 Infections

By rolling an *infection die*, characters receive injuries (wounds, diseases, etc.). freezing or biting). Injuries can always happen when you are in contact with interacts with a zombie. An interaction with a zombie always occurs when you move normally (exception see card FUEL, table 3), attack a zombie or you are attacked by a zombie.

Note

This cube has a total of twelve sides, three of which represent a wound, two a frostbite and one a bite. You can find out exactly how the cube is implemented in section 4.2.

- If **a wound or frostbite** is rolled when the infection die is rolled, then the character receives the corresponding wound.
- If a **bite** is rolled, the character dies and the morale of the colony drops by 1 and the infection spreads to its location. For this, the character with the lowest social status also rolls the infection die. Should he suffer an injury, he dies, the morale of the colony drops again because of this by 1 and the infection spreads further. Otherwise the character survives and the spread of the infection ends. If the last character of a player die, a new character is spawned first and then the chain of infection is broken. dealt with further.
- For each frostbite, a survivor at the start of the first turn of the next round of his player receives another wound.
- As soon as a character suffers the third injury (wound or frostbite!), he dies.
- In any other case, nothing happens.

If all of a player's survivors die during the game, the player is his or her turn is over, he or she loses all cards in hand [1] and he or she receives randomly assigns a new survivor, who then spawns in the colony. With he or she can then continue as normal in the next round. play.

If no survivor can respawn, the player is out.

[1]These are removed from the game, they are not placed in the waste pile.

***Example:* Movement with infection**

The player Julia goes with her character Bob from the colony to the *petrol station* location, where the characters Curd (social status 5), Henry (social status 2) and Charlotte (social status 3) are also located.

→ Julia rolls the infection die for Bob and gets an 11. Bob is bitten by a zombie and dies. The morale of the colony drops by 1 (fortunately, the morale has not yet dropped to 0, otherwise the game would be lost at this point). The infection die is then rolled for Henry, as he has the lowest social status of the remaining characters at the petrol station.

The result is 5, i.e. just lucky. Henry is not wounded, remains alive and the infection does not spread any further.

3.2.6 Crises

The survivors are afflicted by a new crisis every round (i.e. there is a new crisis cards are drawn from the crisis card deck). For each crisis, a certain amount of number of cards of a certain card type can be sacrificed. The value of the card does not matter.

It is specified in the configuration file:

- the ID of the crisis (`identifier`).
- how much morale the colony will lose (`moralChange`) if they do not overcome the crisis, i.e. have not contributed enough or the right cards
- How many cards must be contributed to overcome the crisis (`requiredCards`)
- which card types must be contributed (`food`, `stuff`, `medicine` or `fuel`)

3.2.7. skills

As we already know, each survivor has exactly one special **ability**, which are divided into **four active** and **four passive** skills.

Active skills must be actively used in moves, whereas passive skills must be automatically influence the corresponding actions.

A character with the ACTIVE ability

`BARRICADE` can create barricades more easily.
`KILL` can kill zombies more easily.
`FEED` can add more food to the food supply more easily.
`HEAL` can heal injuries.

There are a total of four PASSIVE skills:

`NO-INFECTION` has an influence on injuries that can occur during attacks.
`WOUND` has an influence on injuries that can occur during movements.
`SEARCH` influences how a character may search a location.
`TRASH` influences how much waste a character can dispose of.

You will find all (technical) details on capabilities in tables 1 and 2.

3.2.8. cards

All players start the game with five cards in their hand. You can get new cards by searching locations (= one card from the card deck of the location). Cards played from your hand are immediately placed in the *waste pile*.

There are a total of **six card types**:

`LOCK`: influences how easily an entrance can be barricaded `SCISSORS`: influences how many zombies can be killed `STUFF`: you can reroll the action die with the lowest number `FUEL`: influences movements or the killing of zombies
`FOOD`: food can be contributed to the food supply
`MEDICINE`: Injuries can be healed

3.2.9. actions

The players control various characters (the *survivors* of the crisis), who can perform actions each round. For example, you can search for food and materials in search the surroundings, attack zombies or barricade locations.

For some actions, action cubes must be used and thus consumed, for others do not. At the start of each round, each player is given as many shares as they have. on cubes equal to the size of his or her group of survivors (plus one extra).

$$\#ActionCubePlayer = 1 + \#SurvivingPlayer$$

How exactly a player can plan their actions with the action dice is described in subsections. explained in more detail in section 3.3.1.

In this section, we will first discuss the possible actions in general terms, so- and the influence of skills on these. The details on configurability can be found in please refer to the corresponding tables.

(**1) Actions that *do not* require *an* action die:**

- **Contribute to the crisis** (= sacrifice hand cards for the crisis)
- **Moving character between locations**

  Each character can move once per round. As you can move outside a location zombie, there is always a certain risk of injury. (see subsection 3.2.5).

  Influence of skills:
  If the character has the passive ability `WOUND`, they gain instead of the violation rolled, the type of violation specified in the configuration file. file is specified for its capability.

  Note: By playing the `FUEL` card, the character can move around without sustaining an injury.
- **Playing cards and using (almost all) active skills**

  Playing cards and using the active abilities of the characters are active decisions made by a `player`.
  Many actions can be performed either with the corresponding ability or with the card can be executed:

  - add food to the food supply
    * by playing a FOOD `card`
    * by using a character's FEED `ability` (1x per round)
  - Healing the injuries of a survivor
    * by playing a MEDICINE `card`
    * by using the HEAL `ability` (1x per round)
  - Create barricade(s)
    * on exactly one space of an entrance by playing a LOCK `card`
    * by using the `BARRICADE` capability (highly configurable)

- Kill zombies
  - Exactly one zombie at an entrance by playing a SCISSORS `card`
  - or by using the active ability `KILL` (highly configurable, requires possibly an action cube)

  We would like to mention in advance that both the playing of cards (`UseCard-Command`), as well as the use of a `capability` (UseAbility-Command) from the server is not like an attack on zombies (`attack-Command`) is dealt with. Technical details follow in section 4.4.
- By playing the `FUEL` card, the character can move without moving. an injury.
- When the `STUFF` card is played, the player can use his or her low reroll the highest action die.

(**2) Actions that require action dice:**

- Dispose of waste

Each round, a player can roll **three** action dice (regardless of the number) at the cost of one.
Remove cards from the waste pile.

Influence of skills:
If a character has the `TRASH` passive ability, they can play as many cards as they want. as specified in the configuration file for its ability. A player or a player can play with *any* character that has this ability in a round. Dispose of waste.

- Barricade

For each space of an entrance that a player wants to barricade in a round, he must use one action cube (regardless of the number).

Influence of skills:
Note at this point: A player can perform the *Barricade* action on Cos- of an action cube at any time (provided there are still free places on an action cube). entrance of the location where the player is standing).
This `Barricade command` action is technically not the same as the `Barricade action`. same as the use of a character's active ability `BARRICADE` (UseAbility-Command) to create barricades or play the `LOCK` card (UseCard-Command) to create a barricade, even if the result is of course the same in all

three cases is the same: One (or more) places of an entrance are blocked. caded.

- Attack zombies

For a character to be able to attack a zombie at their location, the player must or the player uses one action cube, with

$$\text{Number of points} \geq \text{Attack}_{\text{valueCharacter}}$$

After the zombie has been killed, the injury that the cha- rakter could have contracted as a result of the attack (see Table 3.2.5). In the worst case In this case, the survivor is bitten and dies. A zombie, on the other hand, dies in the always through an attack.

Influence of skills:
If the character has the passive ability `NO-INFECTION`, no infection will occur. dice are rolled after the attack.

- Search locations

Searching means that you take the top card from the location's card deck. pulls. To search a character's location (see subsection 3.2.4), the player must use up one action die, with

$$\text{Number of points} \geq \text{Search}_{\text{valueCharacter}}$$

Influence of skills:
If the character has the `SEARCH` passive ability for this location, he may the player can draw more than one card. For full details of this ability, see Table 1.

***Example:* Attacking a zombie**
A player stands at the *school* location with his survivor Tina and wants to attack zombies.

Let's assume that Tina's attack value is 5 ≤ the value of one of the player's remaining action dice.

→ The zombie is killed and removed from the game. Next, the infection die is rolled for Tina, as she has interacted with a zombie. The result is 7, i.e. Tina suffers a wound. As it is Tina's first wound, nothing else happens.

3.3 Gameplay

3.3.1 Player phase

At the start of the game, all players receive a total of five *cards* from the starting deck. These cards correspond to various actions, such as generating food, making an over-Heal living or create a barricade. To get more cards during the game , locations must be searched. The number of cards that a game player can have in his hand is not limited.

After a card has been played, it is placed in the waste pile.

All map types can be found in Table 3 and Table 4.

The player phase then proceeds as follows per round:

1. a **crisis** must be overcome in each round: a new crisis card is drawn from the Crisis card pile of the game is revealed.
2. then ALL players roll their **action dice**. The action dice are simple six-sided cubes.[2].

   The total number of dice rolled is equal to the number of players and survivors in the game. gives. You roll the dice once per player and per survivor (children do not count here to this), i.e. if a player has a group of three survivors, he rolls a total of four times.

   You can assign your action dice to any of your survivors' actions. share. It applies in particular that if a character of a player dies during the current round, no dice are lost. The same applies to If you do not add a new one, you should add an additional one in the round. character.

Note

The number of dice that can be used in a round does not change during the round. It is only adjusted to the number of remaining characters of the player(s) at the start of the round.

The players can now use these values to determine their actions for this round. plan. It is important to note that the smallest action cube (or the smallest number rolled) that can be used to perform the action.

You can perform any number of actions per round. The number of

[2]You can find out how to implement the cubes in section 4.2

Actions that a player can perform with his survivors are basically limited only by the number of cards in the player's hand, as well as by what Character depending on his abilities etc. with the action dice rolled of actions is possible.

3 Then the players start their turn in the same order. moves to complete. As soon as a player has finished all their actions, the next player. The maximum number of players is specified in the Configuration file specified.

3.3.2 Colony phase

As soon as all players have finished their turns, the colony phase. The colony phase gradually deals with various points:

1. **Distribute food:** Food must be provided for the people in the colony. The amount of food to be provided is determined as follows:

$$\#Food = [\,(_{\#SurvivorsatColony} + \#Children)/2\,[$$

The required food is taken from the **food supply**. The food supply of the Colony consists of the food markers, which are created by playing FOOD cards. have entered the game (see Table 3), and is accordingly blank.

If there is not enough food in the food supply at the end of a round, to feed EVERYONE according to the above formula, no food is taken. Instead, a **hunger marker** is added to the food supply. Per hunger marker in the food supply, the colony loses one morale per round. There are no possibility to remove the starvation markers, so the colony should be At best, always make sure you eat enough.

2. **check waste:** The number of cards in the waste pile is checked. Per 10 cards in the waste pile, the colony loses one morale in this step. Every time, When a card is played, it ends up in the waste pile. Excluded from this are cards that are contributed to the crisis. A player without the corresponding special ability can dispose of three cards of trash per round.

3. **deal with the crisis:** This step checks whether there are enough cards for the crisis. were contributed. If this is the case, the crisis has been successfully averted. Should two more cards than needed have been added to the crisis, the morale increases by one. If the crisis fails, the morale drops by the amount specified at the time of the crisis. Value.

4. **bring zombies into the game:** New zombies are spawned in this step:

(1) The zombies spawn at the colony first. The number of zombies that spawn at the colony depends on the number of survivors in the colony:

$$_{\#SpawningZombiesColony} = [(\#Survivors + \#Children)/2[$$

(2) Zombies then spawn at the other locations. The order indicates the `locationID` (ascending order). As many zombies spawn at these locations as there are survivors at the location:

$$_{\#SpawningZombiesLocation} = \#Survivors$$

When zombies spawn, all free spaces at an entrance are occupied first. Here the zombies are spread evenly across the entrances to the colony.

The process always starts with the input with the lowest `entrance` and then continue in ascending order. If there are more zombies than entrances spawn, you start again with the entrance with the smallest `entrance`, until all zombies are spawned.

After that, barricades (if any) are removed instead of a zombie spawning.

As there is a free space again after the first barricade has been removed, this space is first before another barricade is destroyed.

If all spaces are occupied by zombies and another zombie spawns, the zombie dies. a survivor instead.

Which survivor dies d e p e n d s on the social status of the survivor. The Survivors with the lowest social status die first. Children always have a lower social status than other survivors.

5 **Review the common goal:** In the last step of the colony phase, the common The same goal is checked and it is determined whether the game has been won.

A few examples of a common goal are:

→ Survive until the rounds have expired
→ Create a certain number of barricades
→ ...

After the colony phase, a new round begins.

3.4 The original game

This year's game is (very strongly) inspired by "Winter *of the* Dead" (*Dead of Winter*).

This tutorial Learn to Play Dead of Winter in 16 minutes (YouTube) can help you to get an overview of the game for which you will spend the next few weeks together. develop a game server.

In order to make the implementation of the game server easier, we have made the game slightly amended.

- The *exposure die* corresponds to our infection die.
- The *Helpless Survivors* became children in our person.
- There is no *first player token*, the starting player always remains the same.
- There is no *noise* - in the original game you can make noise, e.g. when passing through look for another card to draw. However, zombies are attracted in return. This gives not available in our version of the game.

Explicitly irrelevant:

- 02:37 to 02:58 (Secret Objective) There is neither the *Secret* nor *Personal Objective* in our version. There is only the common goal of the colony.
- 09:08 to 10:42 (Food marker to increase action cube values, traitor, Rule for exile)
- 15:24 to end (How betrayal works)

3.5 FAQ

- **Why do you get four characters to choose from when you only get two? used? Why can't you save yourself the step?**

  Imagine it like this: The game begins for the players with the fact that the common me target is read out or displayed. In other words, each player will play those characters whose skills are likely to contribute to the achievement of the common goal could be particularly helpful. The game server you are implementing, is not interested in this freedom of choice :-) Nevertheless, he must provide this functionality.

- **What was that again with the dice?**

  All action dice are rolled at the start of the round. One die is rolled per players and survivors, i.e. when a player controls five characters, she has $5 + 1 = 6$ action cubes for this round.

  The infection die is always rolled when a character moves normally (without FUEL) or attacks a zombie. In other words: Every time a survivor from one location to the next, there is a certain probability of a of encountering a zombie and being injured. This probability is calculated using Infection cube modeled.

- Where do you get which cards now?

At the start of the game, all players receive *five* cards each from the Start card stack. This stack is specified in the configuration file. During the game, you can only get more cards by searching a location.

Which cards are at a location is also specified in the specification file.

- How many cards may a player have in his or her hand?

There is no limit to the number of cards a player can have in their hand.

- How many actions can a player perform per round?
lead?

The number of actions a player can perform with their survivors, is in principle only limited by the cards in the player's hand and what the characters that the player controls, depending on their abilities etc. with the action dice rolled.

- Which cards end up in the trash pile?

All cards played by the players end up in the waste bin. pile. The only exceptions to this are cards that a player loses if one of his or her survivors dies. Crisis cards do not count to the possible cards in hand. These cards are only used to minimize the crisis per round.

to uncover.

- Does every player have to contribute the same number of cards to the crisis?

Basically no. It can happen that players have different numbers of their cards in your hand. This depends on which cards are needed and which cards the players have in their hand. (Here it is important to To act in the interest of the Community ;-)

- In which order are the dice rolled?

The player phase begins with the following in the order in which they log on to the server the action dice of all players are rolled. Technical details following in chapter 4.

- What are the options for adding new characters to the game?
can?

When searching locations, random encounters may occur in which a character is spawned. New characters also come into play after the last character of a player has died. New characters *always* spawn in the colony. For technical details see chapter 4.

- How many times per round can a character move?

Each character may only move once per round. In particular A character with the FUEL `card` cannot move additionally. It can only be used for a movement can be used.

- Can there also be children at locations?

No, children only exist in the colony. If you click on children (see subsection 4.2.6), they spawn in the colony and stay there. In a random encounter, 0-3 children always spawn.

- At the end of the round, check whether there is enough food left to all, the colony then loses one morale per unsupplied person. Survivor?

No. If there *is* not enough food for everyone *in the colony* at the end of the round *survivors* (including children), then exactly 1 hunger markers are added, regardless of how much food was missing. At the same time, no Food markers are removed from the food supply. (*In other words: Either all of them are destroyed. or everyone must starve*). Then, for each hunger marker in the food supply one morale is deducted. The hunger tokens remain in the near field until the end of the game. and cost one morale each round.

- Is the colony a site?

Yes, but there are no piles of cards to search through and the colony has no Space restriction for survivors.

- If my character has the KILL `ability`...do I have to use the Roll an infection die when he kills a zombie? That would be totally impractical. table.

That would be very impractical, exactly. And that's why it's important to understand: `KILL` counts NOT an attack in that sense, so only one infection die must be rolled, if the configuration file specifies this.

4. Technical details

In this chapter, we explain further technical details - from the build script, through code analysis tools through to technical implementation details. Here we explain also the framework that is provided to you by us and used by you. has to be.

4.1 Game sequence

The game is divided into different phases. It begins with the *registration phase*, which is followed by the *preparation phase* and finally the actual game. In the following , the individual phases are described in detail.

4.1.1 Objective

A **common target** is defined for each game in the configuration file (section 4.3). nized. This goal specifies the conditions that must be met in order to win the game. is. It also indicates how many zombies and how many helpless children are in the game is started.

4.1.2 Registration phase

During the registration phase, the server waits for all players to register. Register players who want to take part in the game. Each player will be given is assigned a `playerID`, which starts at 0 and is always incremented by 1.

During this phase, a player who is already registered can start the game, even if there are still the maximum number of players is registered.

During the registration phase, a timeout causes the server to terminate. Otherwise treats a timeout as if the server had received a `leave()` command.

4.1.3 Preparation phase

The preparation phase starts after either the maximum number of players or the maximum number of players is reached or when a registered player starts the game. As soon as the phase starts, the server first informs all clients which players are playing (`player events`).

Then, for each player (in ascending order of `playerId`), four over-
living ones are drawn and communicated. The player then selects two of these.
The five cards in hand are then drawn for the same player. Then
the game continues with the next player. As soon as all players have been dealt with
, the actual game begins.

4.2 Random calculations

In order to keep our game deterministic despite the many random calculations, we use
for all actions that are based on randomness, the *java.util.Random random number generator*,
which we initialize with a given seed.

Why is this important?

" *If the seed parameter is passed, the random number generator initializes its internal counter with this value, and the subsequently generated sequence of random numbers is reprodu-
catable. If, on the other hand, the parameterless constructor is called, it initializes the random number generator based on the current system time. The random number sequence in this Case not reproducible."* [1]

To ensure that all calculations are performed with the correct number from this initialized, random number
sequence, it is essential that you adhere exactly to the specified order and sequence.
Maintain the frequency of random calculations.

4.2.1. maps

When initializing the game, we first use the random `object` to create the cards.
mix.

We first shuffle the starting pile of cards and then the piles of the other cards.
locations in ascending order of their IDs.

We shuffle the cards by using *Collections.shuffle()*:

```
Collections.shuffle(cards, random);
```

The first card from the shuffled deck is always drawn during the game.

4.2.2. crises

After the piles of cards have been shuffled, the crises must also be shuffled. These
are also shuffled with *Collections.shuffle().*

```
Collections.shuffle(cards, random);
```

[1] https://dbs.cs.uni-duesseldorf.de/lehre/docs/java/javabuch/html/k100105.html

Here, too, the first card from the crisis card deck is always drawn during the game. gen.

4.2.3 Survivors

After the crisis cards have been shuffled, survivors are randomly selected who should be brought into play. We implement this as follows:

1. before the characters are drawn for a player, we shuffle all the characters through:
   ```
   Collections.shuffle(survivors, random);
   ```
2. we select the first four characters.
3. we add the two characters not selected by the player back to the back of the Amount of unused characters.
4. We continue with the next player.

After all the players have chosen their characters, we shuffle all the characters together. one last time (as in step 1). **From now on, the first character from the taken from the beginning of the list when a new character enters the game.**

It is important to always call *Collections.shuffle()* with the Random `object`!

Note

The characters are stored in a list in the order in which they are read from the JSON schema. The state of the shuffled list depends on the initial state of the list,
i.e. so that the state of the game is deterministic depending on the seed, the objects must logically also be read in from the schemas in the specified order.

4.2.4 Action cube

Action dice are normal six-sided dice. To roll these, we call the *next-Int()* method of the random `object` as follows:

```
    random.nextInt(6) + 1;
```

4.2.5. infection cube

The infection cube is a twelve-sided cube in which, however, no numbers but wounds are thrown. To simulate this, we again use the *random.next-Int()* method:

```
    random.nextInt(12);
```

The following applies to the result n:

- $0 \leq n < 6$: **no** wounding
- $6 \leq n < 9$: one **wound**
- $9 \leq n < 11$: one **frostbite**
- 11 : one **bite**

4.2.6 Random encounters

characters can be created not only at the start of the game, but also through random encounters.
come into play. Random encounters can always be found when searching a location.
occur. Whether a chance encounter occurs and whether, in addition to the character children still come into play is calculated as follows:

1. check whether there are any cards left in the location's deck.
   - If not, the search will fail and the random encounter will also take place.
     does not take place.
2. check whether all characters are already in the game or whether there are still unused cha-
   tions.
   - If there are no more unused characters, there is no random encounter.
     and the search is carried out normally.
3. check whether there is a chance encounter:
   - To do this, we again take a random number with *random.nextInt()*,
     where the upper limit is the number of cards initially in the deck.
     were to take.
     If the random number is a 0, a random encounter occurs, i.e. the

32 first character of the characters not yet in the game is selected

and spwant immediately in the colony. Then *random.nextInt()* is used again. is called with 4 as the upper limit. The result of this method call returns indicates how many children enter the game. These then spawn directly in the Colony.

This ends the search and no more cards are drawn.

- If there is no random encounter, the search is started normally. (= drawing a card).

Example: **Searching without a random encounter**
One player is at the *movie theater* location with her survivor Alice, whose deck contains two cards (out of an initial 7).

We assume that the number of one of the player's remaining action dice is ≥ Alice's search value. There are also characters left that are not yet in the game.

→ The player now rolls a 7-sided die. The result is 3, so there is no chance encounter and the player draws a card from the movie theater's deck. This ends the search.

Example: **Searching with random encounter**
One player is at the *library* location with his survivor Bob, whose deck still contains one card (out of an initial 5).

We assume that Bob's search value is ≤ the value of one of the player's remaining action dice. There are also characters left that are not yet in the game.

→ The player now rolls a 5-sided dice. The result is 0, so there is a random encounter. First, a new character is drawn and spawns in the colony. Then the player rolls a 4-sided die. The result is 2, so 2 children now spawn in the colony and the search is over.

4.3. configuration file

Instead of setting all game parameters during implementation, we will instead reads a configuration file when the game server is started, which the game can use to large parts are modeled. This is used, for example, to achieve the aim of the game, the actual Game world, but also characters and their abilities are predefined.

We use JSON as notation[2]. In addition, we specify schemas[3] that describe the structure and, if necessary, define value ranges for fields in the configuration file.

It makes sense to deal with these formats in detail, as there is already a many aspects of the game can be defined.

Prerequisites for a valid configuration file:

- There must be a sufficient number of survivors:

$$\#Survivors \geq maxPlayers * 2 + 2$$

- There must be enough cards per location:

$$\#CardsLocation \geq maxPlayers * 5$$

- There must be at least as many crisis cards as rounds.
- Sufficient starting cards must be available:

$$\#StartCards \geq maxPlayers * 5$$

- No two characters may have the same social status.
- The `ID` of a card must be *unique*, and a card may only be used in exactly one Stack max. 1 time.

Note: Requirements for the configuration file do not necessarily correspond to the requirements of the the general state of the game.

Example of a configuration file:

```
{
  "cards":[
    {
      "food ":{
        "identifier":3,
        "amount" : 3
      }
    },
```

[2]https://www.json.org[3]
https://json-schema.org

```
    {
      "hammer":{
        "identifier":1
      }
    }
  ],
  "characters":[
    {
      "name":"Carla␣Thompson",
      "identifier":1,
      "status":22,
      "attack":4,
      "search":2,
      "ability":{
        "search":{
          "location":1,
          "numCards":1
        }
      }
    }
  ],
  "goal":{
    "rounds":10,
    "moral":7,
    "zombies Colony":6,
    "zombiesLocations":1,
    "survive":true,
    "childrenInColony":4
  },
  "locations":[
    {
      "colony":{
        "identifier":42,
        "entrances":6,
      "startCards":[
          1,
          2,
          3,
          4,
          5
        ]
      }
    },
    {
        "name":"Police␣Station",
        "identifier":1,
        "entrances":1,
        "survivorSpaces":3,
        "cards":
          [
           10105,
           10106,
           10107,
           10108,
           10010,
           10011
          ]
      }
   ]
}
```

4.4 Client-server communication

4.4.1 Commands & Events

The game is controlled by a server, which communicates with the clients (players) via *commands & events* are communicated. *Commands* are sent from the clients to the server to tell it what the player wants to do. The server in turn uses *events* to to tell the clients what has changed in the game. We differentiate between the following events between individual events and broadcast events. Individual events are only sent to the players to whom it applies, while broadcast events are sent to all players must. The CommLib only ever sends events to exactly one player, i.e. the server must ensure that broadcast events are actually sent to everyone. To do this, each player has a CommId (in addition to their ID). This is automatically is assigned by the CommLib and must be mapped to the PlayerId. All commands receive the CommId of the executing player as a parameter. For the Sending the events also requires the CommId as a parameter.

All commands that are sent by non-registered clients will be deleted during ignored during the game.

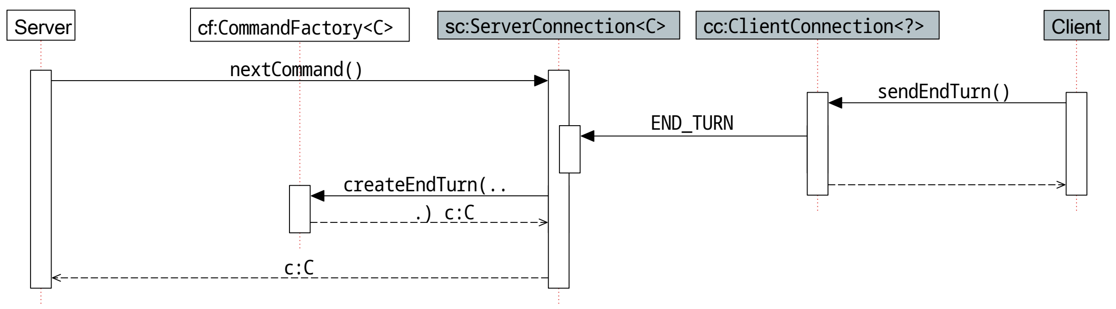

Figure 1: Server-side interaction of the `CommandFactory` with the `ServerConnection`.

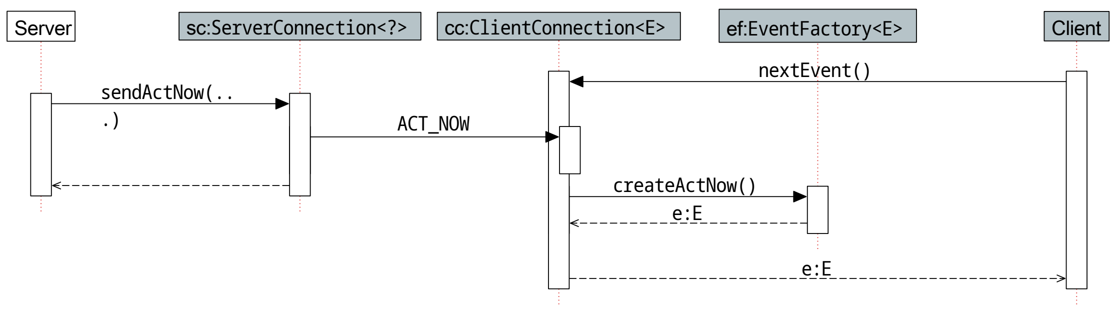

Figure 2: Client-side interaction of the `EventFactory` with the `ClientConnection`.

4.5 Command line interface

The server is started with these command line parameters:

`--config <path>` The path to the file from which the configuration in JSON *format* is to be loaded.

`--port <int>` The port on which the client can communicate with the server.

`--seed <long>` The seed with which the game's random number generator is initialized. (see random calculations).

`--timeout <int>` The timeout of the server in seconds (the maximum time that the server can waits for commands from the client).

4.6. build script

We provide you with a build script. A build script is responsible for dependencies of a software project and create a finished project from the project files, executable program. We use *Gradle*[4] as our build tool.

[4]`https://gradle.org/`

The build script can be found in the file `build.gradle`. Changes to your build script are reset by the test server for execution. You can (and should) use the pro build the project yourself by either executing the `./gradlew build` command or using the Execute *Gradle* task `build` directly in an IDE (e.g. *IntelliJ*). We use *Java 16*. build script contains dependencies on various libraries. We recommend that you to look at these libraries and consider whether you can use them. More Libraries that are not entered in the build script must not be used. From For security reasons, we do not allow any reflection, network connections or other connections to the outside world with the exception of the *CommLib* used (see Client-Server Communication). With the exception of loading configuration files, you may also - do not read, create or change any files in the file system. For loading configuration files may only point to the already existing resources `directory` in the project repository can be accessed.

4.7 Code quality

Good code quality helps to ensure that software remains readable and errors are better avoided can. In the software internship, you will get to know automated tools that you can use to to help you write good and error-free code. You can also use these tools with *Gradle*.

Checkstyle This tool checks whether you are adhering to the style guide you can specify. hold. We provide you with a code style config that you can adapt or leave unchanged. can use.

PMD is a static analysis tool, which means that it does not execute your code. to find errors. PMD examines the source code for certain patterns, which are known to cause errors. Examples of such patterns are unreachable code (you probably forgot to call a method) or the comparison of two strings with == (in Java, strings with `equals()` must be set to equality can be checked). The report can be opened with a web browser and `lists` all problem areas and links to the exact problem descriptions.

You will notice that *PMD* does not allow `System.(out|err).println` to use. Such console outputs should only be used during debugging. the. Instead, you can use the *SLF4J* logging framework[5] , which is more powerful. is configurable.

SpotBugs SpotBugs is also a static analysis tool, but it does not work directly on the source code, but on the compiled Java bytecode. This means that it can find problematic code positions that are overlooked by *PMD*, e.g. when be- certain expressions always have the value `zero`. There is also an error report here, in which the problem is pointed out.

[5] `https://www.slf4j.org/manual.html`

5. Tests

In the group phase, we expect you to carry out system, unit and integration tests. The an- requirements that are placed on your application can be found in the specifications from from the previous chapters.

During the implementation phase, try to achieve the highest possible statement and branch coverage. This means that your code should contain as many statements and branches as possible. should be covered by the tests. The coverage can be calculated, for example, with *JaCoCo*[1] calculated

, and *IntelliJ* already has integrated tools for this.

5.1 Unit tests

For the unit tests, use *JUnit 5*[2] and *Mockito*[3]. Various tutorials for the use of You can find a description of these frameworks on the Internet at[4] . We will show you at the beginning of the implementa

`provide` a sample test for the assessment phase.

The unit tests can be executed with `./gradlew test` (or `gradlew.bat test`). den.

Implementation note

Unit tests belong in the given directory `src/test/java` in the project folder.

5.2 Integration tests

It is your task to also write integration tests to ensure that the modules work together. test. In integration tests, individual methods are accessed in a similar way to unit tests. accessed. In principle, you can write and execute these tests in the same way as unit tests.

12
https://www.jacoco.org/https://junit.org[3] https://site.mockito.org
[4]https://junit.org/junit5/docs/current/user-guide/

5.3 System tests

System tests must also be written that adequately prepare the game server for the test the entire functionality. This time, the framework offers the option for sub different system tests.

We provide you with a framework that you can use at the start of the implementation phase. together with the *CommLib*. The framework takes over the communication with the server.

Your system tests are also executed on the reference implementation, so that you can determine whether your assumption about the server's behavior is correct.

Please note that your system tests can also be run on faulty game servers (mutants). , i.e. faulty implementations must be found by your tests. sen.

Implementation note

System tests belong in the given directory `src/systemtest/java` in the project folder.

A. Appendix

A.1. skills

Table 1: Passive skills

`WOUND`

A survivor with this ability may receive a different type of injury when moving (if the infection cube results in an injury). Which type of injury (`wound`, `frostbite`, `bite`) is converted into which other injury is freely selected in the configuration file (`before`/`after`).

`NO-INFECTION`

A survivor with this ability can perform an attack without being injured (= without rolling the infection die after the attack).

`SEARCH`

A survivor with this ability can draw and keep more than one card when searching a location. The ability applies to a specific location, which is specified in the configuration file (`locationId`). In addition, the number of cards (`numCards`) that can be drawn can vary. The ability is automatically activated when the survivor searches the specified location and can be used as often as specified in the configuration file (`maxActivations`). After the maximum number of activations of the ability has been reached, the usual rules apply for further searches of locations. If there are fewer cards left at the location than could be drawn with this ability, all remaining cards are simply drawn.

Table 1: Passive skills

TRASH

A survivor with this ability can dispose of a different number of cards than usual when disposing of waste (and therefore only once per round). The exact number of cards is specified in the configuration file (`numCards`).

Table 2: Active skills

HEAL

With this ability, a survivor can heal a wound or frostbite of another survivor in the same location. Frostbite is healed first and then wounds. The survivor to be healed is `specified` by the `characterId` as the `target`. This ability can be used a maximum of 1 time per round.

FEED

With this ability, a survivor can add a certain amount of food tokens to the food supply. The exact amount of food tokens that are added is freely configurable and must be specified in the configuration file (`numFood`). This ability can be used a maximum of 1 time per round.

BARRICADE

A survivor with this ability can create a barricade at an entrance to the location where they are standing. How often this ability can be used in a round is specified in the configuration file (`maxActivations`). It also specifies how many spaces of an entrance are `barricaded` by using the ability (`numBarricades`). In this case, the `target` is the `entrance` of the entrance to be barricaded.

Table 2: Active skills

KILL

A survivor with this ability can kill zombies more easily than other survivors. The configuration file specifies:

- at which location this capability is active (`locationId`),
- whether an action cube is required and if so, how high the number of points must be (`theValue`),
- how many zombies are killed by activating the ability (`numZombies`),
- whether the infection die must be rolled (`infectionDie`),
- whether children must be in the colony to activate the ability (`children`),
- and how often the ability can be activated in a round can be activated (`maxActivations`).

1 **A.2. cards**

Table 3: Cards without destination

FOOD

This card can be used to add food to the food supply. Each `FOOD` card indicates a quantity of food that can be added to the food supply in the form of *food markers* by playing the card. This card can also be contributed to a crisis that requires `FOOD`. When contributing to a crisis, each card only counts as one card, regardless of the amount of food it would have added to the food supply when played.

FUEL

With this card, survivors can perform a movement without having to roll the infection die. This card can also be used to contribute to a crisis that requires `FUEL`. If a player has more than one `FUEL` card, the `FUEL card` with the lowest `id` is used.

Table 3: Cards without destination

STUFF

With this card, the player can re-roll their lowest action die. This card can also be contributed to a crisis that requires `STUFF`.

Table 4: Cards with `target`

MEDICINE

A card heals a wound or frostbite of a survivor who is specified as a `target`. Since frostbite is the more dangerous wound, frostbite is healed first when this card is used. This card can also be contributed to a crisis that requires `MEDICINE`.

LOCK

With this card, a survivor can barricade exactly one entrance of their current location. The `entrance` at which a barricade is to be set up must be specified as the `target`. This card can also be contributed to a crisis that requires `STUFF`.

SCISSORS

With this card, a survivor can kill a zombie at an entrance of his location. The `entrance` at which the zombie is to be killed must be specified as the `target`. If there is no zombie at this entrance, the card will not be played. This card can also be contributed to a crisis that requires `STUFF`. Playing the `SCISSORS` card does NOT count as an attack, so no infection die is rolled.

1 A.3 Commands and events

Table 5: Administrative Commands

Administrative Commands
`Register(String playerName)` The register `command` is sent by a client to log in to the game. He also enters his name. If a registered player attempts to register, a `CommandFailed` is sent. Can be sent without an `ActNow()` being received first.
`StartGame()` With this command, the player informs the server that the game should start. This command can be used to start the game before the maximum number of players has been reached. If this command is sent when the game has already started, a `CommandFailed` is sent. Can be sent without an `ActNow` being received first.
`SelectCharacters(int characterId0, int characterId1)` With this command, the player tells the server which characters he wants to start the game with. This command is expected as a response to the `characters` command.

Table 6: In-game commands

In-Game Commands
`Leave()` With this command, a player informs the server that he wants to leave the game. The player is then removed from the game and all his characters are killed (starting with the player with the highest social status and then continuing in descending order). ~~An `ActNow()` does not necessarily have to be received beforehand.~~ A `Leave command` may only be sent when it is the current player's turn. Otherwise the command will fail.

Table 6: In-game commands

In-Game Commands

`UseAbility(int characterId)`

With this command, the player informs the server that she wants to use the ability of the character with the specified `characterId`. If this is not possible, the command fails and a `CommandFailed` is sent.

`UseAbility(int characterId, int target)`

With this command, the player informs the server that she wants to use the ability of the character with the specified `characterId`. The target of the ability is specified with the `target`. If this is not possible, or if the character and the target are not in the same location, the command fails and a `CommandFailed` is sent.

`Barricade(int characterId, int entrance)`

With this command, the player informs the server that he wants to erect a barricade at the specified entrance with the specified character. If this is not possible, the command fails and a `CommandFailed` is sent.

`EndTurn()`

With this command, the player informs the server that she does not wish to perform any more actions in this round. This ends her turn. If the player sending the command is not the active player, a `CommandFailed` is sent and the game continues as normal.

`Move(int characterId, int locationId, boolean fuel)`

With this command, the player informs the server that he wants to move the specified character to the specified location. The value of `fuel` indicates whether the player plays a fuel card, otherwise the infection die is rolled for the character after it has been moved to its new location. Each character can change location a maximum of once per round. If the command cannot be executed (e.g. because there is no space at the destination location, the character has already moved, or because the player does not have enough fuel), a `CommandFailed` is sent.

Table 6: In-game commands

In-Game Commands

`UseCard(int cardId)`

With this command, the player informs the server that she wants to play the specified card. If she does not have the card or it is not her turn, a `CommandFailed` is sent.

`UseCard(int cardId, int characterId, int target)`

With this command, the player informs the server that she wants to play the specified card with the specified character. A target is also specified which should receive the effect of the card. This version of the command is used to heal characters and to barricade themselves with a corresponding card. If the character and the target are not in the same location, the command will fail. The command also fails if the player does not have the specified card or if it is not her turn. In both cases, a `CommandFailed` is sent.

`ContributeCard(int cardId)`

With this command, the player informs the server that he wants to contribute the specified card to the crisis. If he does not have the card or it is not his turn, a `CommandFailed` is sent.

`CleanWaste(int characterId)`

With this command, the player informs the server that she wants to dispose of the garbage with the specified character. If the character is not in the colony, or the player has no more action cubes left, or there is no garbage available, the command fails and a `CommandFailed` is sent.

Table 6: In-game commands

In-Game Commands

`Attack(int characterId, int entrance)`

With this command, the player informs the server that he wants to attack a zombie at the specified entrance with the specified character. After the zombie has been killed, an infection die is rolled for the attacking character. If the player cannot attack with the specified character because they do not have an action die with a number ≥ the attack value of the character, there is no zombie at the specified entrance or it is not the player's turn, the command fails and a `CommandFailed` is sent.

`Search(int characterId)`

With this command, the player informs the server that she wants to search the character's location with the specified character. If the player is unable to search the location with the specified character because she has no action cube left with a number ≥ the search value of the character, there are no cards in the location's card deck, the location has no card deck (except for the colony) or it is not the player's turn, the command fails and a `CommandFailed` is sent.

Table 7: Events Individual

events (sent to exactly one client)

`Config(String config)`

With this event, the server sends the game configuration file to the client as a string. It is sent to the client directly after a successful `registration` command.

Table 7: Events Individual

events (sent to exactly one client)

`ActNow()`

With this event, the server sends the client that it is its turn and the server is expecting a new command. The event is sent every time a command is expected. A player who no longer has any survivors cannot make any moves and will therefore no longer receive this event.

`Characters(int characterId0, int characterId1, int characterId2, int characterId3)`

With this event, the server sends the client a selection of characters from which the player must choose two to start the game with.

`CommandFailed(String message)`

With this event, the server informs the client that the last sent command has failed. The `message` specifies the reason for the failure. *Note: We do not explicitly test the content of the message. However, it must not be empty.*

Table 8: Events Broadcast

events (are sent to all clients.)

`RegistrationAborted()`

This event notifies the client that the server has received an invalid command during the registration phase and is therefore terminating. An invalid command during the registration phase is any command that is not a registration `command` or StartGame `command`.

`Player(int player, String playerName)`

After the registration phase, the server sends this event to all clients to inform them about the players participating in the game. The events are sent in ascending order of the `players`.

Table 8: Events

Broadcast events (are sent to all clients.)

`CharacterSpawned(int player, int characterId)`

With this event, the server informs the client that a new character has been spawned for the specified player. If several characters spawn at the same time, the events are sent in ascending order of `characterId`.

`ZombieSpawned(int locationId, int entrance)`

With this event, the server informs the client that a zombie has spawned at the specified location and the specified entrance. The event is sent immediately after the zombie spawns.

`ChildSpawned()`

With this event, the server informs the client that a child has been spawned.

`CardDrawn(int player, int cardId)`

With this event, the server informs the client that the specified player has drawn the card with the specified `cardId`. This event is also sent at the beginning when the cards are distributed at the start of the game.

`GameEnd(boolean win)`

With this event, the server informs the client that the game has ended. The `win` parameter indicates whether the game has been won or lost.

`GameStarted()`

With this event, the server informs the client that the game has started.

`NextRound(int round)`

With this event, the server informs the client that a new round has started. `round` specifies the round number. The first round has the number 1.

Table 8: Events

Broadcast events (are sent to all clients.)

`Crisis(int crisisId)`

With this event, the server informs the client that the crisis with the ID `crisisId` must be fulfilled in this round. The crisis `event` is the first event after the `NextRound event`.

`DieRolled(int player, int value)`

With this event, the server informs the client that a die has been rolled for the specified player. `value` stands for the value of the rolled die. DieRolled `events` are sent after the Crisis `event` in ascending order of player ID.

`AbilityUsed(int characterId)`

With this event, the server informs the client that the specified character has actively used its ability. This event is only sent if the ability has no target.

`AbilityUsed(int characterId, int target)`

With this event, the server informs the client that the specified character has actively used its ability. This event is sent when the ability used requires a target to which it is applied.

`Barricaded(int characterId, int locationId, int entrance)`

With this event, the server informs the client that the specified entrance at the specified location has been barricaded by the character with the specified `characterId`.

`Moved(int characterId, int locationId, boolean fuel)`

With this event, the server informs the client that the specified character has moved to the location with the corresponding entrance. `fuel` indicates whether gasoline was used for the movement.

Table 8: Events

Broadcast events (are sent to all clients.)

`Frostbitten(int characterId)`
With this event, the server informs the client that the specified character has suffered frostbite. This event can be sent directly after a movement without gas or an attack.

`Wounded(int characterId)`
With this event, the server informs the client that the specified character has suffered a wound. This event can be sent directly after a movement without gas or an attack.

`Bitten(int characterId)`
With this event, the server informs the client that the specified character has been bitten and therefore dies. In this case, an outbreak occurs, which must be dealt with before the game can be continued. This event can be sent directly after a movement without gas or an attack. It is also sent if a survivor dies when an infection spreads.

`Request()`
This event is sent if a child is bitten and dies when an infection spreads. An outbreak occurs that must be dealt with before the game can continue.

`Contributed(int player, int cardId)`
With this event, the server informs the client that the player has contributed the specified card to the crisis. This event is sent after the `ContributeCard` command.

`CardUsed(int cardId)`
With this event, the server informs the client that the specified card has been used. This event is sent after the UseCard `command`.

Table 8: Events

Broadcast events (are sent to all clients.)

`CardUsed(int cardId, int characterId, int target)`

With this event, the server informs the client that the specified card has been used by the character with the specified `characterId` on the specified target. This event is sent after the UseCard `command`.

`FoodChanged(int amount, FoodChange reason)`

With this event, the server informs the client that the `amount` of food available in the colony has changed by an `amount`. The `reason` enum specifies why the amount of food has changed.

`ColonyPhaseStarted()`

With this event, the server informs the client that the colony phase has begun. This event is sent after the last player in the round has finished their turn.

`StarvationTokenAdded()`

With this event, the server informs the client that a hunger marker has been added to the food supply. This event is sent instead of the FoodChanged event if there is not enough food in the food supply.

`MoralChanged(int amount, MoralChange reason)`

With this event, the server informs the client that the morale has changed by an `amount`. The `reason` enum indicates why the morale has changed. If the morale has dropped to 0 as a result, the game is ended.

`SurvivorKilled(int characterId)`

With this event, the server informs the client that the specified character has been attacked and killed by a zombie. This event is only sent if a zombie should spawn at an entrance but there was no more room there.

Table 8: Events

Broadcast events (are sent to all clients.)

`ChildKilled()`

With this event, the server informs the client that a child has been attacked and killed by a zombie. This event is only sent if a zombie spawns at an entrance but there was no more room.

`BarricadeDestroyed(int locationId, int entrance)`

With this event, the server informs the client that a barricade has been destroyed at the specified location and entrance.

`WasteChanged(int amount)`

With this event, the server informs the client that the amount of garbage has changed. The new amount of garbage is `amount`. This event is always sent when a card is played (which is added to the waste pile) and when waste has been disposed of.

`ZombieKilled(int characterId, int locationId, int entrance)`

With this event, the server informs the client that the specified character has attacked and killed a zombie at the specified location and entrance.

`Searched(int characterId, int locationId)`

With this event, the server informs the client that the specified location of the specified character has been searched. This event is followed by `CardDrawn` events.

`Left(int player)`

With this event, the server informs the client that the specified player has left the game.