Foundations of Discrete-Event Simulation

Theoretical and practical aspects using the example of card games

Pascal Crenzin

21.08.2020

Abstract

This report introduces discrete-event simulations and classifies the purpose of discrete-event simulation based on its characteristics. Discrete-event simulations are stochastic, dynamic and based on discrete events. Every simulation is based on a model. A generic model of discrete-event simulation is the queuing of arriving events, followed by the processing of the events. Every event changes the system state. An application of using discrete-event simulation to analyze and balance card games is demonstrated with the example of *UNO*. The presented approach realizes the main components and the simulation paradigm in a custom implementation of a framework for the simulation of card games. The simulation of *UNO* analyzes the duration of *UNO* games depending on the presence of special cards, number of players and number of initial hand cards. The result is, that a *UNO* game takes normally 10 to 30 rounds to finish.

Contents

1 Introduction **3**

2 Foundations **3**
- 2.1 Terminology . . . 4
  - 2.1.1 System . . . 4
  - 2.1.2 Event . . . 4
  - 2.1.3 Model . . . 4
- 2.2 Classification of Simulations . . . 5
  - 2.2.1 Static & Dynamic . . . 5
  - 2.2.2 Continuous & Discrete . . . 5
  - 2.2.3 Fidelity, Resolution & Scale . . . 5
  - 2.2.4 Games, Simulation Games & Simulators . . . 6
- 2.3 Discrete-Event Simulation . . . 7
  - 2.3.1 Components . . . 8
  - 2.3.2 Trace-Driven Discrete-Event Simulation . . . 8
  - 2.3.3 Next-Event Simulation . . . 8
  - 2.3.4 Time-Advance Mechanism . . . 9
- 2.4 Modelling & Simulation Cycle . . . 9

3 Example: Model Card Games as Discrete-Event Simulation **10**
- 3.1 Data Structures . . . 11
- 3.2 Simulation Logic . . . 11
- 3.3 Implementation of Uno . . . 11

4 Experiments and Evaluation **13**
- 4.1 Impact of Special Cards . . . 13
- 4.2 Impact of Game Parameters . . . 14

5 Discussion **15**
- 5.1 Verification and Validation of the *UNO* simulation . . . 15
- 5.2 Insights from the simulation . . . 16
- 5.3 Extensebility of the framework . . . 17

6 Conclusion **17**

1 Introduction

Simulations are a problem-based discipline that allows for repeated testing of a hypothesis [1]. This statement covers two important aspects of simulations in general. The first key element in this statement is *"problem-based"*. This means a simulation always addresses a question. The second key element is *"repeated testing"*. This means a simulation is intended to run multiple times.

The field of simulations covers a wide range of applications. Simulations, in particular, discrete-event simulations can be used to analyze the complex behavior of systems. Examples are stock inventory monitoring, optimizing supply chains [8], health care systems [5, 7] or games.

In order to simulate a real-world system, a model must be created. All observations are derived from the model. Understanding the foundations of how to build such models is the base for data-driven insights. It enables people to build a precise abstraction of reality, having a methodology to master complexity, understand required techniques and tools, and prove its validity by solid mathematical foundations [1].

This report sets its focus on the simulation of board games, especially card games.

Board games are discussed by researchers in multiple branches of science like social science, computer science, mathematics, or psychology. Board games are either object of study or models for developing analogies [3]. An example of an analogy is by Janssen who used the analysis of shredded card games to understand the evolution of institutional arrangements [6]. If the board game becomes the object of research a simulation can be used to train a computer on how to play a game. A popular example is the development of AlphaGo [15]. Another approach is to use simulations during the game design phase which is more challenging than gameplay. This can be done by tuning the game parameters or by modifying the rules [4]. To achieve the right balance of fun, time, or easiness of the game, parameters and rules must be chosen carefully. For example the number of cards at the beginning of a game or the number of jokers in the game. Using simulations makes it simple to collect quantitative data about little details that can have a huge impact on the game.

This report continues with the foundations of discrete-event simulation, followed by an example simulation of the card game *UNO*. Then, the design of *UNO* will be analyzed. After that I discuss the model of *UNO* and finally, I draw a conclusion.

2 Foundations

The general approach to simulate something is to build a model as an abstraction of a system in the real world and then modify and observe the model according

to the hypothesis that needs to be tested. The last step is to draw a conclusion out of the observations.

2.1 Terminology

System and model are widely used. To clarify their meaning in this report I explain their meaning in the context of simulations.

2.1.1 System

A model is intended to represent a system in the real world. Therefore the abstract term *system* must be defined. In the scope of this report, a system is a collection of different elements that influence the simulation. This includes among other people, hardware, software, institutions, documents, and environmental factors. A system has a state which is a representation of the current situation. A system can be physically present in the real world, but it could also be a plan or concept for something that does not yet exist.

2.1.2 Event

An event is an occurance that impacts the the state of the system [9, page 187]. An event could may change the state of the system, but it doesn't have to necesserially. By definition, the state of the system can only change at an event time. Each event has an associated event type.

2.1.3 Model

The concept *model* is a physical, mathematical, or otherwise logical representation of a system. Models serve as representations of events and things. The events can be real, e.g. from an observation of the real world, or contrived. A model represents a system. While a system could be dangerous to observe or even don't exist, a model could be used to study the evolution of a system. When investigating a system, a quantitative assessment is derived. This may include how the system performs with various inputs and in different environments. Different scenarios (occurrence of specific events) can be investigated. It is important to get a quantitative evaluation of the performance of the system. The performance could cover multiple aspects, which depend on the purpose of the model. A model exists at three levels of abstraction: conceptual, specification, and computational [9, chapter 1]. A conceptual model abstracts the general concept while a specification model contains concrete information. A computational model refers to an implemented program of a model, which means that the computational model is nothing else than a simulation. In this report computational model and simulation are equivalent. The classifications for simulations are also classifications of models.

In the following, I will go into more detail regarding different types of simulations, followed by a detailed explanation of discrete-event simulation, and lastly,

introduce the cycle of modeling & simulation.

2.2 Classification of Simulations

Simulations in general can be classified depending on multiple aspects. In this section, I will introduce a few approaches to classify simulations.

2.2.1 Static & Dynamic

A system model is static or dynamic. A static model is one in which time is not a significant variable. A dynamic model evolves over time [9, chapter 1].

Static models can answer the question of how much money gets distributed if N people play the lottery next week. The result is completely independent of the timing of the players. To solve such questions, there is the Monte Carlo simulation, that relies on repeated random sampling to obtain numerical results. Another use case for Monte Carlo simulations is finding the area under a curve [13]. In the context of board games, a Monte Carlo simulation could be used to determine how frequently fields are visited on a game board.

If the system evolves over time it is a so-called dynamic model. A dynamic model would emerge if the lottery example gets extended by players that play week for week. Their spendings on the lottery could depend on their last result.

2.2.2 Continuous & Discrete

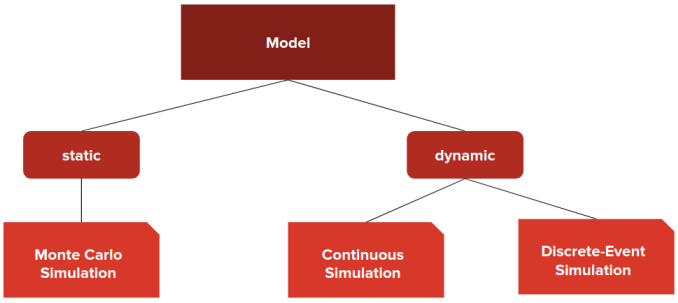
A dynamic system model is continuous or discrete. Continuous simulations are based on a continuous function of time. Examples of continuous simulations can be found in classical mechanics. Examples are oscillating pendulums or a block sliding on an inclined plane. In each of these cases, the motion is characterized by one or more differential equations that model the continuous-time evolution of the system. In contrast, discrete simulations are based on events that occur at discrete points in time. Only events can lead to a change in the system. Examples are inventory systems or board games. An inventory system changes its state every time a new order arrives or new products arrive. These changes are not continuous, but they occur at a certain point in time. The next section focuses on discrete-event simulation in more detail.

2.2.3 Fidelity, Resolution & Scale

There are three primary attributes that can be mapped to models or simulations. These are fidelity, resolution, and scale [1].

Fidelity indicates how close a simulation matches reality. A model that closely behaves like a real system has high fidelity. High fidelity means that every aspect of the system must be covered. Normally models aim to describe only the parts that are necessary for the initial hypothesis. Aiming for

Figure 1: Chosing the right simulation paradigm depending on the model requirements (adopted from [9, page 3])



a high fidelity would mean a great effort. It should only be done if the hypothesis requires a high fidelity of the simulation.

Resolution describes the level of details. The more detail included in the simulation, the higher the resolution. To illustrate this, a model of a plantation could be broken down into models of all single plants. Modeling all plants would achieve a higher resolution. The target resolution depends on the intention of the model. Referring to the plantation example, a model that should describe harvesting around the globe, would better use a low resolution, namely models of plantations, while a model to optimize the crop would need a higher resolution.

Scale indicates the overall size of the simulation. A larger system has a higher scale. The scale is a fairly subjective metric. It always depends on the relations. One example is simulating a fabric compared to simulating a single device in a production line.

2.2.4 Games, Simulation Games & Simulators

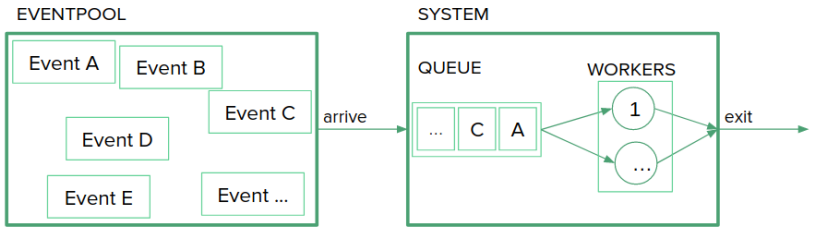
Nearly every system can be simulated. In this report, I would like to have a deeper look into the field of games. Games play a popular role in the entertainment industry. There is a wide range of games that aim to simulate a system in an interactive way. These can be distinguished whether they aim for fun or for training. Games, simulation games, and simulators can be distinguished based on various characteristics [12]. In this scope *games* mainly refers to computer games. But the characteristics also apply for board games. All (computer) games have a simulation part somehow, but one characteristic is that simulation games do not always involve a specific goal-oriented activity within the context of the game. Games and simulation games aim for fun and entertainment. They provide challenges and have typical game patterns. The world can be imaginative or fictitious. Simulators are designed for training. Training simulators only provide a real-world environment. They aim for skill-development rather

than fun. Challenges in training simulators are challenges with equivalents in the real-world.

2.3 Discrete-Event Simulation

There are certain characteristics that define a discrete event simulation. First of all, there must be a stochastic component. This means the state of our model depends on probabilities. At least one variable that defines the current state of the system must be random. Secondly, the model is dynamic. This means, there is a change over time. Thirdly, the changes of the model depend on discrete-events. These discrete-events can occur randomly at discrete time instances.

Figure 2: Visualization of a discrete-event simulation system



To visualize a discrete-event simulation, it can be seen as a queuing system. There are certain events that can occur and a system contains queue and workers that handle the events. Depending on the scenario events are also called jobs or customers. Workers also have synonyms like server. The queue is sometimes renamed to *event list* or *calendar*. And the system is also called *service node* in other literature. Events arrive at the system at random points in time and each event could take a different amount of time to get processed by a worker before the event it can exit the system. The systems operate as follows: If a new event arrives, it will be moved into the queue. If the queue is not empty and at least one worker is not busy, one event is taken from the queue and gets processed.

From this behavior, two important terms regarding the timing can be derived. Delay. The delay is the time that an event remains in the queue. the delay is greater or equals zero. Wait. The wait for an event is the duration between arrival and exit. In other words, the wait is the delay plus the processing time of the worker.

Reducing delay and wait is a common optimization goal that discrete-event simulation can be used for. This could be achieved by adding more workers or changing the way how events are selected from the queue. A common technique is FIFO (first in, first out) that always takes the event with the greatest delay. Other techniques try to optimize the delay by prioritizing events with a small processing time (also known as *shortest job first*).

2.3.1 Components

There are some main components that a discrete event simulation requires.

The System State. This captures all variables that characterize the simulation.

The Simulation Clock. This is a tool that tracks the elapsed time. Different approaches to increment the simulation clock are explained in *Time-Advance Mechanism*.

Next - Event List. This is a queue for the upcoming events.

Statistical Counter or Accumulator. This is a tool that records the evolution of the system state.

2.3.2 Trace-Driven Discrete-Event Simulation

The events in a discrete-event simulation can be generated or prerecorded. If the simulation relies on prerecorded data, it is a so-called trace-driven discrete-event simulation. By definition, a trace-driven simulation relies on input data from an external source to generate realistic events. [9, chapter 1] The advantage of trace-driven discrete-event simulations is that no data needs to be generated and the data source reveals realistic data. On the other hand, the reliance on external data limits the capabilities to simulate new scenarios. A common question is "What if X" happens. And if X not part of the recorded data it is not possible to find out.

2.3.3 Next-Event Simulation

The concept of next-event simulation extends the basic concept of discrete-event simulations. To construct a next-event simulation model, all events must be assignable to an event type. An algorithm decides how the system state changes depending on these event types [9, chapter 5]. The algorithm for a next-event simulation is based on four steps.

Initialization: In this The simulation clock is initialized (usually to zero) and the first events get queued.

´Processing the current event. The queue is scanned to determine the next event. Then the event gets processed the event and the simulation clock advances.

Schedule new events. New events (if any) that may be spawned by the current event are placed in the queue.

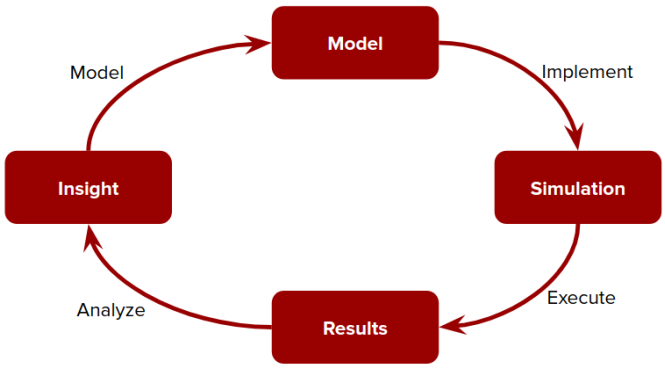
Terminate. The process of advancing the simulation clock and processing events and scheduling new events continues until some terminal condition is satisfied. The terminal condition can be a specific event that only occurs once. Another terminal condition could be reaching a certain time on the simulation clock.

2.3.4 Time-Advance Mechanism

The progress of time in a simulation is a very essential aspect. In discrete-event simulations, there are two well-known approaches: Fixed-increment advance and next-event time advance. The fixed-increment advance initializes at time zero and advances at fixed time increments. The disadvantage is that events occur not in sync with the increments. This means the arrival time and exit time is probably not exact. Another characteristic of the fixed-increment advance is that the size of the time increment is subjective. But choosing the increment interval has implications for the performance measures of the system. Large increments lead to imprecise timing of the events, while small increments could lead to a situation where no changes happen for many increments. In contrast, the next-event time-advance approach is more versatile. It initializes the simulation clock at zero and then progresses to the next upcoming event. The incrementation step is not fixed, but always in sync with the arrivals and exits of all events. This solves the disadvantages of the fixed-increment advance.

2.4 Modelling & Simulation Cycle

Figure 3: Process of generating and evaluating simulations (adopted from [16, page 7])



Simulations are based on models and simulations are designed to answer a hypothesis. This means simulations must also be understood as a piece a larger process. It starts with modeling. Modeling includes theories, information, algorithms of the topic. A decision must be taken which type of model and with paradigms are used. A supporting tool for that is the decision tree in 3. The result of the first phase is a model. The model gets implemented in the next step. There is dedicated software for simulation purposes [10]. The result of this step is an executable program that implements the model. During the execution phase, the simulation program generates data. For the execution, the processing environment must be taken into account. Complex simulations may need more computation power than a normal home computer could provide.

One way to achieve a better performance is to run a distributed simulation [11]. The output data must be transformed to get the desired performance insights. If the model contains variability and uncertainty, then techniques from probability and statistics will likely be required for the analysis. All further analysis must happen regarding the initial hypothesis. A model must be verified and validated. This means to check whether the model was built correctly and whether the model is appropriate compared to the real world. It is very likely that the results reveal that the model must be adjusted or extended. This means to reiterate through this cycle. It is a good practice to repeat the cycle as often as needed.

I will go through this cycle with an example of a discrete-event simulation use case in the next chapter.

3 Example: Model Card Games as Discrete-Event Simulation

I focus on card games as examples. The focus on card games is due to the wide range of different types of board games. Board games, in general, are too diverse in terms of materials (game board, cards, meeples) and rules. In contrast to that, most card games share the same game elements. To be even more specific, the implementation mainly addresses shedding games, like *UNO*, *The Great Dalmuti* and *Phase 10* [2]. They have at least one discard pile, where all players contribute to. They have a hand for each player and optionally shared or not shared open played cards. Additionally, they may have a draw pile. Cards consist of color or a symbol, a number, and maybe a special rule is attached to them. The game rules of card games are that all players take their turn one after the other. There is no concurrency. This excludes some card games that aim for competing in reaction time, but this is not the scope of this implementation.

The decision to build a framework from scratch was derived from the specific conditions that card games have. In the next section, I will explain my implementation, followed by a concrete implementation of the card game *UNO*.

The objective of the implementation is to build a framework that

- maps all main components of discrete-event simulation to the context of card games
- uses an object-oriented approach for intuitive readability
- is generic enough to cover common card games
- keeps the implementation effort low for new player behavior and new games

The code is written in Python and published on GitHub at `https://github.com/paszin/card-game-simulation`.

3.1 Data Structures

All piles are implemented with the data structure of stacks. In computer science, a stack is a collection of data in which data can be inserted or deleted only at the top of the stack. [14, page 86] Stacks implement the functions of push (add data at the top) and pop (delete data at the top).

Cards are the main data objects. They have the read only attributes *color* and *number* along with one writeable attribute *valid*. The Attribute *valid* is used to save the information if the effect of the card was already applied.

3.2 Simulation Logic

There are some main components that a discrete event simulation requires. This follows up on the components introduced in 2.3.1. I will explain how I implemented the components in my card game framework. The overall execution logic of the simulations follows the concepts of next-event simulation in 2.3.3.

System State This captures all variables that characterize the simulation. In my case, the system state consists of the number of the current turn, the discard pile, the draw pile, and the hand of the players.

Simulation Clock This is a tool that tracks the elapsed time. For card games, the simulation clock is a counter of the turns. It gets incremented every time a player starts its turn. The initial value is zero. The simulation clock uses a fixed-increment advance. The turn-counter always increments by one and no interruptions by other players are possible. As interruptions are uncommon for card games, the simulation clock acts similar to a next-event time-advance, because the turn counter only increments, if the next player has a turn. The only accepted interruptions are triggered by the current player or other game objects like the draw pile. An exception handler is able to handle the exception. One exception is the end of the game, which is handled by breaking out of the simulation loop.

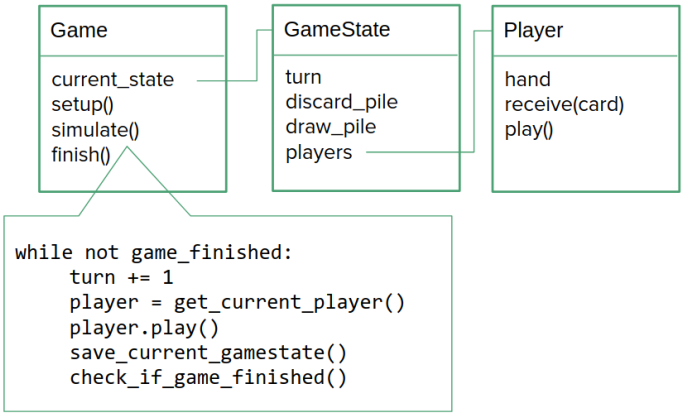
Next-Event List This is a queue of the upcoming events. For games, events are equal to making a turn. In my implementation, the next event is defined by a function that returns the next player. The choice of which player comes next depends on the current system state.

Statistical Counter or Accumulator This is a tool that records the evolution of the system state. My implementation saves a copy of the current system state after each turn. From this list of game states, a specific development of one variable can be derived.

3.3 Implementation of Uno

As a first example, I will introduce *UNO*. In *UNO*, every player gets seven cards. The rest of the cards are placed face down on a draw pile. Next to the pile is a discard pile. The top card should be placed on the discard pile and the game

Figure 4: Overview of objects, its methods and pseudo-code of simulation function



begins. The players make their turns one after each other. A player could either put a card on the discard pile if it matches the last card or draws a card until one player has no more cards. Some special cards could have an effect on how the game evolves. Janssen derived eleven rule variations for *UNO* by comparing it to *Crazy Eights* for his studies [6].

At first, I implement a simplified version of *UNO*. My *simplified UNO* drops all special cards. The idea of the modification is that this is still a good abstraction to simulate the flow of *UNO*. Furthermore, observations of this version of the game can be used as an indicator of whether there is a need for special cards that make the game more difficult or easier. Special cards could be added sequentially to measure the impact of a new card. Without special cards, the card deck consists of the cards one to nine, two times in four different colors.

To extend the framework I have to implement derived classes from the *Game*-class and the *Player*-class. The new *Game*-class must implement the *setup*-method and the new *Player*-class must implement the *play*-method.

The implementation of the *setup*-method must start with the generation of the card deck. This is done by nested for-loops over the colors and numbers of the deck. The last step is setting the game state. Inbetween the *setup*-method follows the game instruction of *UNO*. To extend this, special cards can be added to the card deck.

The player behavior is implemented as choosing the first card that could be put on the discard pile. The first card means the card that holds the player for the longest time. This results in a deterministic turn.

The advanced version the player behavior follows these steps:

1. Check if the last card on the discard pile expects any special considerations.
2. Filter all cards that could be put on the discard pile.
3. Rate the cards. Special cards get a higher rating.
4. Chose the card with the lowest rating.

The intention of the player behavior is to avoid playing wild cards without the need to change the color and to keep other special cards for the end game. This player behavior is derived from personal observations. Different player types could be implemented that play special cards more aggressively. Also, interactive player types are possible. The *play*-method could depend on the input by a user.

4 Experiments and Evaluation

Dynamic simulations are a method to support a hypothesis. I will analyze how game parameters influence the duration and how special cards impact the duration. The aim is to either confirm the *UNO* rules or to suggest modifications in order to achieve a better duration. I define the duration of a game as the number of rounds. The number of rounds is the number of turns devided by the number of players. A assume a good duration for *UNO* is between 15 and 30 rounds. To convert the number of rounds to a duration in time, I assume that a turn takes five seconds on average. This value is derived from personal observations. 30 rounds of four players would result in a duration of ten minutes. This kind of question are useful for game developers when they have to balance the game. For instance a game like *UNO* should not take to long, because it is mainly designed for kids.

4.1 Impact of Special Cards

The card deck of *UNO* includes some special cards. Special cards named special because they trigger an effect that is an unfair advantage or disadvantage. The special cards are *wildcard* (chose new color), *skip* (next player is skipped), *plus2* (next player has to draw 2 cards or add a *plus2* as well), *wildcard plus4* (combination of wildcard and next player has to draw 4 cards). I analyzed multiple arrangements of special cards. Figure 5 shows a sorted scatter plot of the number of rounds. The plot reveals that adding the *plus2* extends the game, while adding the *wildcard* shortens the game. Adding all special cards rather extends the duration. With the aim of 15 to 30 rounds per game, the configuration with all special cards or only *plus2*-cards works the best as shown in the histogram in figure 6.

Figure 5: Simulation of 1000 Games of three players with seven hand cards and different configurations of the card deck

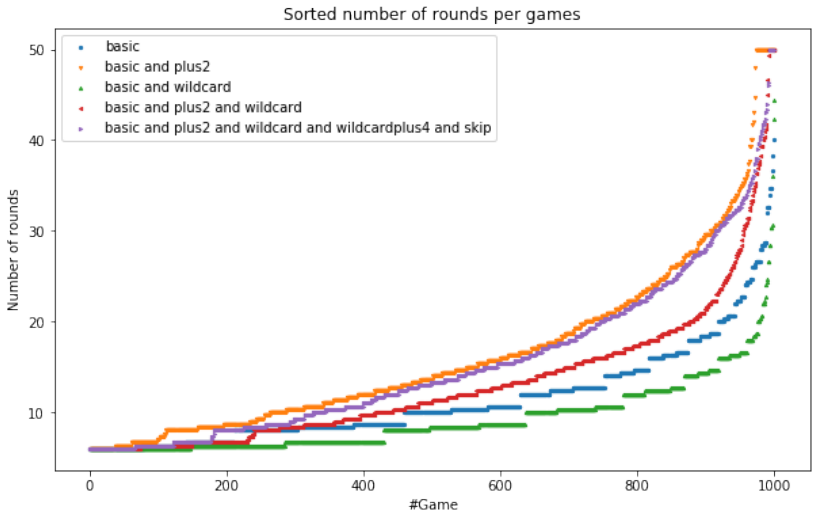
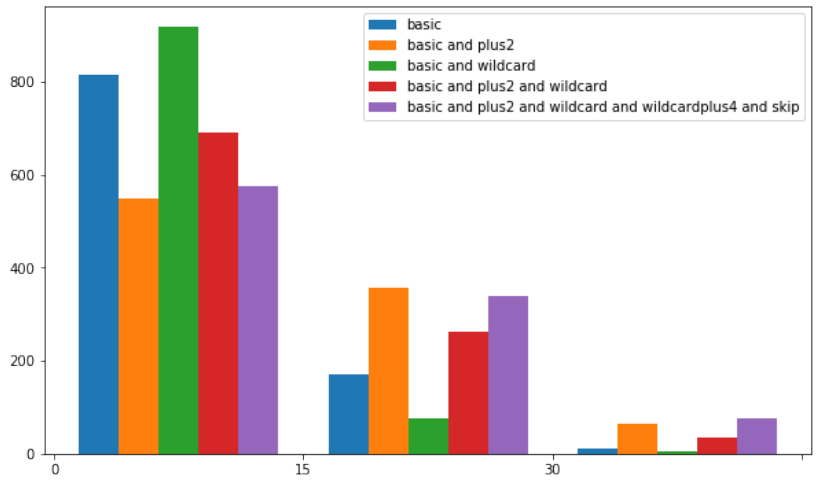


Figure 6: Histogram of simulation of 1000 Games of three players with seven hand cards and different configurations of the card deck. The bars indicate how many games ended after 0 to 15 rounds, 15 to 30 rounds, and more than 30 rounds.

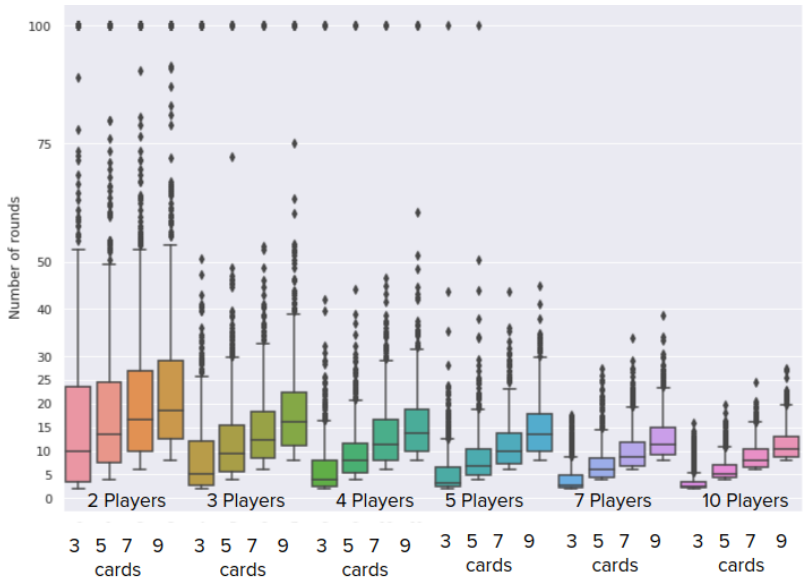


4.2 Impact of Game Parameters

The easiest parameter that can be tuned are number of hand cards according to the number of players. Therefore I simulated all combinations of two, three,

four, five, seven and 10 players with three, five, seven and nine hand cards. The result is shown as boxplots in figure 7. An increase of players result in a decrease of rounds per game and in a smaller span of the result. More cards lead to an increase of rounds. Outliers of 100 or more rounds exist for all configurations up to five players. 100 is the maximum because the simulation stops after 100 rounds.

Figure 7: Number of rounds to finish a game for a selection of two to ten players with three to 9 hand cards



5 Discussion

To summarize the experiments, *UNO* seems to be well-balanced. In order to make this statement, the results must be analyzed. But not just the results, also the model itself. I will do this in the next paragraph and then analyze the results. Lastly, I would like to give an outlook on extensibility for other card games.

5.1 Verification and Validation of the *UNO* simulation

The question regarding verification and validation is the question if the is model built correctly and was it the correct model? Verification can be simply proven for game simulations. A certain game state can only be transformed into a new game state according to the rules. The set of possible new game states is finite. For example, the discard pile shows a red five and the current player does not have any cards that match, then the player must have one more card in hand

after his turn. This can be simply observed by checking the game state. Coming up with scenarios, that cover edge cases is a common best practice. For *UNO* one of these edge cases could be multiple *plus2* cards on the discard pile and less than two cards on the draw pile.

Validation asks to make a statement regarding reality, representation, and requirements. [1, page 13] The statements are subjective, but they should be convincing.

Reality means how closely does the model matches reality. The players in the game are simulated with very simple and equal behavior. This won't be the case in the real world, but nonetheless, as *UNO* is a simple game with no sophisticated tactics the implemented behavior matches the player behavior of humans. Mistakes (e.g. forget to say *"uno"* when just one hand card remains) are not considered.

Representation means that some aspects are represented, some are not. I covered the basic concept. This doesn't include a graphic representation of the game.

Requirements means that different levels of fidelity are required for different applications. The requirement of the *UNO* model was to analyze the game duration. It was broken down into the number of rounds. Duration could also include thinking time, but this was not the intention of the model, thus no requirement.

Another relevant aspect of simulations is the number of overall simulations. Drawing conclusions after one simulation could lead to misinterpretations. Therefore I picked the scenario of three players with seven hand cards as a reference and analyzed the evolution of the mean and the standard deviation. After 1000 simulated games the mean and the standard deviation converge. The *means* remain between 12.17 and 12.31 rounds per game and the *standard deviation* remains between 5.96 and 6.2. I consider these variations as irrelevant for my question, thus I decided to run 1000 simulations per configuration.

5.2 Insights from the simulation

The frequency of special cards can be used to shorten or to extend the game. Wildcards shorten the game and *plus2* cards extend the game.

Regarding the ratio of the number of players to the number of cards, figure 7 reveals with an increase of players the number of rounds of the games decreases. Considering the fact that a round has more turns with more players, this effect is desirable. My target number of rounds of 15 to 30 rounds is not reached. Most games tend to be shorter. Maybe my assumption of a minimum of 15 rounds was too high. The limit of 30 rounds is only beaten by a few simulations. With seven hand cards, more than 95% of the games end before 30 rounds for three or more players. Scenarios with only two players seem to stand out. Some games end with more than 100 rounds. Also, the balancing who wins is unequal. In

simulations with two players and seven hand cards, the starting player wins 75% of the games. Based on this data, I would argue that seven cards per player independent from the number of players is a good rule, but for two players it needs some modification of the rules.

5.3 Extensebility of the framework

I consider two directions of extensions. The extensibility of the simulation with different player types is straight forward. In the initialization phase of the game, a list of player objects is passed to the game constructor. The filter functionality of playable cards can be implemented outside the player object. A demo of a player type based on human input is part of the code repository.

The extensibility regarding other card games may require some modifications. The current restriction is, that all players contribute to the same piles. But within these restrictions are many other games. An implementation that is part of the published code is an implementation of *The Game*. *The Game* is a cooperative game with the goal to put all hand cards on four discard piles. The final result is the sum of cards that are not on the discard pile. An interesting factor is the communication between the players. The communication is implemented by checking the hand cards of the other players. This allows also to define levels of allowed interactions. During the implementation, I noticed that sophisticated tactical moves are difficult to cover in an algorithm. A better approach for the future would be to learn the behavior of human players.

6 Conclusion

This report introduces discrete-event simulations and classifies the purpose of discrete-event simulation based on its characteristics. Discrete-event simulations are stochastic, dynamic, and based on discrete events. Every simulation is based on a model. A model can have a high or low fidelity, resolution, and scale. The introduced example of *UNO* has a low fidelity, low resolution, and high scale. A generic model of discrete-event simulation is the queuing of arriving events, followed by the processing of the events. Every event changes the system state. In the implementation of the card game framework, events are the turns of the individual players. The events occur at a fixed time. An event changes the system state. The evolution of the system state is logged. From these system states questions regarding the game evolution, can be answered. One shown example is the duration of *UNO* games. *UNO* which is played with seven hand cards, ends normally after 10-30 rounds. The simulation reveals that some special cards extend, some special cards shorten the game. The implemented framework can be used to run a further analysis of *UNO* or other games as well. I can confirm discrete-event simulation can be used to simulate card games.

The Institute of Industrial Engineers published a list of advantages and disadvantages of simulations, that I would like to follow up on [1]. One advantage

is that simulations enable us to make better decisions because every aspect can be tested. I can confirm this. I was able to look for games that take unexpectedly long and then visualize the game state to identify the bottleneck of the system. Another advantage is to understand how a system operates. While implementing rules, I noticed how precise rules must be written down. So as a game developer I would always try to implement a simulation just to confirm that all aspects are covered. But there are also disadvantages. Results need an interpretation. At some point, the is some uncertainty if the players are not implemented smart enough or if the game needs a modification.

All in all, discrete-event simulations are a powerful tool to reveal some hidden effects or to validate rules.

References

[1] Catherine M. Banks. Discrete-event simulation. In John A Sokolowski and Catherine M Banks, editors, *Modeling and simulation fundamentals: theoretical underpinnings and practical domains*, chapter 1, pages 1–24. John Wiley & Sons, 2010.

[2] Category:Shedding-type card games. Category:shedding-type card games — Wikipedia, the free encyclopedia, 2019. [Online; accessed 20-August-2020].

[3] Fernand Gobet, Jean Retschitzki, and Alex de Voogt. *Moves in mind: The psychology of board games*. Psychology Press, 2004.

[4] Vincent Hom and Joe Marks. Automatic design of balanced board games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 25–30, 2007.

[5] Sheldon H Jacobson, Shane N Hall, and James R Swisher. Discrete-event simulation of health care systems. In *Patient flow: Reducing delay in healthcare delivery*, pages 211–252. Springer, 2006.

[6] Marco A Janssen. The evolution of rules in shedding-type of card games. *Advances in Complex Systems*, 13(06):741–754, 2010.

[7] JB Jun, Sheldon H Jacobson, and James R Swisher. Application of discrete-event simulation in health care clinics: A survey. *Journal of the operational research society*, 50(2):109–123, 1999.

[8] Damian Krenczyk and Malgorzata Olender. Production planning and control using advanced simulation systems. *Int. J. Mod. Manuf. Technol*, 6(2):38–43, 2014.

[9] Lawrence M Leemis and Stephen Keith Park. *Discrete-event simulation: A first course*. Pearson Prentice Hall Upper Saddle River, NJ, 2006.

[10] List of discrete event simulation software. List of discrete event simulation software — Wikipedia, the free encyclopedia, 2020. [Online; accessed 20-August-2020].

[11] Jayadev Misra. Distributed discrete-event simulation. *ACM Computing Surveys (CSUR)*, 18(1):39–65, 1986.

[12] Viknashvaran Narayanasamy, Kok Wai Wong, Chun Che Fung, and Shri Rai. Distinguishing games and simulation games from simulators. *Computers in Entertainment (CIE)*, 4(2):9–es, 2006.

[13] Marvin Ohriner. Finding the area under a curve by the monte carlo method. *The Physics Teacher*, 9(8):449–450, 1971.

[14] Mayank Patel. *Data Structure and Algorithm With C*. Educreation Publishing, 2018.

[15] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[16] John A Sokolowski and Catherine M Banks. *Modeling and simulation fundamentals: theoretical underpinnings and practical domains*. John Wiley & Sons, 2010.