CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Web application for demonstrating domain-independent game-playing algorithms

Bc. Nikita Mishchenko
Study programme: Open Informatics
Specialization: Software Engineering

January 2019
Supervisor: Mgr. Branislav Bošanský, Ph.D.

Acknowledgement

I thank Mgr. Branislav Bošanský, Ph.D., the supervisor of the diploma thesis for his valuable advice and suggestions, which were very helpful.

Declaration

I declare that presented work was made independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague 06. 01. 2019

. . . .

Abstrakt / Abstract

Diplomová práce se zabývá úlohou vytvoření webové aplikace pro demonstraci a hraní her pomoci doménové nezávislých algoritmu. V této práci jsme se seznámili s herní teoretickou knihovnou, která obsahuje sbírku algoritmů nezávislých na doménách schopných řešit a hrát různé třídy her. Zanalyzovali jsme existující řešení a zvolili vhodné technologií pro vývoj serverové aplikace, která se slouží jako adaptér pro herně-teoretickou knihovnu. Porovnali jsme různé JavaScript knihovny pro vývoj webových aplikaci a vizualizace grafického rozhraní ve webovém prohlížečů. Na základě toho jsme zvolili technologií, které použili pro vývoj klientské části. Byly implementovány aplikace na straně klienta a na straně serverů. Byli implementovány React komponenty, které umožňují vývojáři vytvářet a vizualizovat karetní, grafové a deskové hry. Byla navržena a implementována infrastruktura aplikací, která umožňuje vývojáři spravovat a přidávat nové hry a algoritmy.

Klíčová slova: gtlibrary, herně-teoretická knihovna, teorie her, Java, Javascript, webová aplikace, desková hra, karetní hra, grafová hra, vizualizace algoritmu.

This master thesis deals with the task of creating a web application for demonstrating domain-independent game-playing algorithms. In this work, we got familiar with the game-theoretic library that contains a collection of domain-independent algorithms capable of solving and playing various classes of games. We reviewed and chose suitable technologies for games visualization and building a back end application that appears as an adapter for the game-theoretic library. This work contains a comparison of various JavaScript libraries for game visualization. The technology stack was selected. Client-side and server-side applications were implemented. React components that allow a developer to build and visualize card, graph, board games were implemented. Back end application infrastructure that allows a developer to manage and add new games and algorithms was designed and implemented.

Keywords: gtlibrary, game-theoretic library, game theory, Java, Javascript, web application, board game, card game, graph game, algorithm visualization.

Contents /

Tables / Figures

Chapter 1
Introduction

Game theory is a huge scientific area. It could solve wide range of decision making problems using various algorithms.

Games are often used to test and evaluate game playing algorithms and its quality.

Many different games(domains) and algorithms are implemented in the game theoretic library [1]. This library was developed in the Artificial Intelligence Center of Faculty of Electrical Engineering in Czech Technical University in Prague. Game-theoretic library is a software library that contains a collection of domain-independent algorithms capable of solving and playing various classes of games.

One of the use cases how to demonstrate algorithms capabilities is to let a human player play against them. However, as a library, it does not have any visualization and does not provide an interface for the human player that will allow him to interact with algorithms.

The main goal of this diploma thesis is to create a web application that allows human participants to play against selected algorithms on a collection of at least 3 fundamentally different games.

In this work, we have to solve the following tasks:

- Get familiar with the game-theoretic library, main implemented algorithms, and select domains (games) for the web application.
- Get familiar with existing web frameworks and libraries that allow designing different types(board, card, graph, etc.) of games supported by the game-theoretic library, compare their characteristics, and choose the most suitable one for the work.
- Design the web application that allows human participants to play against selected algorithms with the emphasis on possible future extensions (new games or algorithms).
- Implement the web application and demonstrate its practical usability when multiple users will be simultaneously playing different games against different algorithms.
- Analyze and describe possible improvements, interesting use cases, existing problems and ways of solving them for the future application development and extension.

The resulting web application will be the only first part of game-theoretic library visualization. We will touch only a small part of the functionally provided by this library.

One of the outputs of this work is a technology stack that later will be used to implement other algorithms and domains.

Chapter 2
Background

In this chapter, we will get acquainted with a technical background and key definitions which will be used in this work. We are using these definitions in whole work. For this reason, it is important to understand their meaning to be in the context of this work.

2.1 Game theory

Game theory is a widely used methodology for analyzing multi-agent systems by applying formal mathematical models and solution concepts. One focus of computational game theory is the development of scalable algorithms for reasoning about very large games [2].

Since the main goal of this work is to develop a web application to visualize games and let people play against the algorithms, game theory is one of the key terms. It is a very huge scientific area and here we are touching an only small part of it.

A core of the application is the game theoretic library [1]. It contains a huge set of different domains and algorithms. We will focus on some of them.

Simultaneous move games is a finite two-player game with simultaneous moves and chance events (also called Markov games, or stacked matrix games) is a tuple (N, S, A, T , $\Delta_\star$, u_i, s_0), where $S = D \cup C \cup Z$ [3].

- Z is a set of terminal states.
- D is a set of states where players make decisions.
- C is a set of states where chance events occur. Could be empty.
- S is a set of all states.
- $N = \{ 1, 2, \star \}$ is a set of players. It contains a player labels, where $\star$ denotes the change player. A player is denoted $i \in N$.
- $A = A_1 \times A_2$ is a set of joint actions of individual players. $A_i(s)$ is a set of actions available for the player i in state s, where $i \in N, s \in S$.
- T is a transition function definded by $T : S \times A_1 \times A_2 \mapsto S$. This function defines the successor state by given a current state and actions for both players.
- $\Delta_\star : C \mapsto \Delta(S)$ describes a probability distribution over possible successor states of the chance event.
- u_i is the utility function. $u_i : Z \mapsto [v_{min}, v_{max}] \subseteq \mathbb{R}$ gives the utility of player i, where v_{min} and v_{max} are the minimum and maximum possible utility.
- s_0 is an initial state. We assume zero-sum games: $\forall z \in Z, u_1(z) = -u_2(z)$. The game begins in an initial state s_0 and a subset of a game that starts in some node s is called a subgame.

2.2 Game playing algorithms

In this work, we are working only with on-line algorithms.

The on-line algorithm is an algorithm that receives a sequence of requests and performs an immediate action for each request [4].

For the demonstration purposes we selected the following algorithms:

- Backward Induction
- Backward Induction with Serialized Alpha-Beta Bounds
- Double-Oracle
- Double-Oracle with Serialized Alpha-Beta Bounds
- Online Outcome Sampling
- Monte Carlo Tree Search Upper Confidence Bounds for Trees
- Monte Carlo Tree Search Exponential-Weight Algorithm for Exploration and Exploitation
- Monte Carlo Tree Search Regret Matching
- Random

For the reason that implementation details of these algorithms are not part of this work, we will briefly describe a few of them. Detailed information about these algorithms could be found in paper „Algorithms for Computing Strategies in Two-Player Simultaneous Move Games“ [3].

2.2.1 Double-Oracle $\alpha\beta$

Double-Oracle with Serialized Alpha-Beta Bounds is an enhancement of Backward Induction with Serialized Alpha-Beta Bounds algorithm with doulbe-oracle algorithm. Instead of immediately evaluating each of the successors of the current game state and solving the linear program, the algorithm can exploit the double-oracle algorithm [3].

The goal of double-oracle algorithm is to find a solution of the matrix game without solving complete the whole game [3].

We can split the algorithm to the following steps [3]:

- Algorithm tests, if the whole game can be solved by using serialized variants of the game.
- If not, then the algorithm initializes the restricted game with an arbitrary action for each game state.
- The algorithm starts iterations of the double-oracle algorithm.
- The algorithm computes the value for each of the successors of the restricted game that have the current value not known.
- The algorithm solves the restricted game and stores the optimal strategies.
- The algorithm computes best responses for each player and updates bounds.
- The algorithm expands the restricted games with the new best response actions until lower and upper bound are equal.

2.2.2 MCTS UTC

Monte Carlo Tree Search is a heuristics best-first search algorithm that uses stochastic simulations [5]. This algorithm is trying to choose most benefitial actions. An algorithm builds a tree of possible future game states. Each iteration consist from four steps: Selection, Expansion, Simulation, Backpropagation.

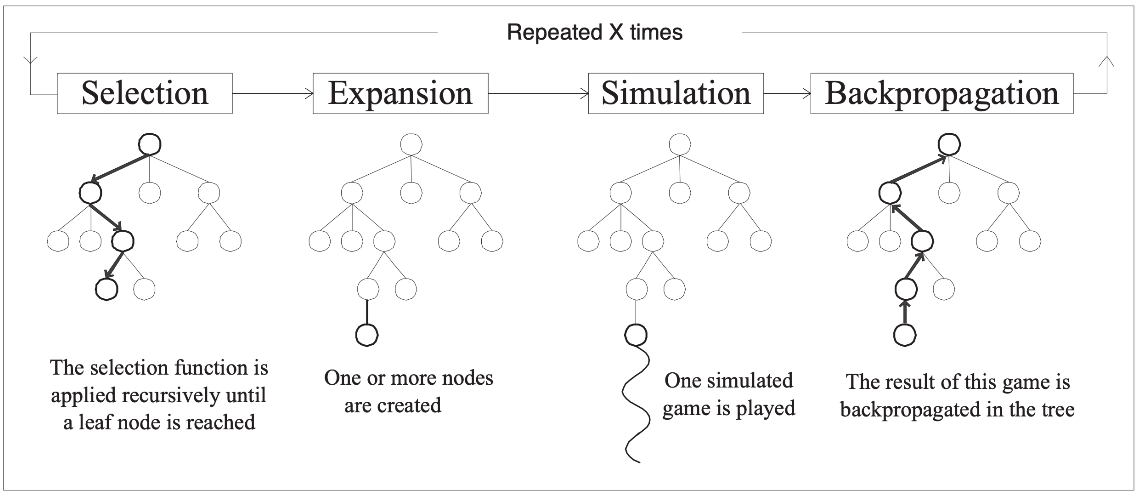

Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Figure 2.1. Monte Carlo Tree Search [5].

Upper Confidence Bounds for Trees [6] is a Upper Confidence Bound 1 [7] algorithm applied to trees. It is used to balance between nodes with high reward and depth.

In UCT [6] algorithm we select an action that maximizes $Q_t(s, a, d) + c_{N_{s,d}(t), N_{s,a,d}(t)}$, where:

- s is a state.
- d is a depth.
- $Q_t(s, a, d)$ is the estimated value of action a in state s at depth d and time t.
- $N_{s,d}(t)$ is the number of times state s has been visited up to time t at depth d.
- $N_{s,a,d}(t)$ is the number of times action a was selected when state s has been visited up to time t at depth d.

2.2.3 Online Outcome Sampling

Online Outcome Sampling (OOS) is a simulationbased algorithm based on Monte Carlo Counterfactual Regret Minimization [8]. OOS adds to main modifications: Incremental Game Tree Building and In-Match Search Targeting. It starts from a single root information set and adds at most one set to the memory in each iteration in case if information set that is not in memory reached. Also, this algorithm trying to search towards the histories that are more likely to occur during the match currently played instead of start searching from the root in each iteration.

2.2.4 Random algorithm

Random algorithm has very simple logic. It expands current game state to get a set of available actions for the player and then select a random action from this set.

2.3 Game domains

In this section, we will describe games rules and principles. As it was mentioned in Section 1, we have to create a web application that allows human players to play against game playing algorithms on a collection of 3 fundamentally different games. These games are GoofSpiel, Oshi-Zumo, and Pursuit-Evasion.

2.3.1 GoofSpiel game

GoofSpiel is a simultaneous move card game. Usually is playing with 3 identical decks of cards. One deck is for the chance player and one for each player.

Each deck could have from 1 to d cards, where d is a natural number. In standard rules, d is equal to 13. Every card has its value. Values of the number cards are equal to their number. Ace value is equal to 1. Jack value is equal to 11. Queen value is equal to 12. King value is equal to 13.

The game is played in rounds. At the beginning of each round, one card from the chance deck is revealed. Then each player selects one of his cards and makes a bid. Both players show their selected card at the same moment. Then the player who selected a card with the biggest value is winning the round and receive a reward equal to the revealed card value from the chance player deck. In a case when both players select the card with the same value, the round result is drawn and both players receive 0 points.

When there are no cards left, the game ends. The winner of the game is a player with the biggest sum of card values he received during the whole game.

2.3.2 Pursuit-Evasion game

Pursuit-Evasion is a graph game. There is a single evader and a pursuer that controls 2 pursuing units on a four-connected grid.

Players are starting at the predefined positions and are making moves simultaneously. During one turn players can move each of their units to adjacent nodes. There is also a goal node.

Evader player wins when is reaching a goal node avoiding a pursuer player. Pursuer player wins when is catching the evader by one of his units.

2.3.3 Oshi-Zumo game

Oshi-Zumo is a board game. There are two players, both starting with N coins. The game is played on the board with $1 \times 2K + 1$ dimensions. There is also a wrestler starting at position K (in the center of the board).

The game is played in rounds. In each round, both players simultaneously bid a number of coins from M to the amount they have, where M is predefined constant and represent the minimum bid. If the player has less then M coins he can bid only zero. At the round end, the player with the highest bid moves the wrestler one position closer to the opponent. If player bids are equal, the wrestler does not move.

The game ends when both players will have 0 or less then M coins, or if the wrestler will leave the playing field. The winner is the farthest player from the wrestler.

2.4 Server-side technologies

We need to develop a web application. It means that we need to allow the human player to interact with a game theoretic library via the web interface. For this purposes, we have to separate application to two parts - client and server.

2.4.1 Client-server style

Client-server is a two-level layered architectural style. It has 2 parts client and server. A client is requesting for some services. A server is providing these services. In this style, communication is initiated by the client. The client makes a request with parameters to the server to get the required data or to execute some process. The server responds to this request with the result of request evaluation.

In the scope of this work we call a client or fronted a JavaScript application running in the web browser and server or backend a Java application which uses game theoretic library running on the server.

Game theoretic library is written in Java.

2.4.2 Java and JVM based languages

Java is a general-purpose, concurrent, class-based, object-oriented programming language [9]. Java was inspired by C++ but writing code in Java is easier and safer than in C++.

Java applications are compiled to bytecode which is then executed on Java Virtual Machine (JVM). It helps to execute Java code on different platforms without any modifications. Java Virtual Machine is written for the specific platform, but there are many implementations for all most popular platforms. JVM is doing many different optimizations to provide good performance and optimize hardware resource usages.

Java has automatic memory management. It uses a garbage collector to manage object lifecycle. When we create an object, it will be automatically allocated in memory. We don't need to care about object destruction, the garbage collector will do it for us, using principles based on its type.

Relatively fast and safe development, lots of extensions and different built-in tools makes Java one of the most popular programming languages in the world.

Java ideas inspired developers to create new JVM based languages. These languages are also compiled to the bytecode, and bytecode could be executed on Java Virtual Machine. It also brings full compatibility with Java classes. One of these languages is Groovy.

Apache **Groovy** is a optionally typed and dynamic language with familiar to Java syntax. It has capabilities to use static-typing and static compilation. Groovy features allow us to use it as scripting language as well. Groovy is often used for domain specific languages definition. It has nice built-in language features which could simplify code writing and make code more readable.

Groovy characteristics makes this language very useful for specific tasks and domains. In this work, we used Groovy for writing tests. It is described more in details in Section 8.1.

2.4.3 Spring framework and modules

Spring is a very flexible open source application framework. It provides all necessary tools to the developer to create Java enterprise application easily. It gives the flexibility to use any kind of architecture we need.

At its core, Spring offers a container referred to the Spring application context. Spring automatically create and manage these containers. It could limit their scope, control instantiation policy, add additional functionally using proxies, etc. One of the main Spring benefits is dependency injection. Spring can inject required components to other components and will care about providing correct instance based on defined rules. An application composed of such components is maintainable, extensible, flexible and clean. It could also be used to keep isolation on the level we wanted.

It is important to know, that in simple words in the context of Spring component is an auto-configured bean.

Some of the Spring Framework principles [10]:

- *Provide choice at every level.* Spring lets programmer defer design decisions as late as possible. It allows us to switch functionality providers using configurations. For example, we can switch persistence providers through configuration without changing your code.
- *Accommodate diverse perspectives.* Spring embraces flexibility and is not opinionated about how things should be done. It supports a wide range of application needs with different perspectives.
- *Maintain strong backward compatibility.* Spring's evolution has been carefully managed to force few breaking changes between versions. Spring supports a carefully chosen range of JDK versions and third-party libraries to facilitate maintenance of applications and libraries that depend on Spring.
- *Care about API design.* The Spring team puts a lot of thought and time into making APIs that are intuitive and that hold up across many versions and many years.
- *Set high standards for code quality.* The Spring Framework puts a strong emphasis on meaningful, current, and accurate Javadocs. It is one of very few projects that can claim clean code structure with no circular dependencies between packages.

Spring has a big number of different modules. By combining them, we can achieve the desired functionality. One of these modules is a **Spring Boot** 2.2.

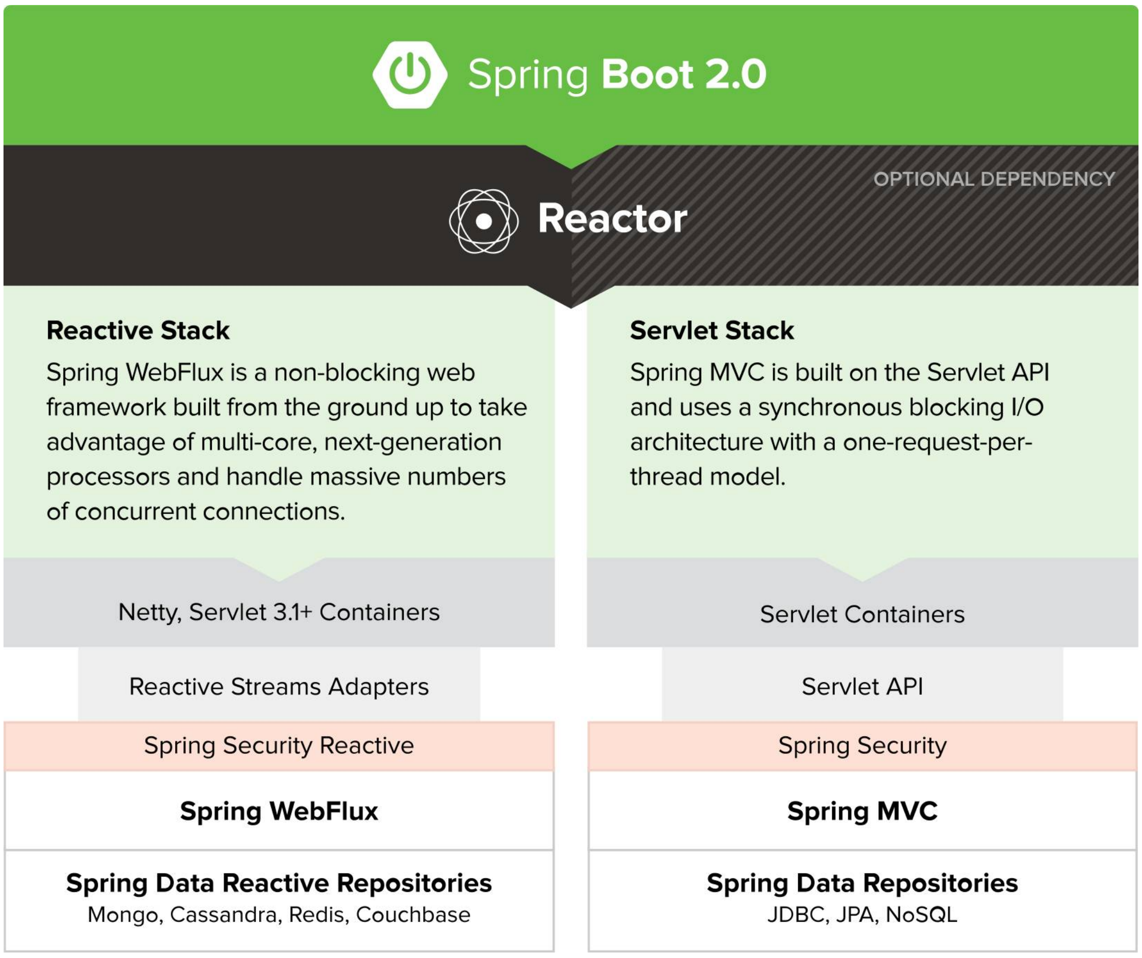

Spring Boot 2.0
Reactor
OPTIONAL DEPENDENCY
Reactive Stack
Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.
Servlet Stack
Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.
Netty, Servlet 3.1+ Containers
Servlet Containers
Reactive Streams Adapters
Servlet API
Spring Security Reactive
Spring Security
Spring WebFlux
Spring MVC
Spring Data Reactive Repositories
Mongo, Cassandra, Redis, Couchbase
Spring Data Repositories
JDBC, JPA, NoSQL

Figure 2.2. Spring Boot structure.

Spring Boot let the developer create and run an application even faster. It provides a powerful and flexible way to configure an application using configuration beans, application.properties, etc. Spring Boot autoconfiguration mechanism dramatically decrease the amount of code to run an application. Furthermore, it generates fat jars. It means that as build result we will receive a normal jar file which could be run like ordinary Java SE application, but it will contain all necessary dependencies, configurations and application server. And Spring Boot Test allow writing integration tests easy and comfortable.

By using the Spring framework and its modules, a developer can concentrate on application logic and do not spend much time on configurations and integrations.

2.5 Client-side technologies

On the client-side the only one language supported by all web browsers is a JavaScript.

2.5.1 JavaScript and ECMAScript standard

JavaScript is a high-level, dynamic, weakly typed, multi-paradigm, interpreted programming language that conforms to the ECMAScript specification. It allows us to write complex dynamic applications for the web. With JavaScript, we can manipulate HTML page content and use various APIs provided by the web browser.

ECMAScript(ES) is a scripting language specification. It was created to standardize JavaScript implementations by specifying standard methods, APIs and functionally.

We can often hear JavaScript code of ECMAScript 6 standard. It means that this code is using functionality specified by the 6th edition of ECMAScript standards. It also used to validate browser support for the specific technology.

ECMAScript the 6th edition (ECMAScript 2015) adds some significant syntax changes. It defined support for classes, modules, arrow functions, generators, iterators, promises, new collections, typed arrays, reflections, and other improvements.

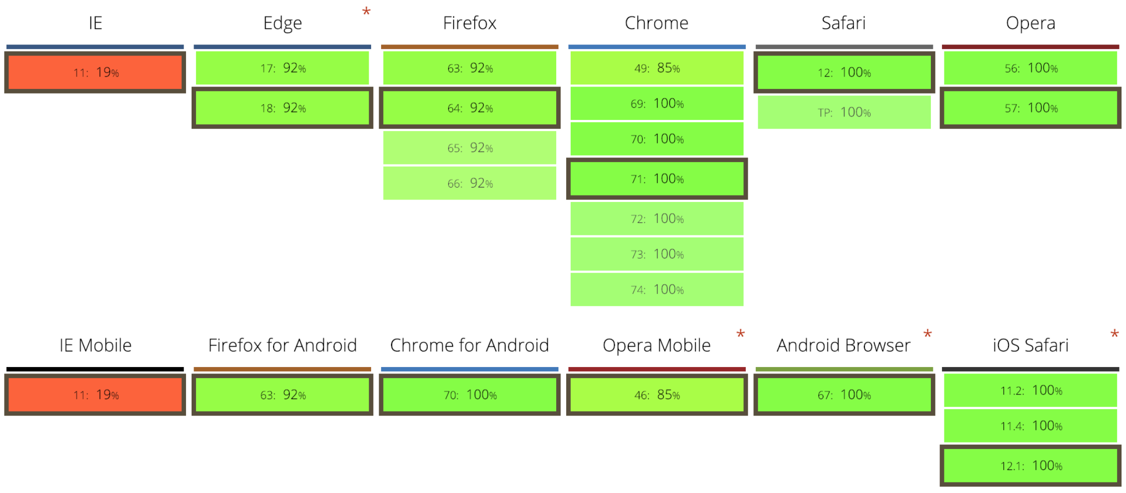


Figure 2.3. ECMAScript 6 browser support [11].

In Figure 2.3 we can see the percentage of ECMAScript 6 functions supported by most popular web browsers. For the reason that peoples want to run their ES6 applications in all browsers, and it is still not fully supported, they are using babel compiler. It can compile ES6 code to the previous standard which is supported by all modern browsers.

2.5.2 Babel compiler

Babel is JavaScript compiler tool. It was created to compile JavaScript code of ECMAScript 2015+ to the version supported by all browsers. Currently ECMAScript standard, supported by all web browsers is ECMAScript 5th edition. Using babel, we can write nice and beautiful code and use it everywhere. For example, following arrow function:

```
[1, 2, 3].map(n => n + 1);
```

Listing 2.1 ECMAScript 6th standard syntax example.

will be converted to

```
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

Listing 2.2 ECMAScript 5th standard syntax example.

For this reason, babel is often used in JavaScript development to simplify it and make sure that application will work correctly in any web browser.

Nowadays modern JavaScript applications are compiled to the resulting files on the backend using different building tools. One of these tools is a webpack.

2.5.3 Webpack

Webpack is an open-source JavaScript module bundler. It helps to build an application. Webpack builds dependencies trees and bundle required parts. It also can preprocess and organize resources.

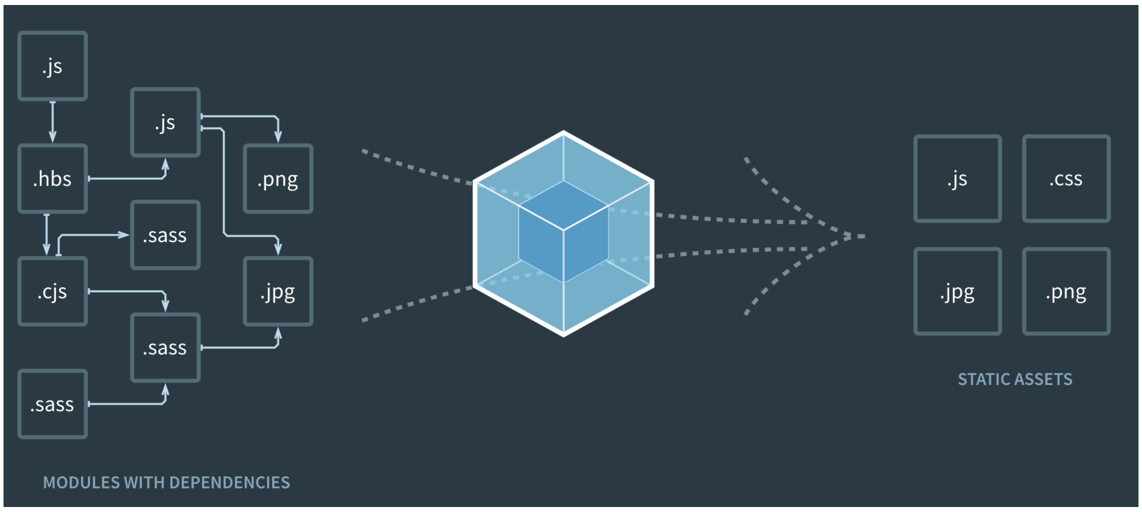


Figure 2.4. Webpack module bundler. [12].

Webpack can do different code manipulations using various extensions. For example, it could minimize JavaScript files, compile `.sass` and `.less` files to `.css`, modify class names, optimize images, execute babel compiler. And as a result, it will generate static assets, supported by all web browsers.

Webpack has flexible configuration model. A developer can specify everything he wants to do with his code.

To use webpack, we need to have a Node.js because webpack runs on it.

2.5.4 Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. It is asynchronous event-driven runtime.

V8 engine compiles JavaScript code to machine native code. It allows Node.js to execute JavaScript code not only in web browsers but also at any platform. Using various modules and extensions, we can write any application in JavaScript. For example, it could be a web server, command line tools, desktop application. It uses a non-blocking asynchronous approach, which makes an application very efficient.

The main idea of Node.js is JavaScript everywhere. It allows developers to write applications for different environments in a single language.

Node.js has preinstalled package manager called npm.

npm is a package manager for JavaScript world technologies. It has a huge public repository with different packages, called the npm registry. npm can simply install the required package and will take care of all its child dependencies with correct versions.

npm store downloaded dependencies in the `node_modules` directory and is able to save packages names and versions in the `package.json` file in JSON format. With this file, we don't need to have installed packages source codes as part of the project. They could be automatically installed by calling `npm install` command.

Chapter 3
Analysis and design of the server-side application

In this chapter, we will analyze what technology stack we will need to use to implement the server-side application. We will analyze all aspects of the application, define the goals, and compare different technologies options to choose the best combination for the back end application.

The purpose of the back end part is to let the client communicate with the game-theoretic library and obtain opponent actions and information about the game from algorithms based on the current game state. First, we need to define the requirements for the server-side application.

Back end application requirements:

- Provide information about supported games to the client.
- Provide information about supported game playing algorithms to the client.
- Provide information about game roles which player can play to the client.
- Provide available configurations for a specific game to the client.
- Allow the client to execute game playing algorithms for a concrete game for current game state and game configuration.
- Let players play simultaneously in different games with different configurations.

3.1 Core technology

Because game-theoretic library was written in Java, we chose Java as the main programming language for a back end application. The back end should be able to receive requests and respond to the client. It means that it will make sense to use one of the realizations of Java Enterprise Edition specification.

Java EE is a powerful tool to build an application. There are a lot of realizations from different companies such as Wildfly application server from Red Hat that implements pure Java EE specification. But also exists different application frameworks that provide us with a lot of functionality out of the box. To pick the best framework for our purposes, we need to define what we want to achieve:

- Modern framework - it should be a modern framework that supports most of the popular technologies or provides a simple way to extend it to get the support of the desired technology.
- Big community - when software has a big community, it usually means that we will be able to quickly find an answer to our question or get a valuable solution. Big community means that it used by many peoples and they help to find and report bugs. Also, for open source projects bugs could be fixed much faster. A big community is a good sign that software is good, otherwise, it will not be so popular. It also means that in most cases project will be supported and developed in the future. Last but

not least fact is that projects with a big community usually have many examples and third-party projects and extensions that could be helpful for specific tasks.
- Easy to use and configure - we want to be able to start using it relatively quick and without hundreds of configurations and incrementally improve and add configurations and functionality based on our needs.
- Offering lots of modules and extensions - we want to use existing, tested and reviewed by many other peoples solutions where it is possible.
- Extensible and Flexible - we should be able to customize or extend different pieces of application depending on application-specific logic.
- Good documentation, articles and examples - learn new things is always easier when examples and reasonable explanations are provided. It could help solve some problems.

The best choice is Spring framework 2.4.3. It is the most popular Java framework now. Spring has a huge community and a big number of projects and extensions. It allows us to do everything we need fast and in a structural way. Moreover, the application could be easily maintained by any developer with knowledge in Java enterprise area. Spring has good integration in all modern IDEs. And using Spring Boot project we can create and run an application as a fat jar in a few minutes.

3.2 Communication between client and server

Server and client need to communicate between themselves. To choose the way of communication we have compared the most popular methods:

- REST
- SOAP
- GraphQL
- WebSockets
- Server-sent events

We have to choose a method that is easy to use and fit our aim.

3.2.1 Simple Object Access Protocol

SOAP (Simple Object Access Protocol) is a messaging protocol specification for exchanging structured information. It operates with two functions GET and POST for retrieving and adding or modifying data. It uses XML format for its messages. For web service purposes XML schema is required written with WSDL (Web Services Description Language). To exchange data between applications, request and response formats should be defined. Use single endpoint URL.

3.2.2 Representational State Transfer

REST (Representational State Transfer) is a software architectural style that defines a set of constraints to be used for building web services. Web services which provide API in REST architectural style called RESTful. RESTful API is typically mapping HTTP request methods:

- **GET** - retrieve record or collection of records with or without details
- **POST** - create a record or collection of records
- **UPDATE** - replace the record or collection of records
- **DELETE** - delete the record or collection of records

- **PATCH** - partial or full modification of existing record or collection of records

Methods without parameters are typically used for reading or modifying a list of elements. Calling methods with parameters used to read or modify specific record or subset of records. Sometimes POST method could be used to query data when one of the parameters values contains sensitive data. For example, searching person by social security number.

RESTful API should support OPTIONS method. When the application receives OPTIONS request, it should provide information about supported methods and content types for current URL.

RESTful APIs also have the following constraints [13]:

- **Client server architecture** - separate client and server brings portability, scalability and allow to do development independently.
- **Stateless** - by calling RESTful API we do not create session and do not keep context for each client. It induces visibility, reliability, and scalability. We do not need to share context between service instances and keep additional information in memory.
- **Cacheabillity** - using cache become very simple. We can easy cache responses for same read requests.
- **Uniform Interface** - implementations are decoupled from the services they provide. REST is defined by four interface constraints: identification of resources, manipulation of resources through representations, self-descriptive messages and hypermedia as the engine of application state.
- **Layered System** - client does not know if is connected to end server or not. Client could communicate with end application via gateway application, load balancer etc. It gives pliability to scale application, modify or duplicate requests on server side.
- **Code-On-Demand** - REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

3.2.3 GraphQL

GraphQL is an application-layer query language that can be used with any database [14]. GraphQL requires a schema with data definition to be able to make requests. It is a powerful query language which allows the client to query data it needs. For example, when we need to get only a person's date of birth and address, we can specify only these fields in request instead of query whole entity. Using GraphQL, we decrease the amount of data which server send and give a possibility to the client to choose what it needs. It could be useful for CRUD applications but brings extra complexity to back end part implementation because developer must define a schema and deal with all tricky moments, such as circular dependencies, by limiting depths or size of the response. Also, it slightly decreases performance.

3.2.4 Summary

To choose which option fits better we defined the following requirements:

- Fast and easy to develop, maintain and extend - we do not want to add unnecessary extra complexity.
- Stateless - in terms of this work we are using online algorithms, we want to receive a response based on request body as soon as possible. From this perspective, it will make sense to add stateless requirement. In other words, we want to receive a request, calculate opponent move based on received data and send a response.

	SOAP	REST	GraphQL
Type	Protocol	Architectural style	Query language
Data formats	XML	JSON, XML, HTML and other microformats	JSON
Communication protocol	SMTP, TCP, HTTP	HTTP	HTTP
Data typing	Strong	Weak	Strong
Use complexity	High	Low	Medium

Table 3.1. SOAP, REST, GraphQL comparison

- Client friendly - we also should care about client application. Because we are developing a web application, the client will be executed in the web browser. And main programming language for web applications is JavaScript. From these facts, we conclude that one of the requirements is comfortable usage from JavaScript.

  In table 3.2 we summed up requirements matches.

	SOAP	REST	GraphQL
Easy to use	-	+	-
Client friendly	-	+	+
Stateless	-	+	+
Fit to the aim	-	+	-

Table 3.2. SOAP, REST, GraphQL match requirements

As sum up, we chose REST with JSON format for communication between client and server.

3.2.5 Server-sent communication methods

There were mentioned two more options above - WebSockets and Server-sent events. These two methods are interesting from the side that they allow the server to provide data even if client didn't request it.

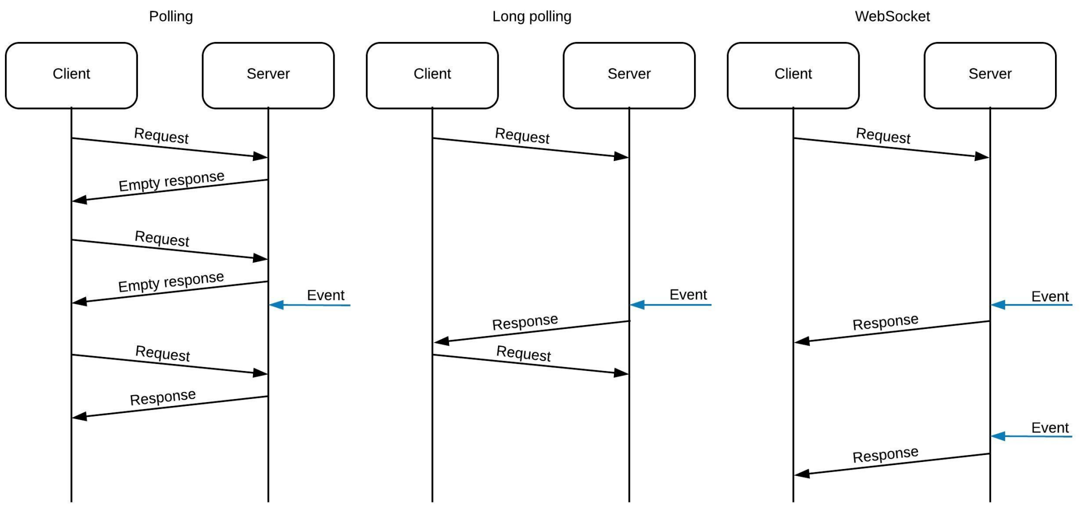


Figure 3.1. Polling and WebSocket comparison.

At first glance, it makes no sense to use for online algorithms, but it could be used in further application development for offline algorithms and even for cases when the algorithm will not be limited by execution time, but by human player move time. For example, while the human player makes his move, algorithm on server side could continue calculating best action and provide better options to the client to move. And at the moment when the player will make his action, the client application will use the best received action.

WebSocket is a protocol which makes possible two-way communication between server and web browser. Handshaking initiated by client required to establish WebSocket connection. After a connection is established server can send messages to the client. From the definition, we see that it's stateful.

Using WebSocket could decrease the load on the server and as we see in Figure 3.1 it provides updates more accurate from the time perspective.

Server-sent events are similar to WebSockets but are only unidirectional. A client could listen for events from the server using the EventSource interface. Server sent events in `text/event-stream` format.

As it was described above. We could use server-sent messages to increase time algorithm could use to achieve a better result without any discomfort for the human player.

3.3 Building tool

In JVM based programming languages world exists set of building tools. Building tool helps to build and manage a project. It could make different code manipulations, execute scripts to make code optimizations, download required libraries and care about their versions to make them compatible with each other, managing application versions, build correct artifacts, run tests and code analysis, doing publishing and many other useful features. Nowadays the two most popular tools are Gradle and Maven.

3.3.1 Apache Maven

Apache Maven is a build automation and dependency management tool. It has different goals to achieve the desired result. For example, we can call `mvn compile`, to execute compile goal using maven. `mvn compile` will compile the project. Some of the goals have prerequisites, in other words, the execution of the goal could depend on other goals. It means that some goals must be completed so that the required goal could start its execution. Maven uses XML files to describe the application and define goals. Maven uses the Project Object Model (POM) to provide all configuration for a project.

3.3.2 Gradle

Gradle is a build automation system. It uses Maven concepts but use Groovy-based DSL for project configuration. Latest versions of Gradle support Kotlin DSL. Gradle has a lot of useful features like incremental building, caching, parallel execution. Because it based on Groovy language, we can write code directly in configuration. It makes super flexible and easy to use.

3.3.3 Summary

Gradle is more efficient then Maven. We can see performance comparison in Figure 3.2

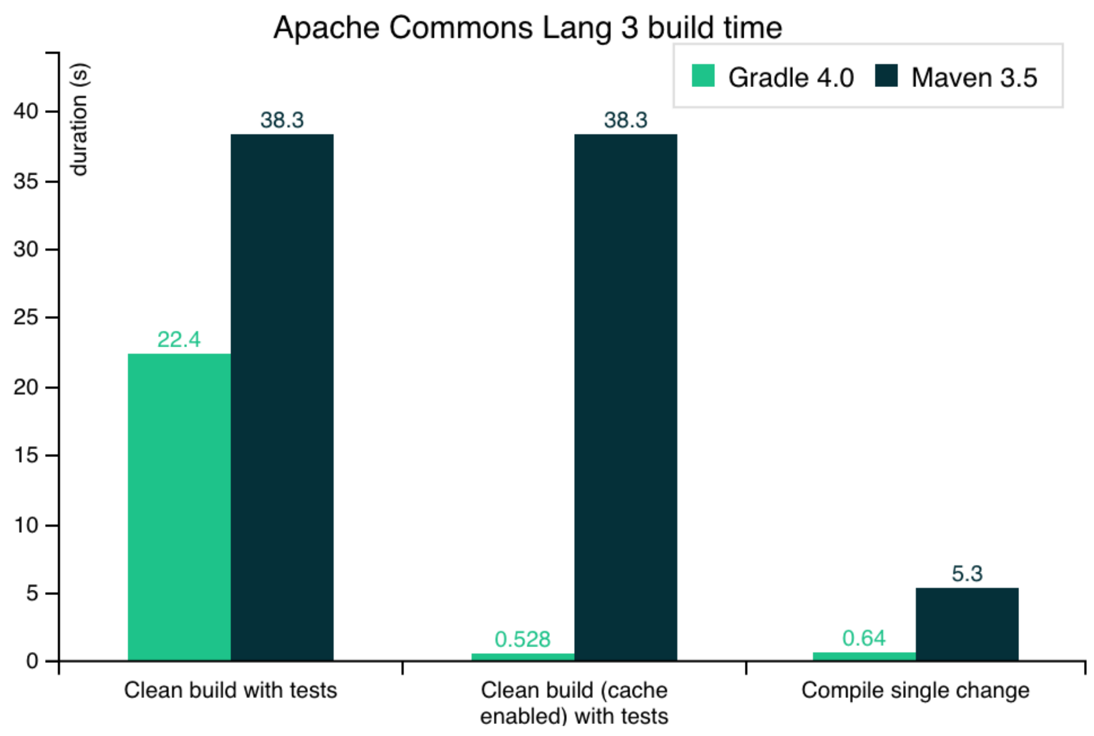


Figure 3.2. Gradle and Maven performance comparison [15].

There are many debates about which tool is better, but Gradle does everything that Maven can do and even more. Moreover, Gradle is easier to debug, extend and it works much faster. We chose Gradle to build backend application.

Chapter 4 Analysis and design of the client-side application

Every game is dynamic content. In modern web browsers, only one technology with built-in support is a JavaScript. This simple scripting language in combination with various APIs provided by the web browser becomes a powerful tool to build interactive dynamic content.

In this chapter, we will define the requirements for a front end application, analyze existing solutions, and choose technologies for the needs of the application.

4.1 Core technology

The first thing we did was choosing the backbone of our application. The following criteria were defined to choose the best tool for the application:

- Should be open source and with free to use license.
- Simple to use - we want to start to use the basic functionality as fast as possible and incrementally increase its complexity.
- Support of all popular web browsers - the main host of our application is a web browser. We want the application to work and look the same in all web browsers.
- Fast to develop - it is important when a library has some basic functionality implemented, and we do not need to write a lot of boilerplate code. We want to start implementing application-specific logic instead of solving problems that already have a solution.
- Popular in modern web - usage of modern technologies helps us to achieve a defined goal much faster and simpler. It could also increase code performance and visual representation.
- Has good integration capabilities with third-party libraries - selected technology should not solve all our problems, but it is good when we can extend it using third-party libraries or libraries written by ourselves.
- Good performance - the end user should not wait for a long time to get the results of his action. We need to find a solution that will have a good architecture for manipulating with different blocks and content.
- Flexible - we need to find a tool that will allow us to build what we want.
- Big community - community is a good indicator of software quality. Also, this is a great source of information and solutions for problems we could meet. Community for the open source project is a testers group as well.
- Has different extensions and plugins - using already implemented solutions is often a good way of solving some task if it fits our needs. It is tested by various peoples and save our time.
- Perspective - we want to find a technology that will be supported and developed in the future.

- Well documented - documentation is very important to start working with new things and even to remind some specific functionality.

There were 4 candidates selected matches most of these criteria.

- React.js library
- Angular framework
- Vue.js framework
- Pure JavaScript

On the table 4.1 we can see some statistics based on Github data related to popularity, support and development process.

	Angular [16]	React.js [17]	Vue.js [18]
Watches	3310	6596	5723
Stars	43837	118488	123273
Forks	11293	21496	17631
Issues	2235	394	163
Commits	12397	10561	2750
Licence	MIT	MIT	MIT

Table 4.1. Angular, React.js, Vue.js Github statistics.

4.1.1 Angular

Angular is an open source JavaScript framework maintained by Google, which provides different APIs and structures to help develop modern applications quickly and simply.

Angular taps into some of the best aspects of server-side development and uses them to enhance HTML in the browser, creating a foundation that makes building rich applications simpler and easier. Angular applications are built around a design pattern called Model-View-Controller (MVC) [19]:

- Extendable - it is easy to enhance applications to create new and useful features.
- Maintainable - Angular apps are easy to debug and fix.
- Testable - Angular has good support for unit and end-to-end testing.
- Standardized - Angular builds on the innate capabilities of the web browser without getting in your way, allowing you to create standards-compliant web apps that take advantage of the latest features (such as HTML5 APIs) and popular tools and frameworks.

Angular applications are usually written in TypeScript. **TypeScript** is open source programming language developed by Microsoft. It is kind of JavaScript extension which provides more functionality than pure JavaScript and allow to write code in a more readable and simpler manner. Using TypeScript compiler, we can compile TypeScript code to JavaScript code compatible with a standard supported by all popular modern desktop and mobile browsers.

4.1.2 React.js

React.js is a JavaScript library for building user interfaces. For writing React applications is typically used pure JavaScript in ECMAScript 6 style. This code then compiled to JavaScript of ECMAScript 5 standard using babel compiler to be compatible with all web browsers.

React allows writing code in a clear declarative way. It uses component-based architecture style which helps to divide application to different components and reuse them. The component has its own state and properties. Using them, we can define how the component should behave and look. This is a great benefit, we do not care about the outside situation, we just have current state and properties and process them, if state or properties are changed React will handle it for us.

React component has its own lifecycle. We can see it in Figure 4.1.

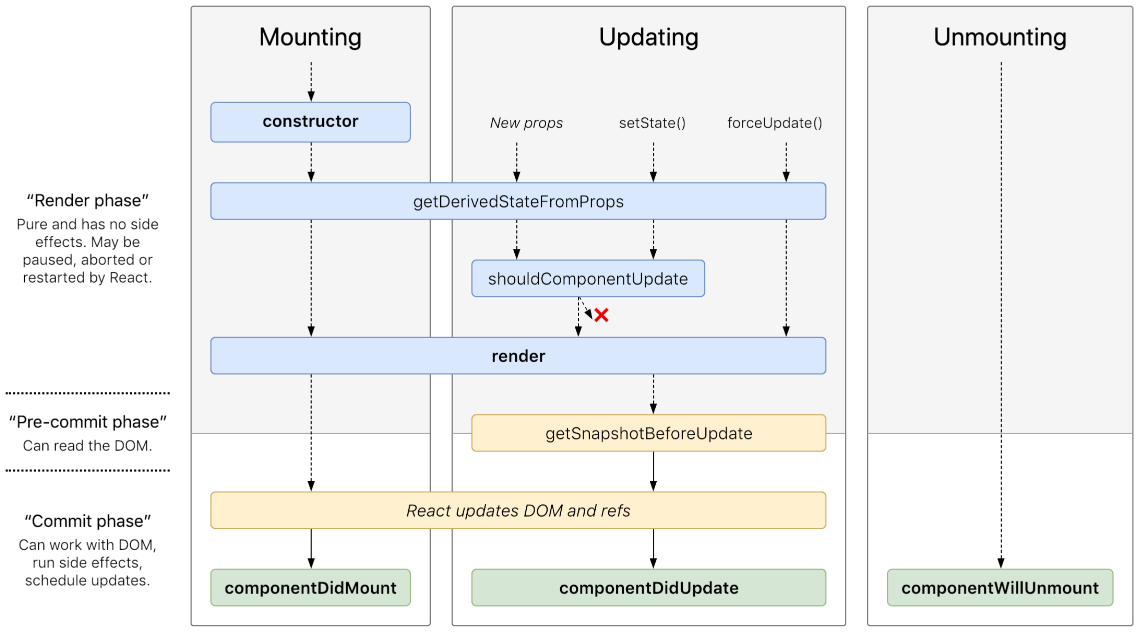


Figure 4.1. React component lifecycle [20].

Using these hooks, we can control component by performing or not performing required actions based on the current situation.

One more important note about a component state is that state is immutable. It means that we cannot modify component state, we can only replace one immutable state with another. That is a great approach. It helps to avoid collisions and make logical flow transparent clear.

For the goal of this work state also is a great benefit, because we concept of game state is ideally fits into React component state.

The great innovation which React bring to web development is a virtual DOM. DOM manipulation operations are very computationally expensive. React has its own virtual DOM model which is linked to HTML DOM elements. Also, react do not replace an element when is possible to change some of their parts or update only some part of DOM tree. This approach made React extremely fast.

React uses JSX. It is an extension on JavaScript used for markup. Instead of separating technologies by putting markup and logic in separate files, React separates concerns. It becomes very comfortable to use JSX in components, we can manipulate with content by using a mix of HTML and JavaScript syntax.

4.1.3 Vue.js

Vue.js is a JavaScript framework which becomes more and more popular. It mixes ideas of React and Angular. It allows to use JSX and ECMAScript 6 and let the developer decide to use the core library or whole infrastructure provided by the framework. In the standard library it uses ECMAScript 5 style with pure javascript object syntax to define all aspects of the component. It is very similar to React and uses most of its ideas like components with their lifecycle, virtual DOM, etc.

By comparing all positive and negative aspects of these tools is hard to tell if it will be comfortable to use until we do not try it by ourselves. Even after sample run of each of them is better to check what development community thinks about this. On figure 4.2 we can see statistics for year 2018 [21].

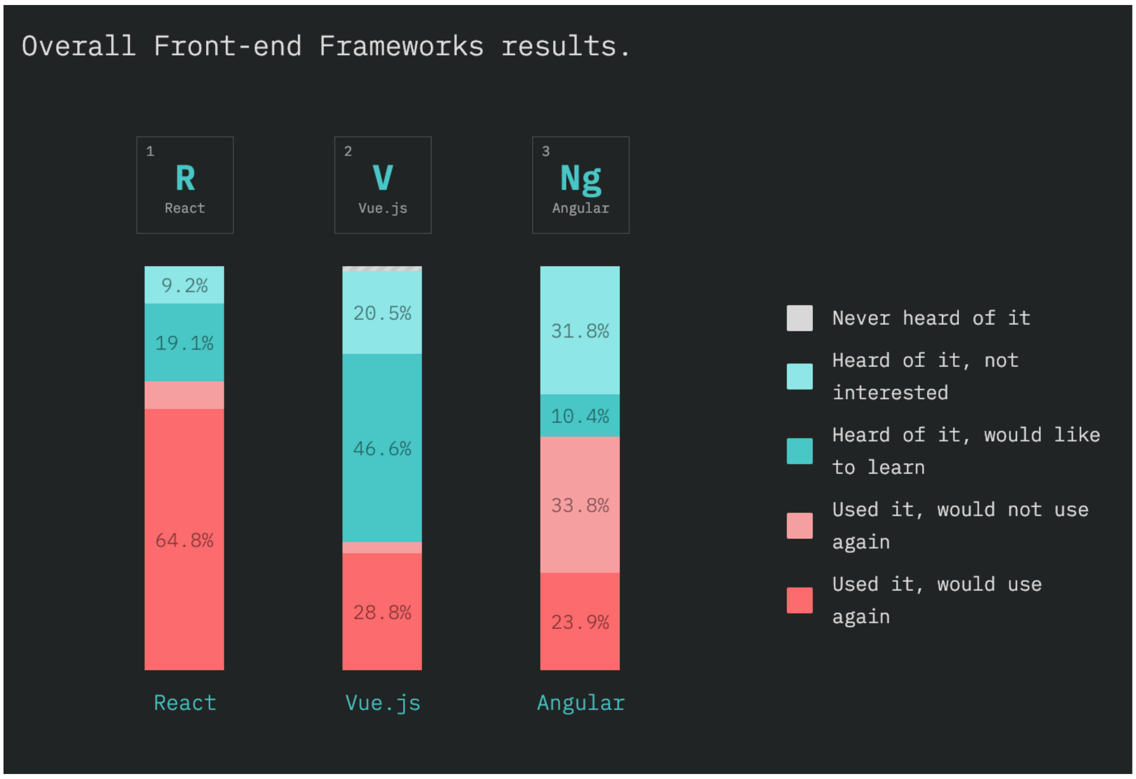


Figure 4.2. Frontend frameworks comparison results.

4.1.4 Summary

Using pure JavaScript was also considered as an option. In combination with babel, we can use the ECMAScript 6 standard without caring about browser compatibility. But we will need to implement a lot of things which are already implemented in these frameworks.

We tried to write a simple application with each of these tools but React is more comfortable for us. And as we can see for most developers as well.

React provides us with component-based architecture. Using this property, we can develop small components which then will be reused for games of the same type. Also, state term fits our needs ideally. It is easy to learn, and a lot of third-party libraries and components exists. Furthermore, it is considered a good practice to use a maximum of native language abilities in software engineering and React uses pure JavaScript. We should not forget that this is a library, but not a framework. It allows us to build an application as we want by adding libraries we need. It will reduce the number of unnecessary dependencies. By choosing React, we can be confident that the application will be simply maintained and extended in the future.

4.2 Card games

In this, 4.3 and 4.4 sections we will analyze existing libraries for solving the corresponding type of games. Some of these libraries could be applied for a few types of games, so we will describe them only once.

The requirements to the libraries were the following:

- Is open source.
- Is maintained - we want to find a library that is supported and meets modern standards.
- Can be integrated with a client application - library should allow us to integrate with a client application without using any hack solutions.
- Is not overfit client application needs - we need a library that will do exactly what we want or close to it.
- Supports features we need to implement corresponding game type.
- It makes sense to use it instead of writing everything manually.
- Is admissible well documented - documentation is important to be able to work with the library correctly. We need to have at least basic documentation.

A huge number of different libraries was reviewed. We include only the most interesting and open source solutions which are most relevant to the application aims.

4.2.1 Cards.js

Cards.js [22] is a JavaScript library for playing cards. From the first view looks promising. But it has very limited documentation and visual representation is not ideal. Also, this library was written almost 7 years ago what is a very long period for the modern web world. And it is not maintained anymore.

4.2.2 Phaser

Phaser [23] is a JavaScript framework for developing desktop and mobile HTML5 games. It is open source and well documented. Phaser allows developing game using HTML5 Canvas and WebGL. It has a relatively big community and set of different plugins. Also, Phaser supports coding using TypeScript. It provides everything to build a game:

- WebGL & Canvas - Phaser uses both and can automatically swap between them based on browser support.
- Preloader - Images, Sounds, Sprite Sheets, Tilemaps, JSON data, XML - all parsed and handled automatically, ready for use in the game and stored in a global Cache for Game Objects to share.
- Physics - Phaser support 3 physics systems: Arcade Physics, an extremely lightweight AABB library perfect for low-powered devices. Impact Physics for advanced tile support and Matter.js - a full-body system with springs, constraints and polygon support.
- Sprites - allow to use sprites and do everything with them: rotate, drag, animate, scale etc. Also, provide support for different event handlers for sprites.
- Groups - allow to group sprites and apply things we can do with sprites to the whole group.
- Animation - Phaser supports classic Sprite Sheets with a fixed frame size as well as several common texture atlas formats including Texture Packer, Starling and Unity YAML. All of these can be used to easily create animations.

- Particles - a particle system is built-in, which allows you to create fun particle effects easily.
- Camera - Phaser has advanced multi-camera support.
- Input - Phaser support different input devices like Touch, Mouse, Keyboard, Gamepad.
- Audio - Phaser supports both Web Audio and legacy HTML Audio. It automatically handles mobile device locking, easy Audio Sprite creation, looping, streaming, volume, playback rates and detuning.
- Tilemaps - Phaser can load, render and collide with a tilemap with just a couple of lines of code. It supports CSV and Tiled map data formats with multiple tile layers.
- Device Scaling - Phaser has a nice build in scaling manager. It allows to make application looks well on all screen resolutions. Supports full screen.
- Plugin System - Phaser has a reach plugin system.

Phaser is actively maintained. It is a very good and flexible framework. Could be a nice basis for any game. But it is overfitting our needs. We do not need so complex tool for our games.

4.2.3 Card Game Library

Card Game Library [24] is a JavaScript library for building card games for the web. It was developed as a hackathon project. The library uses HTML DOM for visualization. It looks well, but it was developed 8 years ago. Has no documentation and not maintained anymore. Also on the webpage was mentioned that it is compatible only with Chrome web browser.

4.2.4 MelounJS

MelounJS [25] is a lightweight HTML5 framework designed from the ground and it is open source. MelounJS has the following features:

- A fresh & lightweight 2D sprite-based engine.
- Standalone library - do not need anything, but HTML5 compatible browser.
- Compatible with most major browsers and mobile devices.
- Device motion & accelerometer support.
- High DPI & auto scaling.
- Multi-channel HTML5 audio support and Web Audio on supported devices.
- Lightweight physics implementation to ensure low cpu requirements.
- Polygon (SAT) based collision algorithm for accurate detection and response.
- Fast Broad-phase collision detection using spatial partitioning.
- 3rd party tools support for physic body definition (PhysicEditor, Physic Body Editor).
- Advanced math API for Vector and Matrix.
- Animation with different effects.
- Object Pooling.
- Basic Particle System.
- Standard spritesheet and Packed Textures (Texture Packer, ShoeBox) support.
- A state manager - help to manage loading, menu, options, in-game state.
- Maltiple tools to work with tiles.
- Mouse and Touch device support with mouse emulation.
- Asynchronous messaging support.
- Basic GUI elements included.

It is a very good tool with a big community and support. MelounJS is actively developed. And we can use it for any type of 2D game. But it is too much for the games we want to visualize. It will well fit for bigger games with advanced graphics. Means that we can implement what we want using this framework or Phaser, but it will give as extra complexity we do not want to have.

4.2.5 HTML5 Deck of Cards

HTML5 Deck of Cards [26] is a pure JavaScript implementation of a deck of cards. This library allows to manipulate with card decks, shuffle them, show certain cards, add or remove cards from the deck. It uses HTML DOM elements for cards visualization. It has poor documentation. Also, it is not actively maintained now. The author mentioned that he plans to rewrite this library. This library was the main candidate to use it in the client application for card games, because of it simple and lightweight. But there is no big advantage of using this library against a code card deck by ourselves.

4.2.6 Summary

Based on reviewed solutions we decided to implement base components for card games by ourselves using HTML DOM elements. It seems to be the most logical decision. We do not need any extra features, and it will be easier and faster to write it by ourselves.

There were nice game development frameworks, but all of them are too complex for us. We do not need functionality that we will not use.

4.3 Board games

4.3.1 Crafty

Crafty [27] is a flexible open source framework for JavaScript games. Contains following features:

- Cross Browser compatible.
- Canvas or DOM - allows using canvas or DOM for rendering keeping logic the same for both cases.
- Entity Component System.
- Sprite Map support.
- Collision Detection.
- Fire & Forget Events - provide a flexible system to manage and handle events.
- Animation and effects support.
- Simple physics support.

Crafty is a very nice lightweight and well-documented framework. More suitable for dynamic games.

4.3.2 Babylon.js

Babylon.js [28] is a complete JavaScript framework for building 3D games and experiences with HTML5, WebGL, WebVR and Web Audio. Probably one of the biggest game development frameworks for the web. Has a very rich list of features. Babylon.js provides too much for the needs of the client application and was mentioned to demonstrate everything that could be done on the web.

4.3.3 Stage.js

Stage.js [29] is an open source lightweight 2D HTML5 JavaScript library for cross-platform game development. It provides DOM-like tree data model to compose application and internally manages rendering cycles and drawing. Stage.js allow to handle and process mouse and touch event on each element. Each node is pinned to its parent and can have any number of image textures. It has optimized redrawing logic. Stage.js provide a set of standard features like animation, scaling, positioning, etc.

Stage.js is a very simple, well-documented tool. It has exact features we need and do not have unnecessary complex functionality.

4.3.4 Enchant.js

Enchant.js [30] is an open source JavaScript framework for developing simple games and applications. It has the following features:

- Objected Oriented - all items displayed are objects.
- Event-driven - based on asynchronous processing via event listeners.
- Animation Engine - allows the use of standard animation such as tweens.
- Hybrid Drawing - supports drawing with both the Canvas API and the DOM.
- WebGL Support - supports 3D games using WebGL with a plugin.
- Content Library - includes a royalty-free image library which could be used in an application.

It was a very popular framework in the past. Enchant.js was released in 2011. There are over 1000 games implemented using this framework. Unfortunately, it not maintained for more than two years.

4.3.5 Boardgame.io

Boardgame.io [31] is an open source game engine for turn-based games. Beside visualization, it also cares about game definition. Boardgame.io provides interfaces to define turn-based games logic. It has the following features:

- State Management - the Game state is managed seamlessly across clients, server and storage automatically.
- Cross-platform Multiplayer - All clients are kept in sync in real-time.
- Declarative AI - Tell the bots what to do and they will figure out how to do it.
- Game Phases - with different game rules (including custom turn orders) per phase.
- Prototyping - Debugging interface to simulate moves even before you render the game.
- Logs - Game logs with the ability to time travel (viewing the board at an earlier state).
- View-layer Agnostic - Vanilla JavaScript client with bindings for React / React Native.
- Component Toolkit - Components for hex grids, cards, tokens.
- Extendable - Subsystems (storage, networking, etc.) can be replaced with custom implementations.

It has a nice way of game logic definition by separating it into different methods. Also, boardgame.io has good integration with React. The library relies more on client side logic definition. And has only basic HTML DOM based visualization capabilities. This is not acceptable for bigger board games. We decided to use a canvas to be able to scale the board.

Boardgame.io has integrated implementation of some game theory algorithms, but we already have this functionality in backend application. If the task were to implement the client-side only application for playing turn-based games from scratch, we could go for this solution as the core engine, but replace the visualization part.

4.3.6 Konva.js

Konva.js [32] is a very popular open source HTML5 2d canvas library for desktop and mobile applications. It allows to interact with HTML5 canvas API in a comfortable object-oriented performant way, handle and process events and is simple to use. Konva.js has the following features:

- Built-in in support for HDPI devices with pixel ratio optimizations for sharp text and shapes.
- Object Oriented API.
- Node nesting and event bubbling.
- High performance event detection via color map hashing.
- Layering support.
- Node caching to improve draw performance.
- Nodes can be converted into data URLs, image data, or image objects.
- Animation support.
- Tween support.
- Drag and drop with configurable constraints and bounds.
- Filters.
- Ready to use shapes including rectangles, circles, images, text, lines, polygons, SVG paths, and more.
- Custom shapes.
- Event driven architecture which enables developers to subscribe to attribute change events, layer draw events, and more.
- Serialization & de-serialization.
- Selector support - we can select element in CSS selectors like way.
- Desktop and mobile events.
- Custom hit regions.

Konva.js has a big community, lots of examples and good documentation. It has rich API which gives the ability to customize every element and have full control over what we want to achieve.

React Konva [33] is a JavaScript library for drawing complex canvas graphics using React. It provides declarative and reactive bindings to the Konva Framework. Using this library, we can have full control of Konva.js in React-like style. The goal of this library is to have similar, declarative markup as normal React and to have a similar data-flow model.

4.3.7 Summary

In this section we skipped many game developing frameworks. Almost all of them are really great and provides a big number of different functionalities out of the box. But all of them overfitting our needs. They could be useful for big and complex games or real-time arcade games.

We decided to use Konva.js and React Konva as an adapter for React in frontend application. It matches all requirements and give us flexibility, good control on application and allows to keep components separately without any additional manipulation to be able to reuse them in the same environment.

4.4 Graph games

For graph games, we wanted to find a library which will be able to visualize graph, support different graph layouts with auto positioning, allow to select nodes and handle events. Optional, but also important requirement is a clean look.

4.4.1 Sigma.js

Sigma.js [34] is a JavaScript library dedicated to graph drawing. It makes easy to publish networks on Web pages and allows developers to integrate network exploration in rich Web applications. Sigma.js provides the following features:

- Custom rendering - allow to use the Canvas or WebGL built-in renderers or write own. And the built-in renderers also provide a lot of ways to already customize the rendering.
- Interactivity oriented - allow to handle and process different events like click, drag, zoom, etc.
- Powerful graph model - sigma graph model is customizable. It allows adding custom indexes on the data.
- Extendable - it is easy to develop plugins or simple snippets to augment sigma's features. Some are already available in the main repository to read some popular graph file formats, or to run complex layout algorithms, for instance.
- Compatibility - sigma runs on all modern browsers that support Canvas and works faster on the browser with WebGL support.

Sigma is simple, nice looking and actively developed library.

4.4.2 D3.js

D3.js [35] is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

D3.js allows to bind data to DOM and then apply data-driven transformations to the document. It provides a huge set of different data manipulation, transformation, and visualization tools. Library provides custom data structures and able to visualize data in different forms like box plots, various types of charts, calendars, dendrograms, direct graphs, stacked bards, Voronoi diagrams, scatterplot matrices, maps, sequences sunburst and many others.

D3.js is a very great tool to visualize data and is must have tool when developing any graphical statistics dashboard. Has big community, documentation and examples. But our aim is different. Data visualization is an only small part of the client application. We need to be able to interact with graphs.

4.4.3 Cytoscape.js

Cytoscape.js [36] is graph theory library for visualization and analysis. Part of Cytoscape platform for visualizing complex networks and integrating these with any type of attribute data. Cytoscape.js allows to display and manipulate rich, interactive graphs and hook into user events. It includes different types of gestures out-of-the-box, including pinch-to-zoom, box selection, panning, etc. Cytoscape.js also has graph analysis in mind: The library contains many useful functions in graph theory.

Cytoscape.js has the following features:

- A fully featured graph library is written in pure JavaScript.
- Permissive open source license (MIT) for the core Cytoscape.js library and all first-party extensions.
- Highly optimized Compatible with all modern browsers, legacy browsers with ES5 and canvas support, module systems and package managers like npm, yarn, bower.
- Has a large suite of tests that can be run in the browser or the terminal.
- Documentation includes live code examples, doubling as an interactive requirements specification.
- Fully serializable and deserializable via JSON.
- Uses layouts for automatically or manually positioning nodes.
- Supports selectors for terse filtering and graph querying.
- Uses stylesheets to separate presentation from data in a rendering agnostic manner.
- Abstracted and unified touch events on top of a familiar event model.
- Built-in support for standard gestures on both desktop and touch.
- Supports functional programming patterns.
- Supports set theory operations.
- Includes graph theory algorithms, from BFS to PageRank.
- Animatable graph elements and viewport.
- Fully extendable.
- Well maintained.
- Used in various well-known projects.

As it was mentioned, Cytoscape.js has very good documentation with lots of examples. It gives full control of the graph and its parts events. It is a highly customizable and optimized library.

4.4.4 React-digraph

React-digraph [37] is a React component which makes it easy to create a directed graph editor without implementing any of the SVG drawing or event handling logic.

React-digraph is an open source solution from Uber. It allows to create and edit graphs on canvas. It works and looks good. Using this component, we can simply manipulate with the graph, adding and removing nodes, edges and labels. But this is a graph editor. It helps to create a graph, but we need to use an existing graph and interact with it.

4.4.5 Vis.js

Vis.js [38] is a dynamic, browser-based visualization library. The library is designed to be easy to use, to handle large amounts of dynamic data, and to enable manipulation of and interaction with the data.

The library is logically separated into the following parts:

- DataSet - Manage unstructured data using DataSet. Add, update, and remove data, and listen for changes in the data.
- Timeline - Create a fully customizable, interactive timeline with items and ranges.
- Network - display dynamic, automatically organized, customizable network views.
- Graph2d - Draw graphs and bar charts on an interactive timeline and personalize it.
- Graph3d - Create interactive, animated 3d graphs. Surfaces, lines, dots and block styling out of the box.

We are interested in the Network part. The Network is a visualization to display networks and networks consisting of nodes and edges. The visualization is easy to use and supports custom shapes, styles, colors, sizes, images, and more. The Network has clustering support. The Network uses HTML canvas for rendering.

Network part allows stylising nodes and edges, adding labels, using different layouts and handle element's events. Library provides an API to dynamically manipulate with data and supports animations and physics. It is an open source solution with good documentation, examples, and support.

4.4.6 Summary

We have been evaluated all options, few of them were suitable for application needs, but we decided to choose cytoscape.js. It specializes in graph manipulation and interactions. Provides wide well-documented APIs to for interactions and gives full control on the graph. It used in many different projects and has good feedback.

Chapter 5
Server-side application implementation

In this chapter, we will describe implementation details of the back end application.

5.1 Application structure

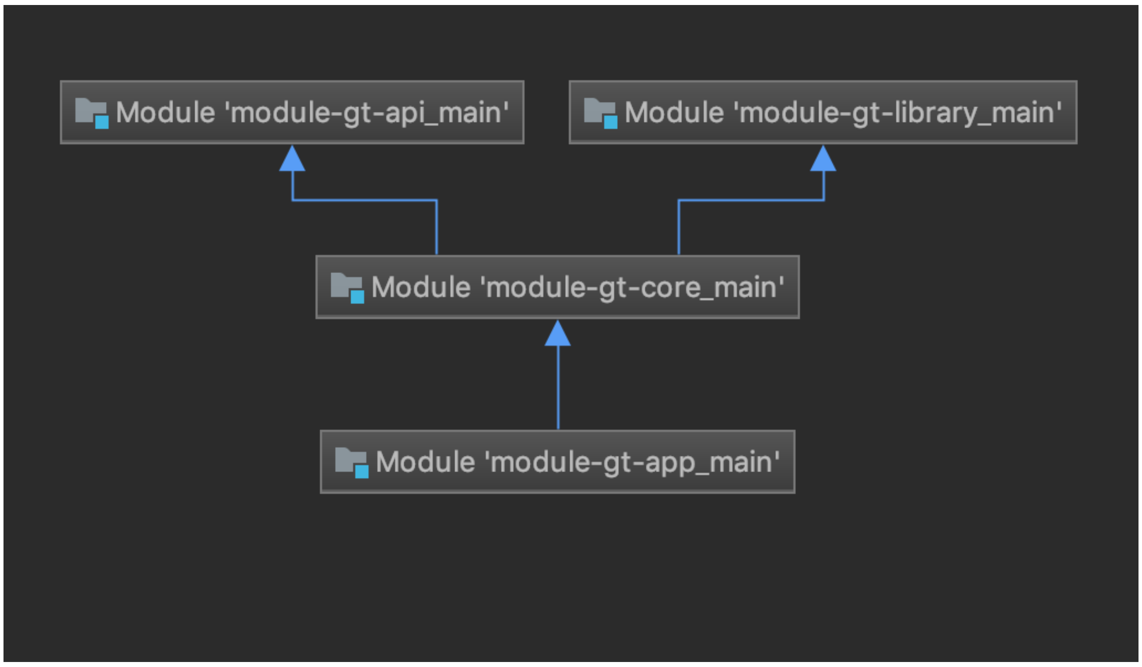

Module 'module-gt-api_main'
Module 'module-gt-library_main'
Module 'module-gt-core_main'
Module 'module-gt-app_main'

Figure 5.1. Back end application module dependencies.

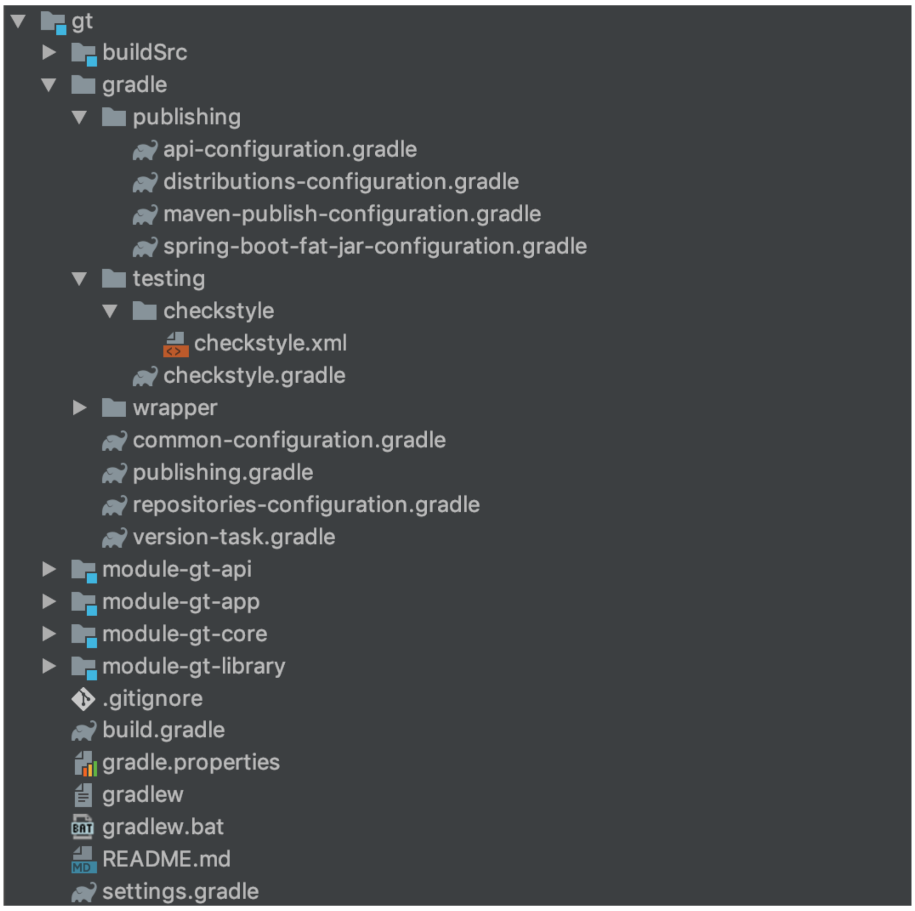
Back end application is separated into four modules:

- **Module API** - contains DTO (Data Transfer Object) classes. These classes are used for interactions with the outer infrastructure. They are defining request and response content format. Using these structures front end and back end are communicating with each other. Keeping such DTOs in a separate module, which is built in its own jar file is a good practice. It could be reused in other applications for interaction.
- **Module GT Library** - contains game theoretic library implementation [1]. Because of this library required additional dependencies which are not abled from maven repositories and CPLEX solver from IBM, we decided to put it in a separate module and automate these processes. It is possible to specify the path to the CPLEX binaries in `gradle.properties` file. Another reason to keep the library as part of the application is that we needed to make small changes in it to be able to execute game playing algorithms simultaneously.
- **Module Core** - contains main part of the code of this work. It is an adapter for the game theoretic library. The core logic is implemented here. It takes care about

receiving a request, converts it, process with game theoretic library and responds back. Code from this module is responsible for a lot of logic and is described more in details in Section 5.2.

- **Module App** - is a wrapper for a whole back end application. It contains the main application class, different configuration files and integration tests for the back end application. In this module, we are able to configure algorithms parameters in a comfortable, structured manner, application attributes, and logging templates.

Figure 5.1 demonstrates module dependencies.



```
▼ gt
    ▶ buildSrc
    ▼ gradle
        ▼ publishing
            api-configuration.gradle
            distributions-configuration.gradle
            maven-publish-configuration.gradle
            spring-boot-fat-jar-configuration.gradle
        ▼ testing
            ▼ checkstyle
                checkstyle.xml
            checkstyle.gradle
        ▶ wrapper
        common-configuration.gradle
        publishing.gradle
        repositories-configuration.gradle
        version-task.gradle
    ▶ module-gt-api
    ▶ module-gt-app
    ▶ module-gt-core
    ▶ module-gt-library
    .gitignore
    build.gradle
    gradle.properties
    gradlew
    gradlew.bat
    README.md
    settings.gradle
```

Figure 5.2. Back end application project structure.

In Figure 5.2 project structure is shown. It is important to know where different parts of the project are located.

- `buildSrc` - contains version utilities which are used for versioning during the build.
- `gradle` - contains Gradle configuration scripts.
- `gradle/publishing` - contains configurations for build artifacts depending on project type such as API project or fat jar project. Also, there is a configuration for publishing artifacts to the nexus.
- `gradle/testing/checkstyle` - contains checkstyle plugin integration to Gradle and its configuration. This plugin is used to control code style rules. Described in details in Section 8.1.
- `gradle/wrapper` - contains Gradle wrapper jar which allow us to execute build without Gradle installed on the operating system by using `./gradlew` from the root directory.

- module-gt-api, module-gt-core, module-gt-library and module-gt-app - contains the modules described above.
- build.gradle - main build file. Contains common configuration for all projects (modules). Additionally, every module has its own build.gradle file with a specific configuration.
- gradle.properties - contains properties which Gradle use during the script execution. It could be not managed libraries versions, application build version, Gradle attributes, and limitations. The path to CPLEX should be defined here as well by setting cplexJarPath and cplexBinPath properties for the path to CPLEX jar file and binary files respectively.

```
...
cplexJarPath=/path/to/jar/cplex.jar
cplexBinPath=/path/to/binaries/cplex/bin/x86-64_osx
...
```

Listing 5.1 CPLEX path configuration in gradle.properties.

5.2 Game theoretic library integration

We have to implement an adapter to the game theoretic library with minimum changes to the library itself and keeping backward compatibility.

The game theoretic library was initially developed to test game-playing algorithms. For these purposes, an application should be executed with predefined parameters and after the evaluation is complete, a user has to receive the result and application is going to shut down. From this perspective for configuration were used static variables. This fact brings some problems for algorithms execution with different configurations and states in parallel on a single JVM. We have to improve it to be able to execute algorithms on single JVM instance.

Only a few changes were made in module-gt-library. We added additional constructors to the game classes to be able to pass them dynamic configuration. After these changes game-theoretic library still could be used in an old manner, but now we could pass there our custom configuration. Unfortunately, we have to make additional customizations for some game info and state classes by using inheritance to be able to override some methods to use provided configuration instead of a static one. For example, we implemented a class CustomGoofSpielGameState that extends GoofSpielGameState class and overrides its methods that are using static variables. It brings us small logic duplication, but there is no other more acceptable way to do this. All these classes are located in module-gt-core.

We wanted to provide a structural way for loading games and algorithms independently on each other. For example, the game-theoretic library has implemented different algorithms and games, and they are working with this back end application. When the new game will be added to the library, we don't want to care about integration with all algorithms. Only some game-specific logic should be added to the application, and it will work with all already supported algorithms. Same should work for algorithms as well. If the new algorithm was added to the library, it should work with all supported games.

Because library features exceed application features and from a style perspective, two enumeration classes were defined. First is a game(cz.cvut.fel.gt.game.Game).

```
public enum Game {

    GS("GoofSpiel"),
    PE("Pursuit-Evasion"),
    OZ("Oshi-Zumo");

...

}
```

Listing 5.2 Game enumeration definition.

And the second one is an algorithm(`cz.cvut.fel.gt.game.Algorithm`).

```
public enum Algorithm {

    BI("Backward induction"),
    BIAB("Backward induction \u03B1\u03B2"),
    DO("Double-oracle"),
    DOAB("Double-oracle \u03B1\u03B2"),
    OOS("Online Outcome Sampling"),
    MCTS_UCT("MCTS Upper Confidence Bounds for Trees"),
    MCTS_EXP3("MCTS EXP3"),
    MCTS_RM("MCTS Regret Matching"),
    RAND("Random");

...

}
```

Listing 5.3 Algorithm enumeration definition.

To add support of new game to the back end application, we need to add game loader component 2.4.3. This component must implement `GameLoader` interface.

```
public interface GameLoader {

    Game getGame();

    SingleGameState<MCTSInformationSet> loadMCTSGame();

    SingleGameState<MCTSInformationSet> loadMCTSGame(GameInfo gameInfo);

    SingleGameState<SimABInformationSet> loadSMGame();

    SingleGameState<SimABInformationSet> loadSMGame(GameInfo gameInfo);

}
```

Listing 5.4 `GameLoader` interface.

We must mention that game-theoretic library uses three main components to describe a game domain:

- `GameInfo` interface implementation. This interface provides information about the game configuration.

- `GameState` interface implementation. This interface provides information about the current game state.
- `Expander` interface implementation. This interface allows us to expand current game state and get available actions for the current player. This interface also keeps information about algorithm configuration and its information set. The information set contains information about the single player and game states for this player.

`GameLoader` interface has 5 methods:

- `getGame()` method is a helper function which returns us the name of the game.
- `loadMCTSGame(GameInfo gameInfo)` method creates a game instance from configuration provided in `GameInfo.class`. `SingleGameState<InformationSet>` is a wrapper class. It contains game information (`GameInfo.class`), root game state(`GameState.class`) and expander(`Expander.class`). By calling this method expander will be initiated with `MCTSConfig`. For example, for Oshi-Zumo game this method will return `SingleGameState<MCTSInformationSet>` instance containing `OshiZumoGameInfo` instance, `OshiZumoExpander` instance and `CustomOshiZumoGameState` instance.
- `loadMCTSGame()` method works the same as `loadMCTSGame(GameInfo gameInfo)`, but since `gameInfo` is not provided, the game will be initialized with default configuration.
- `loadSMGame(GameInfo gameInfo)` method has similar functionality and aim as `loadMCTSGame(GameInfo gameInfo)`, but expander will be initialised with `SimABConfig` instead of `MCTSConfig`. Such segregation helps us to support algorithms with different information set types.
- `loadSMGame()` method works the same as `loadSMGame(GameInfo gameInfo)`, but the game will be initialized with default configuration.

To add support of the game playing algorithm to the back end application, we need to add algorithm loader component 2.4.3. This component must implement `AlgorithmLoader` interface.

```
public interface AlgorithmLoader {

    Algorithm getAlgorithm();

    GameInstance load(Game game, int posIndex);

    GameInstance load(Game game, int posIndex, GameInfo gameInfo);

}
```

Listing 5.5 `AlgorithmLoader` interface.

This interface cares about loading everything we need to be able to run a specific game playing algorithm for a certain game.

- `getAlgorithm()` method provides information about the algorithm.
- `load(Game game, int posIndex)` method return `GameInstance.class` instance. This is a wrapper class which contains `SingleGameState.class` instance and `GamePlayingAlgorithm.class` instance. This method loads the game using `GameLoading` interface based on its type and initializes all things algorithm needs to work. Input parameters are name of the game we want to load and player's position.

- `load(Game game, int posIndex, GameInfo gameInfo)` method has the same logic as `load(Game game, int posIndex)` but allows to pass game configuration information.

Back end application has implementations of `GameLoader` and `AlgorithmLoader` interfaces for supported games and algorithms. Furthermore, algorithm loader could have the same code inside for similar algorithms. For this reason, there are abstract classes which could be reused for new game playing algorithms integration. Figure 5.3 shows a hierarchy of algorithm loaders.

Some of algorithm configuration parameters could be set in `application.yaml` instead of system properties as it was before. This file is located in `module-gt-app` in `resource` directory. For example, we can specify a time limit for algorithm execution and use it in code by via `AlgorithmConfiguration` interface.

```yaml
...
algorithm:
   configuration:
      comp-time: 500
...
```

Listing 5.6 Algorithm specific properties configuration.

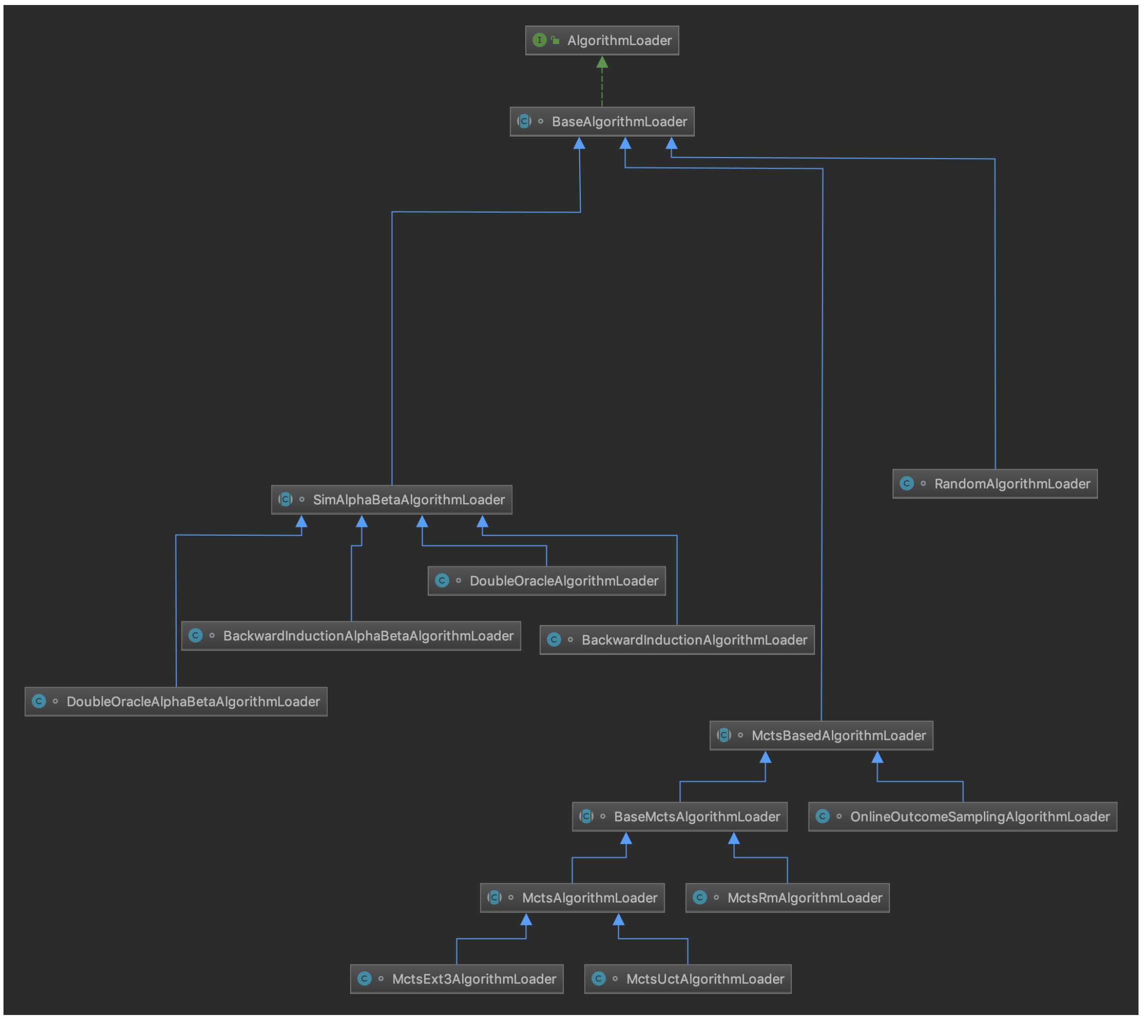

AlgorithmLoader
BaseAlgorithmLoader
RandomAlgorithmLoader
SimAlphaBetaAlgorithmLoader
DoubleOracleAlgorithmLoader
BackwardInductionAlphaBetaAlgorithmLoader
BackwardInductionAlgorithmLoader
DoubleOracleAlphaBetaAlgorithmLoader
MctsBasedAlgorithmLoader
BaseMctsAlgorithmLoader
OnlineOutcomeSamplingAlgorithmLoader
MctsAlgorithmLoader
MctsRmAlgorithmLoader
MctsExt3AlgorithmLoader
MctsUctAlgorithmLoader

Figure 5.3. Algorithm loaders hierarchy.

Next part of the application is a `GameSpecificationService` interface.

```
public interface GameSpecificationService {

    GamesSpecificationDTO getGamesSpecification();

    GameConfigurationsDTO getGameConfigurations(Game game);

}
```

Listing 5.7 `GameSpecificationService` interface.

This interface is responsible for providing information about the games. It was created to let the client know what kind of games and algorithms are exists. And in what configuration player is able to play single games.

- `getGamesSpecification()` method returns a list of all games with main game attributes. Every item of this list contain the following fields:
  - The name of the game
  - An alias - the name of the game from `Game` enumeration. For example, OZ, GS, PE.
  - Type of the game. Should be one of the board, graph or card values.
  - List of player's roles.
  - List of roles which human can play. For example, in Pursuit-Evasion game player are able to choose between evader and pursuer.
  - List of supported algorithms.
  - Parameters map - additional optional parameters which could be specified for a single game. For example, we could specify the graph type for graph games.
- `getGameConfigurations(Game game)` method provides configurations for the provided game. Each configuration contains different game parameters, like game field definition, players start positions, goals and many other attributes based on the game.

`GameService` interface is the main part of the back end application. It has only one method execute which receives an algorithm and request from client process it and return a result with opponent actions and new game state. This service manages player move evaluation and executes game-playing algorithm to receive new game state and opponent action. In Figure 5.4 we can see the simplified version of sequence diagram based on implementation for Oshi-Zumo game. In simple words we are getting algorithm configuration parameters, loading game, loading algorithm runner, setting current game state, executing player action and then run algorithm evaluation to get an opponent move and new state. In the end, we should put all the required information into the response DTO instance. The developer can do it based on specific domain requirements by adding functionality he wants.

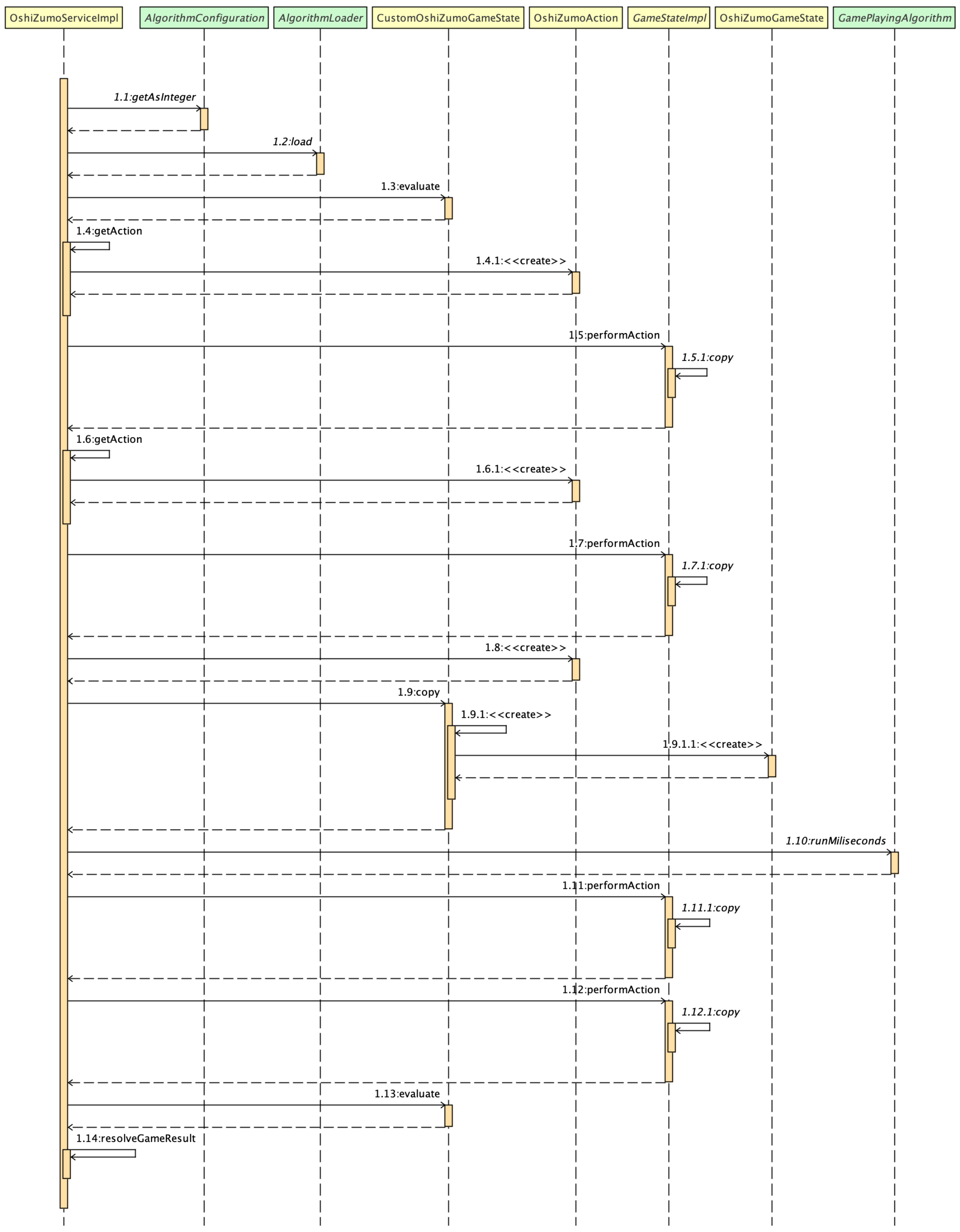


Figure 5.4. `GameService` execution sequence diagram. Oshi-Zumo implementation.

5.3 Summary

To sum up this section we described steps which should be done to add a new game and game playing algorithm.

To add support of new game playing algorithm developer should do the following steps:

- Add algorithm to the `Algorithm` enumeration class from `cz.cvut.fel.gt.game` package.
- Create Spring component implementing `AlgorithmLoader` interface from package `cz.cvut.fel.gt.loader.algorithm`.

To add support of new game developer should do the following steps:

- Add game to the `Game` enumeration class from `cz.cvut.fel.gt.game` package.
- Create Spring component implementing `GameLoader` interface from package `cz.cvut.fel.gt.loader.game`.
- Create Spring component implementing `GameService` interface from package `cz.cvut.fel.gt.service` and all required objects which would be required, such as DTOs, game states, players, etc.
- Create Spring component implementing `GameSpecificationService` interface from package `cz.cvut.fel.gt.service` to provide information about game and possible configurations.
- Create a class implementing `GameInfo` interface from game theoretic library to be able to use custom configuration for each simultaneously running game.
- Create a controller to handle requests.

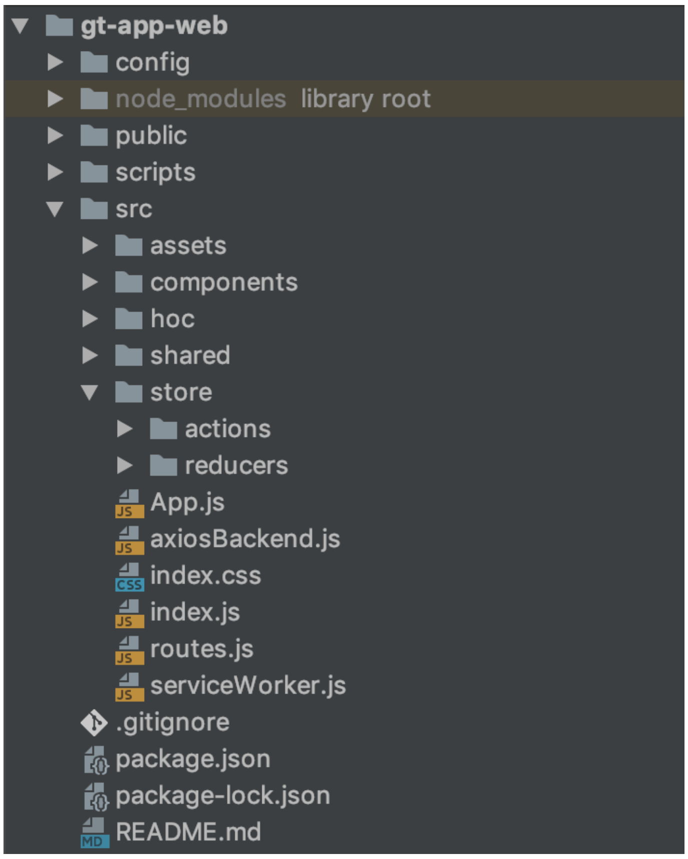
Some game implementations require custom game state which is usually extended from the game state from the game theoretic library, to provide non-static configuration into the game.

Chapter 6
Client-side application implementation

In this chapter, we will describe implementation details of the front end application.

Front end application implementation is built with React.js library. For package managing we used npm, but it is possible to use yarn or npx, since they also can read `package.json`. To build an application, we are using webpack. These technologies were described in Section 2.5.

6.1 Application structure



```
▼ gt-app-web
  ▶ config
  ▶ node_modules  library root
  ▶ public
  ▶ scripts
  ▼ src
    ▶ assets
    ▶ components
    ▶ hoc
    ▶ shared
    ▼ store
      ▶ actions
      ▶ reducers
    App.js
    axiosBackend.js
    index.css
    index.js
    routes.js
    serviceWorker.js
  .gitignore
  package.json
  package-lock.json
  README.md
```

Figure 6.1. Client application project structure.

In Figure 6.1 we can see the client application project structure. It is important to orientate in it to be able to configure and extend the application.

- `config` directory contains configuration files for webpack production and development build, paths definitions, environment configuration, and development server configuration.

- `node_modules` directory contains sources of all required dependencies from `package.json`. It does not exist in the project by default, but npm will create it after calling `npm install` to store required dependencies.
- `public` directory contains `index.html` file which serves as entry point for React application and manifest file.
- `scripts` directory contains npm scripts for running production build, start developing server and running tests.
- `src` directory contains all application source codes:
  - `src/assets` directory contains image files used in the application.
  - `src/components` directory contains React components and their CSS module files. CSS modules are used to limit the scope of styles to a single component. When webpack is building sources, it automatically hashes style classes to make a unique name.
  - `src/hoc` directory contains higher order components with special functions such as `Layout` for managing main application layout, or `AsyncComponent` which could wrap normal component to make component and all its dependencies loading asynchronously. By using `AsyncComponent` webpack will split resulting JavaScript file to smaller bundles and then the web browser will be able to download only required parts of the application.
  - `src/shared` directory contains shared utility helper functions.
  - `src/store` directory contains Redux action definitions and reducers. It is described more in details below.
  - `src/App.js` is client application main component. `App` component wraps other components and logic.
  - `src/axiosBackend.js` contains default configuration and instantiation of axios client. In this file we can set default HTTP request attributes such as header. Also, in this file default back end application address is defined.
  - `src/index.css` contains global application CSS styles.
  - `src/index.js` is used to initialize `App` component, router, Redux, and other middleware.
  - `src/routes.js` contains application routes.
- `package.json` contains all required dependency definitions for the client application. After installation, all these dependencies will be stored in `node_modules` directory.

In structure description, we mentioned few libraries. Next, we will describe them and their role in the application.

Axios is a Promise based JavaScript HTTP client. It allows us to control HTTP requests in a convenient way. Using axios, we can configure default request attributes which would be applied to every request. We can manage the lifecycle of the request, by setting timeouts and handling positive and negative scenarios. Axios provides universal API for all web browsers.

React router is a JavaScript routing library which allows to load and unload React components based on browser URL. Also, it provides API for manipulating with browser history. It helps to show different pages for different URLs and handle history state change, for example, with browser back and forward buttons click.

As was mentioned in Section 4.1 React component could have its own local state. Definition of the state is very important. A state could describe the whole current situation. Using the state design pattern makes application logic very clear and transparent. It helps us without extra manipulations pass current game state to the backend application to be evaluated by game playing algorithm. Because React shares the same principle, it becomes a good choice.

React component state has local scope and share the scope or its parts, become complicated for not child components. The solution to this problem is Redux.

Redux is a predictable state container for JavaScript apps [14]. Redux is mainly a library responsible for issuing state updates and responses to actions. In the beginning, the Redux store is created. This store holds global application state, Redux then able to manage this state using reducers and actions. Components are able to subscribe to required state properties and receive them as normal component properties.

Redux sticks to these three principles:

- Single source of truth - The global application state is stored in an object tree within a single store.
- The state is read-only - The only way to change the state is to emit an action, an object describing what happened.
- Changes are made with pure functions - Pure reducers are used to specify how the state tree is transformed by actions.

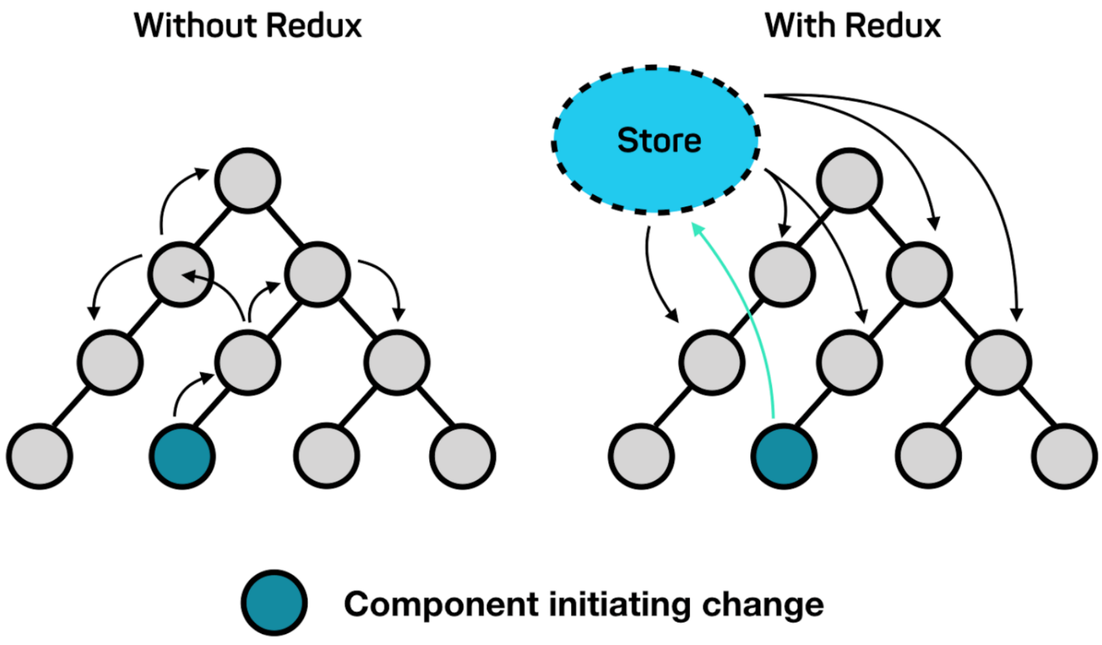


Figure 6.2. Redux work principle.

6.2 Game loader

To be able to load the game we need to know what kind of games are exists and what configurations do they have. For these purposes, we were implemented `GameLoader` component.

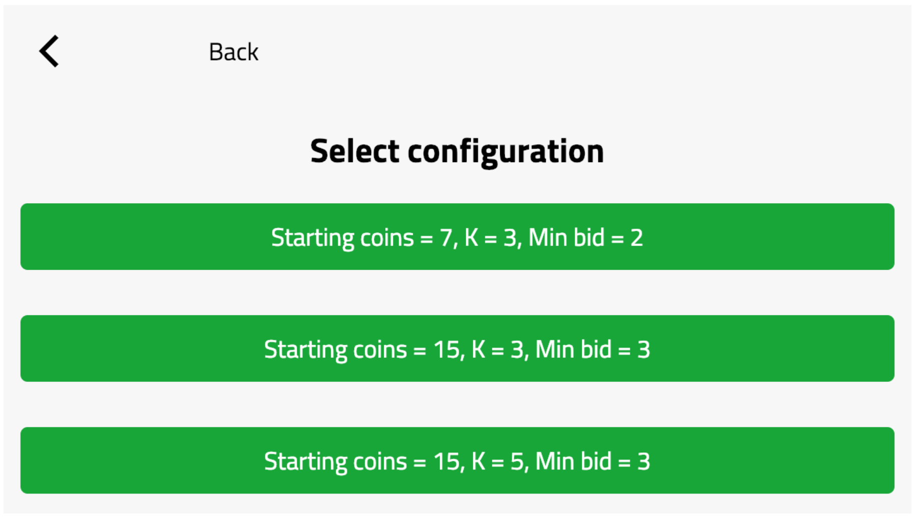

Back
Select configuration
Starting coins = 7, K = 3, Min bid = 2
Starting coins = 15, K = 3, Min bid = 3
Starting coins = 15, K = 5, Min bid = 3

Figure 6.3. `GameLoader` component step.

It was made using a wizard design pattern. `GameLoader` is separated into four steps. Figure 6.3 demonstrates one of these steps.

- The first step allows us to choose the game.
- The second step allows us to choose a player's role.
- The third step allows us to choose the game configuration.
- The fourth step allows us to choose a game-playing algorithm.

`GameLoader` automatically manages the game loading process from information provided by backend application endpoints `/specifications/{game_name}` and `/specifications`. It means that when we add new game implementation support to the backend application, we do not need to do any additional changes to `GameLoader`.

`GameLoader` also provides Back and Forward buttons to navigate between single steps when it is possible. `GameLoader` can skip one of the steps it has zero or only one option. If there is only one option provided, the component will automatically choose it without showing it to the user.

When last step's option will be selected, `GameLoader` will combine selected data and store it in the global state using Redux actions. When action completes, the player will be able to play selected game.

6.3 Game component and higher order wrapper component

A higher order component is an advanced technique in React for reusing component logic. Higher order components could wrap other components and provide them additional functionality.

`Game` higher order component provides additional functionality to game implementation. It accepts two parameters first is a single game implementation component and the second one is game configuration.

```
game(BoardGame, gameConfiguration);
```

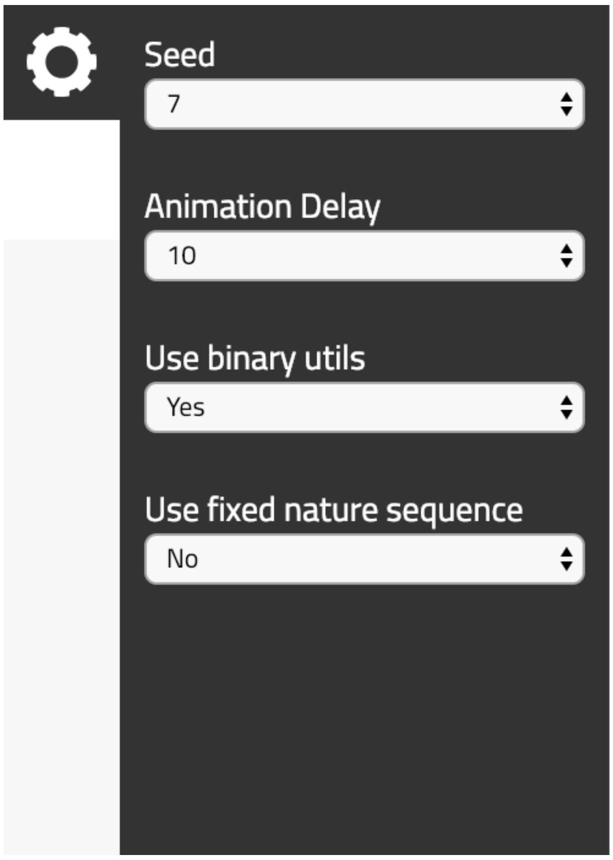
Listing 6.1 Wrapping `BoardGame` component with the higher-order `Game` component.

The game configuration could contain any configurable property for the game. For example, it could be a seed or animation delay. Game configuration is an array of objects with the following structure:

```
{
    name: 'binaryUtilities',
    label: 'Use binary utils',
    value: true,
    type: 'bool',
    options: {
        'true': 'Yes',
        'false': 'No'
    }
}
```

Listing 6.2 Single game configuration item structure.

- `name` is an attribute name.
- `label` used as a label in the user interface.
- `value` is a default value for an attribute.
- `type` is an attribute type. It used to convert and store a value in the correct data type.
- `options` are values range which could be selected.



Seed: 7
Animation Delay: 10
Use binary utils: Yes
Use fixed nature sequence: No

Figure 6.4. Game configuration sidebar.

`Game` higher order component has its own state. Which is initialized from the game configuration, moves array, and game status attributes such as `gameEnd` and `win`. It also provides properties and methods to the wrapped component and automatically appends some user interfaces.

Figure 6.4 demonstrates us such interface for game configuration. It could be opened by clicking on the gear wheel icon in the top right corner of the application screen.

`Game` component provides a method to store players actions. Moves history is a required attribute for some games and also could be used for debugging and monitoring a game process.

One of the main features of the `Game` component is a method provided to the wrapped component for communication with backend application. This method acts as an interceptor for requests. It automatically updates request with game configuration and moves history, so when we are implementing single game logic, we do not need to care about this part of application logic. Also, when we are receiving a response from the server it extracts information about game end status and if it is the end of the game, popup screen with the game result will be automatically displayed and the developer does not need to care about it as well.

This part of `Game` higher order component automatically displays loader which locks the screen while processing a request and shows user visual information that request is processing and the user should wait.

Another feature is a requests failure handling. For example, when a user has network problem and request will not be able to complete, the user will receive a graphical notification popup in the right bottom corner with information about a problem and two options - retry and cancel. By choosing retry action application will try to resend request again so that game state will not be lost, and the user will be able to continue playing.

Usually, React component has to implement `Component` class provided by React. We added `GameComponent` class which extends `PureComponent` class and could be used for game components implementation. It provides additional functionality for wrapped by higher order `Game` component.

`GameComponent` add properties validation and trigger new game start method, provided by `Game` component automatically. Using this component is not necessary, but it automates some game lifecycle methods and defines base game functions.

6.4 Oshi-Zumo implementation

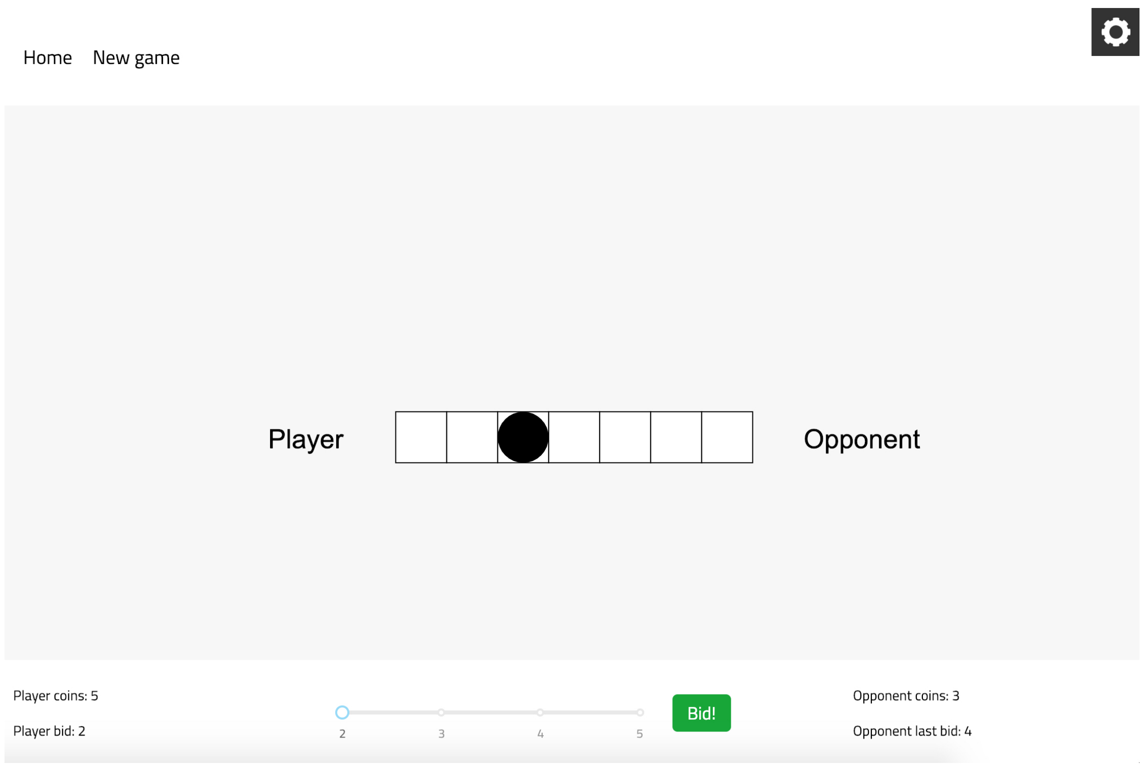


Figure 6.5. Oshi-Zumo game visualization.

Every game could be separated into a few logical blocks:

- Start a new game - initial game state initialization.
- Make first player action - contains logic which handles the first player move. This functionally allows the human player to make an action using a graphical user interface.
- Make second player action - contains logic which handles the second player move. Collects all required data and makes a request to the server to get an opponent action.
- Game end - contains logic which handles the end of the game.

This pattern was used in all game components implementation.

Oshi-Zumo is a board game. Because of this, we implemented the `Board` component using React Konva.

This component uses a `Stage` component from React Konva to define stage parameters and bind drag and zoom event handlers. `Stage` also accepts default offsets to put the resulting board to the screen center.

To draw a board, we used the `Rectangle` component with precalculated positions.

`Board` component accepts the following properties:

- `cellSize` is a size of board cell. Could accept any positive numeric value.
- `rows` is a number of rows of the board. Could accept any positive integer value.
- `cols` is a number of columns of the board. Could accept any positive integer value.
- `elements` is a `Map` of additional elements, where the key is a position in cell coordinates and value is an array of components to be displayed. For example, we pass wrestler using this property.

In Oshi-Zumo implementation, `Board` component was used the following:

```
<Board cellSize={50}
       rows={1}
       cols={configuration.locK * 2 + 1}
       elements={boardElements}/>
```

Listing 6.3 `Board` component usage example.

In Figure 6.5 we can see the implementation result. The black circle on the board is a wrestler. When the game starts, the wrestler position is in the middle of the board.

On the bottom of the screen, we have an action bar. It displays a number of player's coins and currently selected player's bid on the left side and amount of opponent's coins and last opponent bid on the right side. In the middle, we have placed a tool which player use to make an action. It shows possible player's bid values. A player can pick them by clicking on it. To confirm an action player needs to click on the Bid button. After player confirms his bids, the game will collect this data and will send it to the server to evaluate an opponent's action.

6.5 GoofSpiel implementation

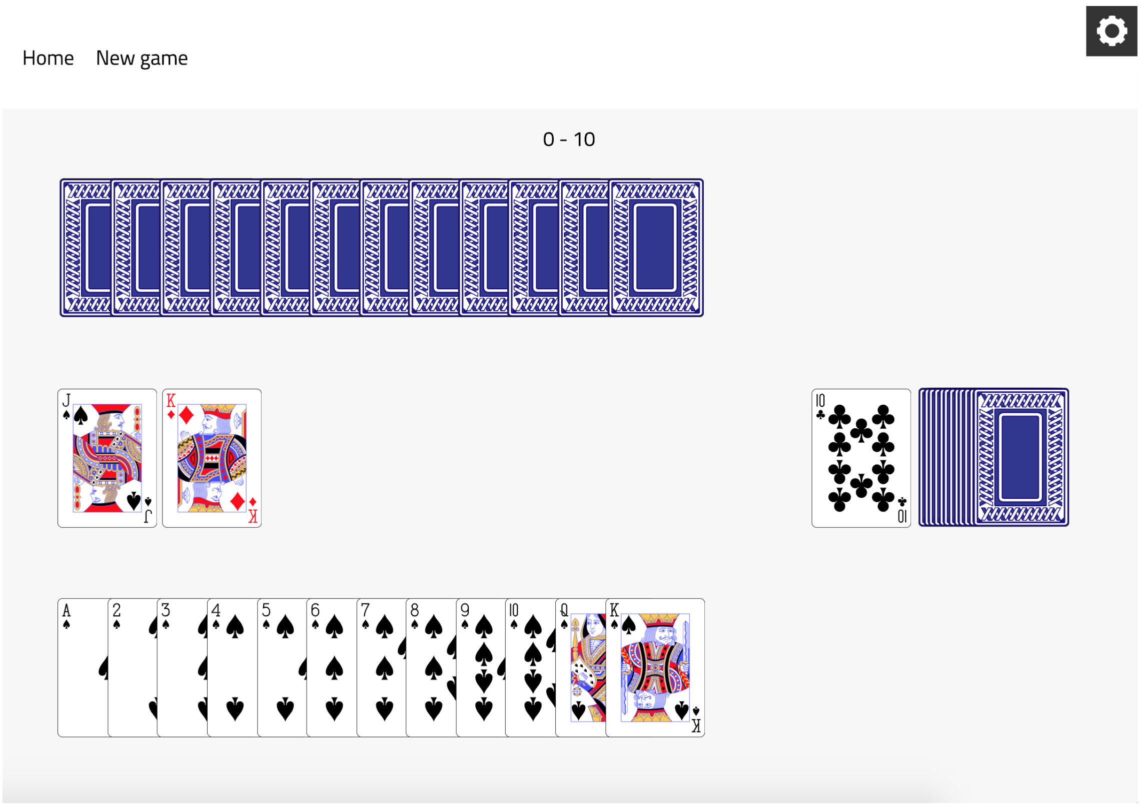


Figure 6.6. GoofSpiel game visualization.

GoofSpiel is a card game. Every card game has two main components card and hand. We implemented them in `Card` and `Hand` components.

We decided to use HTML DOM elements for the implementation. It was a good option to compare with other approaches. And we can see the result in Figure 6.6.

Card component has the following parameters:

- suit is a card suit. Can be one of the C, D, H, S values. Where C is clubs, D is diamonds, H is hearts, S is spades.
- rank is a card rank. Can be a number from 1 to 13. Where 1 is an Ace, 11 is a Jack, 12 is a Queen, 13 is a King, other numbers have the same value.
- selectable is a flag to set card selectable or not.
- open is a flag which determines if the card is open and we can see it, or it is turned by the back.

Also, card has standard react attributes such as onClick and style. Using style attribute, we are defining card position and size related to game table size. The Card can be used as follows:

```
<Card suit="H"
      rank={12}
      style={cardStyle}
      onClick={() => props.onCardSelectHandler(card)}
      selectable={true}
      open={false}/>
```

Listing 6.4 Card component usage example.

Hand component is a set of Card components. It corresponds to a player's hand. We can call it a container for player's cards. This component was developed for GoofSpiel game care about cards positions and sizes relative to the component itself. It has the following parameters:

- suit is a suit of player's hand cards. Could be one of the C, D, H, S values. It is the same as for Card component.
- open is a boolean flag. Can be either true or false. It determines if a human player can see the card or not.
- position is a card deck position on the playing table. Could be one of the 1, 2, 3 value. Where position 1 is an upper left corner, 2 is a center-right side, 3 is a bottom left corner. Hand positioning is absolute related to the parent element.
- compact determine if cards should be displayed in compact mode or wide. In compact mode, cards are positioned closer to each other. Could be either true or false.
- cards is an array of card ranks. It defines the ranks of the cards in player's hand.
- selectable defines if the player can select one of the cards or not. Could be either true or false. For example, it makes sense to make human player's cards selectable and opponent cards not.
- selected defines the rank of the selected card. It will be displayed as an open card, and everyone will be able to see it.
- onCardSelectHandler allows to specify card select event handler. In game implementation used to let the game know that player did select a card.
- containerHeight is a parent element height in pixels. Hand component needs this to be able to calculate position and card sizes. In the game implementation, we used a FluidGameContainer component, which automatically passes its sizes to all first level child components and handles window resizing.
- containerWidth is a parent element height in pixels. Is used for the same purposes as a containerHeight parameter. FluidGameContainer is passing this attribute automatically as well.

Hand component could be used in the following way:

```
<Hand suit="S"
      open={true}
      position={3}
      compact={false}
      cards={this.state.playerCards}
      selectable={this.state.selectedPlayerCard === null}
      selected={this.state.selectedPlayerCard}
      onCardSelectHandler={(rank) => this.makeFirstPlayerMove(rank)}/>
```

Listing 6.5 `Hand` component usage example.

GoofSpiel game implementation consists of 3 hands. Each hand is positioned at different positions. We can see it in Figure 6.6. The human player can choose one of his open cards. And then, the game will take his choice and send data to the server to get opponent action and round score.

6.6 Pursuit-Evasion implementation

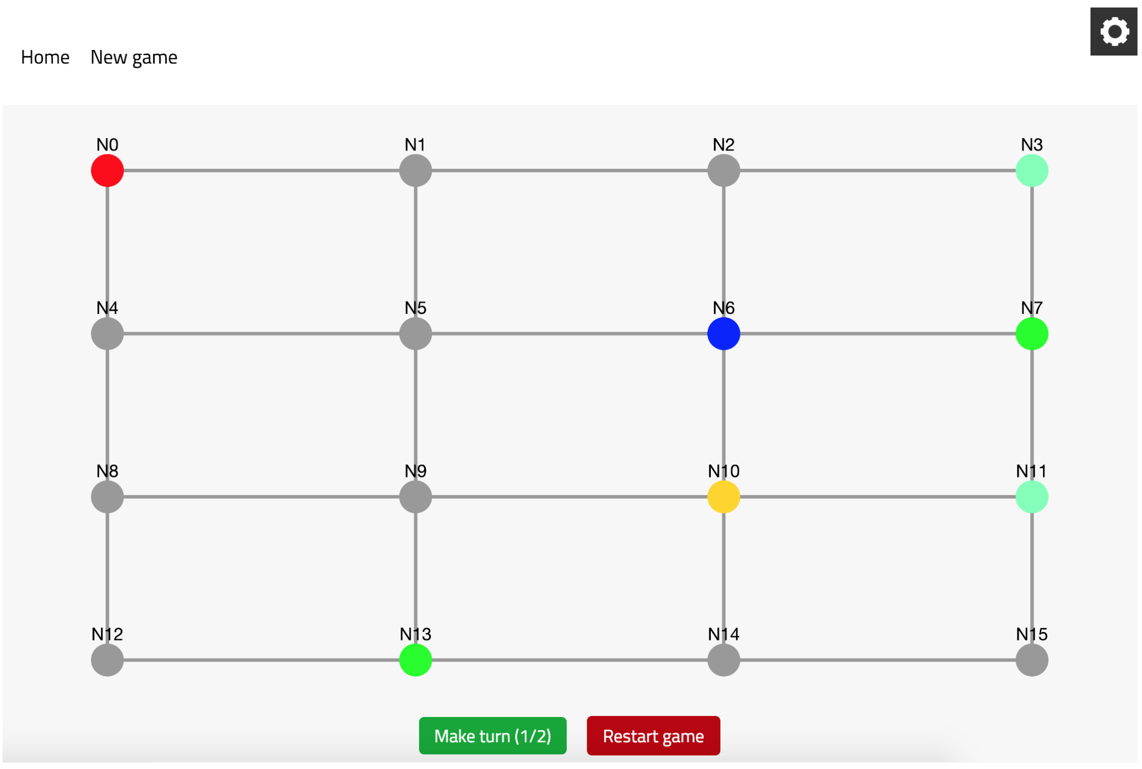


Figure 6.7. Pursuit-Evasion game visualization.

Pursuit-Evasion is a graph game. This game allows to human player to play two different roles - evader and pursuer. From the reason, that pursuer must do two moves we have to implement it in two steps. It means that when human plays pursuer, he needs to choose an action for the first pursuer, submit it, and then do the same for the second pursuer.

We can see the implementation result in Figure 6.7. Green nodes are current player positions. Light green nodes are possible actions. The player can choose one of them and submit his action by clicking on the green button at the bottom of the screen. Blue node is a currently selected node. The blue node will be taken as a player's action if the player will click on submit button. Red nodes are current opponent positions. And the yellow node is an evader goal.

The Pursuit-Evasion game implementation use cytoscape.js library for graph visualization and interaction. Cytoscape.js library has the support of different layouts. As it was described in Section 5.2, we can pass additional parameters with the game specification. One of such parameters we used in graph game implementation was the layout. For Pursuit-Evasion game we used a grid layout.

Game implementation takes care about game-specific logic. It provides an interface to make actions to the player and send the required data to the backend application. Also, implementation evaluates possible actions and make them selectable for a player.

Cytoscape.js is able to manage for us zoom event, positioning and drag events. So, we don't need to handle them manually. For the rest functionality, we created `CytoscapeGraph` component. It has the following parameters:

- `nodes` is a list of all graph nodes. Each node is a Cytoscape node object. In current implementation we are using the following structure for the node definition:

```
{
    group: 'nodes',
    data: {
        id: 'n1',
        label: 'N1',
    },
    selectable: false,
    classes: 'graph_game_goal graph_game_player'
}
```

Listing 6.6 Graph game node definition example.

- `edges` is a list of all graph edges. Each edge is a Cytoscape edge object and has the following structure:

```
{
    group: 'edges',
    data: {
        id: 'e10',
        source: 'n1',
        target: 'n5'
    },
    selectable: false
}
```

Listing 6.7 Graph game edge definition example.

- `layout` is a layout we want to use for the graph. `CytoscapeGraph` component will calculate grid dimensions automatically based on properties if the layout value is equal to a grid.
- `cyInitHandler` allows us to pass callback function which will be invoked after Cytoscape graph object will be initialized. The method parameter `cy` is a reference to Cytoscape graph object. We use it for graph manipulations. For example, in the callback, we are setting selectable nodes for the initial game state after we started a new game.
- `onNodeSelect` allows us to specify an event handler for node select event.

CytoscapeGraph component could be used as follows:

```
<CytoscapeGraph nodes={nodes}
                edges={edges}
                layout={layout}
                cyInitHandler={cy => this.onCyInitHandler(cy)}
                onNodeSelect={event => this.onNodeSelectHandler(event)}/>
```

Listing 6.8 CytoscapeGraph component usage example.

Chapter 7
Communication between client and server

In this section, we will cover communication between client and server applications. We will describe request and response formats to show what data they are sending to each other. These examples will give an overview of designed API.

As it was described in Section 3.2 client and server are communicating with each other using REST API provided by the server-side application. The message body is in JSON format.

The request usually has to use the following URL pattern:

```
POST http://{host}:{port}/{game_alias}/{algorithm_alias}
```

Listing 7.1 Game API endpoint URL pattern.

For example, for Oshi-Zumo game using Monte Carlo Tree Search Upper Confidence Bounds for Trees algorithm, request URL to the backend application running with default configuration on localhost will be `http://localhost:8100/OZ/MCTS_UCT`.

The request body composed from attributes which could be segregated to three groups. Attributes from the first group are related to the algorithm configuration. Attributes from the second group are related to the game configuration we chose during the new game start process. And the third group is describing the current game state, it could contain player action, moves history etc.

The response also has a similar structure for all games. It always has attributes `win`, `gameEnd` and `opponentActions`. The rest attributes in response a game specific and provide information about game state after evaluation of human player action and opponent (algorithm) action.

- `opponetActions` contains a list of selected by algorithm actions. For games when a player can make only one action during his turn it will contain a single item. But, for example, in Pursuit-Evasion game, when the human player has an evader role, it will contain two items, one for each pursuer.
- `gameEnd` determine the end of the game. Could be either true or false.
- `win` is a game result for the human player. Evaluation of this decision is done based on internal game rules. It could use utilities or some constraints to assign one of the following values: win, lose, draw. Make sense to check the value of this attribute when `gameEnd` is true.

7.1 Oshi-Zumo

7.1.1 Request

In Listing 7.2 is shown sample request from the client application.

```
{
    "seed": 10,
    "binaryUtilities": true,
    "generalSum": true,
    "moves": [
        {
            "playerMove": 3,
            "opponentMove": 4
        }
    ],
    "startingCoins": 7,
    "locK": 3,
    "minBid": 2,
    "playerMove": 2
}
```

Listing 7.2 Oshi-Zumo request example.

Attributes `seed`, `binaryUtilities` and `generalSum` are related to the algorithm configuration. Attributes `locK`, `minBid`, `startingCoins` are defining game configuration. And attributes `moves` and `playerMove` are describing the current game state.

Since requests for different games have a similar structure, we will describe only single game related attributes mapping to game domain terms.

- `locK` is a value of the middle field cell index. It means that whole game field will have `2 * locK + 1` cells. For example above, it will be equal to 7.
- `minBid` is a minimal amount of coins player can bid.
- `startingCoins` is a number of coins each player has at the begging of the game.
- In `moves` attribute we have a history of players actions from previous rounds.
- `playerMove` is a value of player bid in this round.

7.1.2 Response

In Listing 7.3, we have a response from the backend application.

```
{
    "opponentActions": [
        "2"
    ],
    "gameEnd": false,
    "win": "LOSE",
    "playerCoins": 2,
    "opponentCoins": 1,
    "wrestlerPosition": 2,
    "playerBids": [
        2
    ]
}
```

Listing 7.3 Oshi-Zumo response example.

As we can see, it contains common attributes described above and game specific attributes which define a new game state.

- `playerCoins` - the new amount of human player's coins after previous round evaluation.
- `opponentCoins` - the new amount of opponent's coins after previous round evaluation.
- `wrestlerPosition` - the new position of the wrestler after the previous round results evaluation.
- `playerBids` - a list of the bids human player can do in this round.

7.2 GoofSpiel

7.2.1 Request

In Listing 7.4, we can see request body for GoofSpiel game. It contains similar attributes, but there is one game specific attribute called `natureMove`.

This attribute reflects open card in a chance deck. At the very begging, when the game starts, the client making a request to the server with empty `playerMove` attribute to get the value of this card.

```
{
    "seed": 7,
    "binaryUtils": true,
    "moves": [
        {
            "playerMove": 6,
            "opponentMove": 10,
            "natureMove": 10
        }
    ],
    "playerMove": 12,
    "natureMove": 2
}
```

Listing 7.4 GoofSpiel request example.

7.2.2 Response

The response is shown in Listing 7.5.

```
{
    "opponentActions": [
        "2"
    ],
    "gameEnd": false,
    "win": "LOSE",
    "natureMove": 8,
    "score": "2 - 10",
    "playerHand": [
        1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13
    ],
    "opponentHand": [
        1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13
    ],
    "natureHand": [
        1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13
    ]
}
```

Listing 7.5 GoofSpiel response example.

Except for standard attributes, it has game specific attributes.

- `playerHand`, `opponentHand`, `natureHand` are cards in players decks after previous round evaluation.
- `score` is a current game score.
- `natureMove` is a new open card in the chance deck.

7.3 Pursuit-Evasion

7.3.1 Request

For the reason that in Pursuit-Evasion game human can play as an evader as well as a pursuer. Request body format is a bit different. In Listing 7.6, we can see an example of the request for a case when human is playing as an evader.

```
{
    "seed": 7,
    "firstPlayerPositions": [
        0
    ],
    "firstPlayerMoves": [
        1
    ],
    "secondPlayerPositions": [
        7, 13
    ],
    "secondPlayerMoves": [],
    "goal": [
        10
    ],
    "humanRole": 0,
    "graphFile": "pursuit_evasion/pursuit_simple4x4.txt"
}
```

Listing 7.6 Pursuit-Evasion evader request example.

- `graphFile` is a name of the file which contains graph definition from the selected configuration.
- `firstPlayerPositions` is a list of current positions of evader. Each item is a node identifier.
- `firstPlayerMoves` is a list of evader moves. Since evader is usually only one node, this list will contain only one item.
- `secondPlayerPositions` is a list of current positions of the pursuer.
- `secondPlayerMoves` is a list of pursuer moves.
- `goal` is a list of evader goals.
- `humanRole` determine of human plays as evader(0) or pursuer(1).

As we can see when a player is playing as an evader, we are sending only attribute `firstPlayerMoves`, and `secondPlayerMoves` is empty. In Listing 7.7, the situation is the opposite. It shows a sample request for a case when human is playing as a pursuer.

```
{
    "seed": 7,
    "firstPlayerPositions": [
        0
    ],
    "firstPlayerMoves": [],
    "secondPlayerPositions": [
        7, 13
    ],
    "secondPlayerMoves": [
        6, 9
    ],
    "goal": [
        10
    ],
    "humanRole": 1,
    "graphFile": "pursuit_evasion/pursuit_simple4x4.txt"
}
```

Listing 7.7 Pursuit-Evasion pursuer request example.

7.3.2 Response

For the reason that we don't need any additional information, a response for Pursuit-Evasion game consists only from standard fields.

```
{
    "opponentActions": [
        "3", "9"
    ],
    "gameEnd": false,
    "win": "LOSE"
}
```

Listing 7.8 Pursuit-Evasion response example.

Chapter 8
Evaluation

Tests are very important in any software. It helps to validate its consistency and verify functionality. It is much easier to maintain and extend an application with correct written tests. Tests could significantly decrease the number of bugs.

8.1 Automated testing

In this section, we will review and describe the tests and tools we used for automating testing. These tests are executed every time we run build. An application will not be built successfully until all these tests pass.

8.1.1 JUnit

JUnit is a well-known popular unit testing framework for Java code. It is a commonly used tool to write unit tests. It provides all necessary functions to clearly define preconditions and postconditions for tests and gives us tools to check different scenarios based on input parameters.

JUnit is often use as core framework for other testing frameworks.

8.1.2 Spock framework

Spock framework [39] is a testing and specification framework for Java and Groovy applications. It was inspired by different testing frameworks, such as JUnit. Spock allows us to write clear and descriptive specifications for an any scenario.

Spock has a powerful mocking system. It allows us to mock objects simulating their behavior or its parts. This is a commonly used practice that helps to isolate tested code unit from outer dependencies behavior.

```
private GameSpecificationProvider mockGameSpecificationProvider = Mock {
    getGame() >> Game.PE
    getSpecification() >>
        new GameSpecificationDTO(name: TEST_PE_GAME_SPECIFICATION_NAME)
    getConfigurations() >>
        new GameConfigurationsDTO([Mock(GameConfigurationDTO) {
            getName() >> TEST_PE_GAME_CONFIGURATION_NAME
        }])
}
```

Listing 8.1 Mocking an object with Spock framework.

Spock defines specifications in Groovy programming language. In Listing 8.1, we can see a mocking example. In combination with Groovy specific syntax, we are able to make tests simple and descriptive.

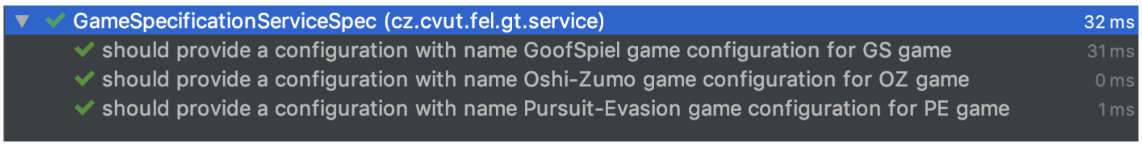
Using Spock, we can strongly separate a single test into logical parts. Spock provides keywords `given`, `then`, `when`, `expect`, `where` that could help to do it. We could even separate some of these sections into subsections using the keyword `and`. Each of these parts is executed in a predefined order. As the result, the test definition could be read as - given statement one and statement two, when executing code, then we expect the result, where parameters are equal to predefined values.

```
@Unroll
def "should provide a configuration with name #gameConfigurationName
    for #game game"() {

    given:
    GameSpecificationService gameSpecificationService =
       new GameSpecificationServiceImpl([
            mockPEGameSpecificationProvider,
            mockGSGameSpecificationProvider,
            mockOZGameSpecificationProvider
       ])

    when:
    GameConfigurationsDTO configurations = gameSpecificationService
        .getGameConfigurations(game)

    then:
    configurations.getConfigurations().size() == 1
    configurations.getConfigurations()
        .get(0).getName() == gameConfigurationName

    where:
    game    | gameConfigurationName
    Game.GS | TEST_GS_GAME_CONFIGURATION_NAME
    Game.OZ | TEST_OZ_GAME_CONFIGURATION_NAME
    Game.PE | TEST_PE_GAME_CONFIGURATION_NAME
}
```

Listing 8.2 Spock test definition example.

In Listing 8.2, we can see an example of such definition.

Another important part is an ability to simply execute parametrized tests and name them using dynamic string templates.



```
▼ ✓ GameSpecificationServiceSpec (cz.cvut.fel.gt.service)                                          32 ms
    ✓ should provide a configuration with name GoofSpiel game configuration for GS game            31 ms
    ✓ should provide a configuration with name Oshi-Zumo game configuration for OZ game             0 ms
    ✓ should provide a configuration with name Pursuit-Evasion game configuration for PE game       1 ms
```

Figure 8.1. Spock tests execution result.

The result of test execution from Listing 8.2 is shown in Figure 8.1. We can notice that Spock generated three tests with different names based on input parameters.

Spock has good integration with Spring framework.

8.1.3 Spring Boot test

Spring Boot test is one of the Spring projects. It combines different testing libraries and utilities that can be used for unit and integration testing.

To make an integration test, we need just add `@SpringBootTest` annotation at the class level. Spring will run a real application and start to execute test scenarios.

Spring Boot test provides various configuration options for integration testing. It is possible to have additional properties for an application executed for integration testing. We can define or override beans definitions and make application context manipulations. It is allowed to have different parameters for each integration test, in this case, Spring will group tests by these parameters and start a new application for each group with corresponding properties.

8.1.4 Jest

Jest [40] is a JavaScript testing framework. It provides a full set of functions to test a JavaScript code. Jest provides a mocking engine, assertation methods, let us test asynchronous code and many others.

Jest is provided by default as a testing tool for React application. It has good integration with React. We can simply use everything that we are using in real components code. Jest allows us to create React components, mock them, simulate events, and write a real use case scenarios to test our application.

Jest framework was created for testing not only React applications. It could be used for vanilla JavaScript applications as well. It has the support of ECMAScript 6 standard syntax and TypeScript.

The naming of methods in Jest is very descriptive. It helps a developer to write transparent tests.

Jest is developed and supported by Facebook. It has a big community, good documentation, and many examples. These facts mean that writing tests in Jest will not be complicated and we will be always available to find a solution for possible problems.

8.1.5 Checkstyle

There are many different testing techniques. Tests that are verifying code functionality is the only part of it.

Another important part is code quality. Good programming code should be readable, clean, well-organized and separated to the smaller reusable parts that respond for only one small piece of logic. Also, the code should fulfill commonly used syntax standards specific for each programming language. Naming is also very important. A well-written code should be readable like a book. Moreover, it would help to add new functions much faster.

Without team made code review is hard to achieve the desired result, but we can cover some basic things using tools like Checkstyle.

Checkstyle [41] is a development tool that helps to write a Java code that adheres to coding standards. This tool has set of predefined rules that are validated on build execution. It allows a programmer to define his own rules and flexible configure all rules for his needs.

For better understanding, we will give an example of some of the rules.

- Unused imports - check that there are no unused imports left. It is a common mistake. Sometimes developer could just miss them.
- Avoid nested blocks - check nested blocks complexity. We can specify how many nested blocks we could have in the scope of one function. Usually, it is zero or one but is possible to set up bigger complexity.
- Equal and hash code - check that classes that override `equals()` method are also override `hashCode()`.
- File length - check for long source files. We should have small classes that are responsible for only one thing.

- Inner assignment - checks for assignments in subexpressions.
- Class, variable, method names - check that related pieces of code are written in a coding standard case.
- Method count - check that the number of methods is not too big. If this check matches, it means that refactoring is required using a suitable design pattern.

This is a very small part of existing standard checks. And we can define our own. It will also help to keep a code written by different persons similar.

8.1.6 Summary

Client and server applications are covered with unit and integration tests using tools described in this section.

We know that having automated tests is very important. It helps us to verify a software quality and will be helpful in future extensions.

Checkstyle is also integrated into a build process. The build will not be successful if any Checkstyle rule is violated. This tool helps to keep code clean and consistent.

8.2 Manual verification

Manual verification is also an important part. It is a part of every deployment process to the production environment.

A user interface is part of the application as well. The end user will not use an application if an interface is not comfortable for him, even if everything works perfectly. During the development process we made various improvements to the user interface to make it intuitive and user-friendly.

Both applications were manually tested during development. And few times after development was complete, to verify that everything works as expected.

The web application was tested in most popular web browsers to catch browser specific problems.

An application was demonstrated to different peoples. Feedback was positive. Issues were not found.

Chapter 9
Installation

In this chapter, we will describe how to build and install applications.

9.1 Back end application

Most of the modern IDEs provide integration with Gradle model. For development purposes, we can import a Gradle project to IDE and run the main application class `cz.cvut.fel.gt.GTApplication`.

To install an application we need to build it first. To do that, we need to have a Java Development Kit version 8 installed, and then execute command `./gradlew build` from a command line interface in the application directory.

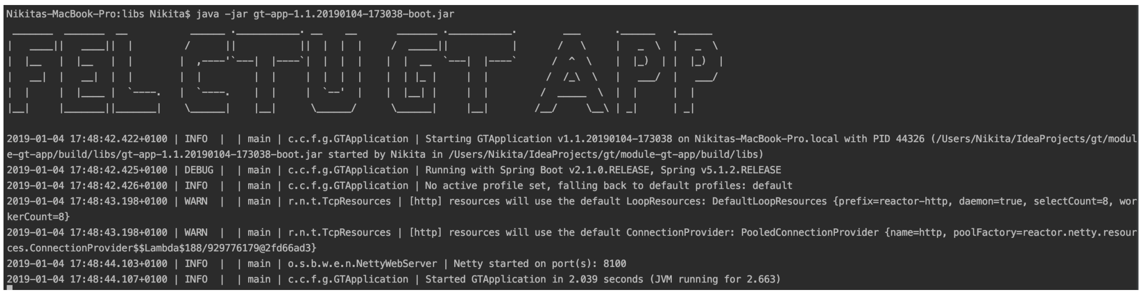
When we will run this command, Gradle will automatically download all required dependencies, execute tests and build jar files. As it was described in Section 5, we don't need to have Gradle instance installed in the operating system.

Note: Do not forget to specify CPLEX path as it was described in Section 5.1.

After command will successfully finish execution, source jar files will be generated for every application module. They are located in `<module_directory>/build/libs` directory, where `<module_directory>` is a single module directory name.

For module `module-gt-app` additional jar file will be generated. This is a fat jar. Both jars are located in `module-gt-app/build/libs` directory. The fat jar has classifier boot in its name. Now, we can deploy fat jar to any environment that has Java Runtime Environment.

The fat jar contains all required dependencies inside. To start an application, we need just execute fat jar as normal jar file using standard command `java -Djava.library.path=<path_to_CPLEX_binaries> -jar <jar_file_name>`.



```
Nikitas-MacBook-Pro:libs Nikita$ java -jar gt-app-1.1.20190104-173038-boot.jar
 _______  _______  __         ______ .___________. __    __      _______ .___________.      ___      .______   .______
|   ____||   ____||  |       /      ||           ||  |  |  |    /  _____||           |     /   \     |   _  \  |   _  \
|  |__   |  |__   |  |      |  ,----'`---|  |----`|  |  |  |   |  |  __  `---|  |----`    /  ^  \    |  |_)  | |  |_)  |
|   __|  |   __|  |  |      |  |         |  |     |  |  |  |   |  | |_ |     |  |        /  /_\  \   |   ___/  |   ___/
|  |     |  |____ |  `----. |  `----.    |  |     |  `--'  |   |  |__| |     |  |       /  _____  \  |  |      |  |
|__|     |_______||_______|  \______|    |__|      \______/     \______|     |__|      /__/     \__\ | _|      | _|

2019-01-04 17:48:42.422+0100 | INFO  |  | main | c.c.f.g.GTApplication | Starting GTApplication v1.1.20190104-173038 on Nikitas-MacBook-Pro.local with PID 44326 (/Users/Nikita/IdeaProjects/gt/modul
e-gt-app/build/libs/gt-app-1.1.20190104-173038-boot.jar started by Nikita in /Users/Nikita/IdeaProjects/gt/module-gt-app/build/libs)
2019-01-04 17:48:42.425+0100 | DEBUG |  | main | c.c.f.g.GTApplication | Running with Spring Boot v2.1.0.RELEASE, Spring v5.1.2.RELEASE
2019-01-04 17:48:42.426+0100 | INFO  |  | main | c.c.f.g.GTApplication | No active profile set, falling back to default profiles: default
2019-01-04 17:48:43.198+0100 | WARN  |  | main | r.n.t.TcpResources | [http] resources will use the default LoopResources: DefaultLoopResources {prefix=reactor-http, daemon=true, selectCount=8, wor
kerCount=8}
2019-01-04 17:48:43.198+0100 | WARN  |  | main | r.n.t.TcpResources | [http] resources will use the default ConnectionProvider: PooledConnectionProvider {name=http, poolFactory=reactor.netty.resour
ces.ConnectionProvider$$Lambda$188/929776179@2fd66ad3}
2019-01-04 17:48:44.103+0100 | INFO  |  | main | o.s.b.w.e.n.NettyWebServer | Netty started on port(s): 8100
2019-01-04 17:48:44.107+0100 | INFO  |  | main | c.c.f.g.GTApplication | Started GTApplication in 2.039 seconds (JVM running for 2.663)
```

Figure 9.1. Back end application start output.

When the application will be started, we will see the same output as in Figure 9.1.

9.2 Front end application

To build a client application, we need to have NodeJS installed.

The first thing we need to do is to run `npm install` command in the project directory. It will download all required dependencies to `node_modules` directory.

We can start an application in development mode using `npm start` command. It will start NodeJS server that will allow us to dynamically change and use an application. In development mode, sources are not compiled, also it provides various helpful information to a programmer.

To make a production build we need to run command `npm run build`. This command will build all dependencies, bundle it and make all defined transformations. As the result we will get a static assets located at `build` directory, that can be placed to any environment.

To make a production build, we need to run command `npm run build`. This command will build all dependencies, bundle it and apply all defined code transformations. As the result, we will get static assets located in the `build` directory. We could deploy compiled resources in any environment.

Chapter 10
Conclusion

The diploma thesis was created according to the assigned task under the Department of Computer Science of the Faculty of Electrical Engineering of the Czech Technical University in Prague within the study program Open Informatics, the specialization Software Engineering.

The main goal of this work is to create a web application that allows the human player to play in fundamentally different games against selected game-playing algorithms.

In this work, we became acquainted with game-theoretic library, compared various technologies needed for implementation and implement a web application consisted from the server-side application and client application that is running in a web browser.

Developed back end application is built on Spring framework and provides REST API that allows the client application to interact with game-theoretic library. We made few modifications into game-theoretic library to be able to play against game-playing algorithms simultaneously without any side effects. These changes were made with respect to backward compatibility. It means that they will not affect any use of game-theoretic library in other projects. Also, infrastructure for a comfortable game and algorithm loading was implemented.

For the front end part, we have to compare many different JavaScript libraries and frameworks. As core library, we chose a React.js. It was hard to find a library that will fit application aim for game-specific visualizations. There are a lot of JavaScript frameworks for big and complex games development with physics and nice graphics support. But they are not suitable for such game types as we need. Games for testing game theory algorithms are very specific. Often similar games are written manually as a project without providing any solution as a library. For this reason, we were implemented most of the required components by ourselves using libraries that allow us to interact with Canvas API in a comfortable way.

In this work, we analyzed and chose a technology stack that fits the current web application development and its future extensions.

Different components were created to visualize different types of games. These components could be reused in other games implementations or extended for game-specific goals.

We used different approaches to diverse games. Oshi-Zumo was implemented using only adapter library for a Canvas API. GoofSpiel was implemented without any third-party libraries using only HTML DOM elements for visualizations. Pursuit-Evasion was implemented using Cytoscape.js as graph visualization that is using a Canvas API. Using these different approaches, we are able to compare them. All of them are fits to application needs. We could use a Konva.js as a universal solution for all game types. The disadvantage of this approach is that we will need to write a lot of code from scratch, but we will get the solution that will do exactly what we want.

10.1 Future work

This work is the first part of a bigger application that should cover all capabilities of game-theoretic library. In this section, we will describe possible improvements of the web application.

- A JavaScript framework could be created. This framework could use Konva.js library and provide React components to operate with its blocks. The framework will provide building blocks for different games. Using it, we should be able to visualize any type of game. The framework should not depend on back end application and game-theoretic library. A game visualization should be the main goal of this framework. It should allow a developer to build a game using provided components without any modifications. It could be used by third-parties to implement their own custom games.
- Server-sent events could be used to provide additional decision improvements as described in Section 3.2.5.
- The game-theoretic library could be also improved by providing an ability to use an event-driven approach. For example, game-playing algorithm implementation could provide to bind asynchronous event listeners for different types of events. Such an event could be complete of one algorithm iteration. In this case, an algorithm will trigger an event with the current algorithm state. It could be used to incrementally maximize the evaluation result utility. It would also help to debug an algorithm.
- The web application could be extended for other game domains and game-playing algorithms.
- Algorithms that require keeping context could be added.
- NoSQL databases could be used for storing evaluation result for a specific game instance. For example, we can whole game tree and store the result in a NoSQL database. When we will need to get an algorithm player action, we will be able to retrieve it from the database. Neo4j or other implementations of graph NoSQL databases may fit the requirements the best.
- Some of the game-playing algorithms could be probably parallelized using CPU or GPU if present.

References

[1] *Game theoretic library.*
`http://jones.felk.cvut.cz/repo/gtlibrary/`. (Accessed on 12/20/2018).

[2] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research*. 2014, 51 829–866.

[3] Branislav Bošanskỳ, Viliam Lisỳ, Marc Lanctot, Jiří Čermák, and Mark HM Winands. Algorithms for computing strategies in two-player simultaneous move games. *Artificial Intelligence*. 2016, 237 1–40.

[4] Richard M Karp. *On-line algorithms versus off-line algorithms: How much is it worth to know the future?.* In: *IFIP Congress (1)*. 1992. 416–429.

[5] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. *Monte-Carlo Tree Search: A New Framework for Game AI.*. In: *AIIDE*. 2008.

[6] Levente Kocsis, and Csaba Szepesvári. *Bandit based monte-carlo planning.* In: *European conference on machine learning*. 2006. 282–293.

[7] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*. 2002, 47 (2-3), 235–256.

[8] Viliam Lisỳ, Marc Lanctot, and Michael Bowling. *Online monte carlo counterfactual regret minimization for search in imperfect information games.* In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. 2015. 27–36.

[9] James Gosling, Bill Joy, Guy L. Steele, Gilad Bracha, and Alex Buckley. *The Java Language Specification, Java SE 8 Edition*. 1st edition. Addison-Wesley Professional, 2014. ISBN 013390069X, 9780133900699.

[10] FELIPE GUTIERREZ. *PRO SPRING BOOT 2 an authoritative guide to building microservices, web and enterprise... applications, and best practices*. APRESS, 2018. ISBN 9781484236765.

[11] *Can I use... Support tables for HTML5, CSS3, etc.*
`https://caniuse.com/#search=es6`. (Accessed on 01/01/2019).

[12] *webpack.*
`https://webpack.js.org/`. (Accessed on 01/01/2019).

[13] Roy T Fielding, and Richard N Taylor. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Irvine, USA, 2000.

[14] Carlos Santana Roldan. *React Cookbook: Create dynamic web apps with React using Redux, Webpack, Node.js, and GraphQL*. Packt Publishing, 2018. ISBN 9781783980727.
`https://www.xarg.org/ref/a/1783980729/`.

[15] *Gradle — Gradle vs Maven Comparison.*
`https://gradle.org/maven-vs-gradle/`. (Accessed on 12/24/2018).

[16] *angular/angular: One framework. Mobile & desktop.*
https://github.com/angular/angular. (Accessed on 12/27/2018).

[17] *facebook/react: A declarative, efficient, and flexible JavaScript library for building user interfaces.*
https://github.com/facebook/react/. (Accessed on 12/27/2018).

[18] *vuejs/vue: Vue.js is a progressive, incrementally-adoptable JavaScript framework for building UI on the web.*
https://github.com/vuejs/vue. (Accessed on 12/27/2018).

[19] Adam Freeman. *Pro Angular 6*. Apress, 2018.
https://www.xarg.org/ref/a/B07FMLRBTD/.

[20] *React lifecycle methods diagram.*
http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/. (Accessed on 12/27/2018).

[21] *The State of JavaScript 2018: Front-end Frameworks - Overview.*
https://2018.stateofjs.com/front-end-frameworks/overview/. (Accessed on 12/27/2018).

[22] Einar Egilsson. *einaregilsson/cards.js: Javascript library for card games.*
https://github.com/einaregilsson/cards.js. (Accessed on 12/28/2018).

[23] *Phaser - A fast, fun and free open source HTML5 game framework.*
https://phaser.io/. (Accessed on 12/28/2018).

[24] Tikhon Jelvis. *Card Games Library — jelv.is.*
http://jelv.is/cards/. (Accessed on 12/28/2018).

[25] *melonJS.*
http://melonjs.org/. (Accessed on 12/28/2018).

[26] *pakastin/deck-of-cards: HTML5 Deck of Cards.*
https://github.com/pakastin/deck-of-cards. (Accessed on 12/28/2018).

[27] *Crafty - JavaScript Game Engine, HTML5 Game Engine.*
http://craftyjs.com/. (Accessed on 12/28/2018).

[28] *BabylonJS - 3D engine based on WebGL/Web Audio and JavaScript.*
https://www.babylonjs.com/. (Accessed on 12/28/2018).

[29] Ali Shakiba. *shakiba/stage.js: 2D HTML5 rendering engine for game development.*
https://github.com/shakiba/stage.js. (Accessed on 12/28/2018).

[30] *enchant.js - A simple JavaScript framework for creating games and apps.*
http://enchantjs.com/. (Accessed on 12/28/2018).

[31] *Overview - boardgame.io.*
https://boardgame.io. (Accessed on 12/28/2018).

[32] *Konva - JavaScript 2d canvas library.*
https://konvajs.github.io/. (Accessed on 12/28/2018).

[33] *konvajs/react-konva: React + Canvas = Love. JavaScript library for drawing complex canvas graphics using React.*
https://github.com/konvajs/react-konva. (Accessed on 12/28/2018).

[34] *Sigma js.*
http://sigmajs.org/. (Accessed on 12/28/2018).

[35] *D3.js - Data-Driven Documents.*
https://d3js.org/. (Accessed on 12/28/2018).

[36] *Cytoscape.js*.
http://js.cytoscape.org/. (Accessed on 12/28/2018).

[37] *uber/react-digraph: A library for creating directed graph editors*.
https://github.com/uber/react-digraph. (Accessed on 12/28/2018).

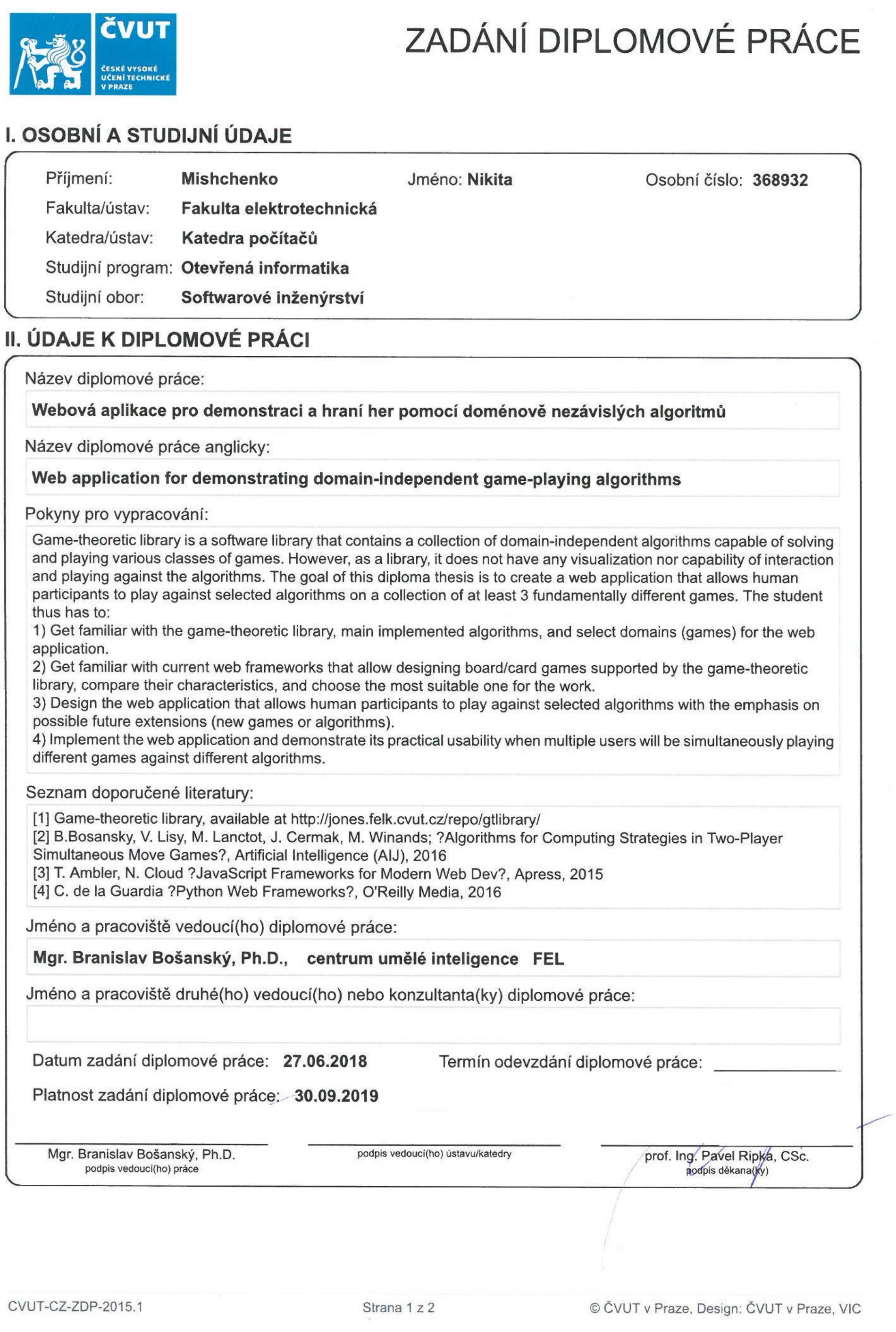
[38] *vis.js - A dynamic, browser based visualization library.*
http://visjs.org/. (Accessed on 12/28/2018).

[39] *Spock*.
http://spockframework.org/. (Accessed on 01/03/2019).

[40] *Jest - Delightful JavaScript Testing*.
https://jestjs.io/en/. (Accessed on 01/03/2019).

[41] *checkstyle – Checkstyle 8.16*.
http://checkstyle.sourceforge.net/. (Accessed on 01/03/2019).

Appendix A Specification



ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Mishchenko**	Jméno: **Nikita**	Osobní číslo: **368932**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Katedra/ústav:	**Katedra počítačů**		
Studijní program:	**Otevřená informatika**		
Studijní obor:	**Softwarové inženýrství**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Webová aplikace pro demonstraci a hraní her pomocí doménově nezávislých algoritmů

Název diplomové práce anglicky:

Web application for demonstrating domain-independent game-playing algorithms

Pokyny pro vypracování:

Game-theoretic library is a software library that contains a collection of domain-independent algorithms capable of solving and playing various classes of games. However, as a library, it does not have any visualization nor capability of interaction and playing against the algorithms. The goal of this diploma thesis is to create a web application that allows human participants to play against selected algorithms on a collection of at least 3 fundamentally different games. The student thus has to:
1) Get familiar with the game-theoretic library, main implemented algorithms, and select domains (games) for the web application.
2) Get familiar with current web frameworks that allow designing board/card games supported by the game-theoretic library, compare their characteristics, and choose the most suitable one for the work.
3) Design the web application that allows human participants to play against selected algorithms with the emphasis on possible future extensions (new games or algorithms).
4) Implement the web application and demonstrate its practical usability when multiple users will be simultaneously playing different games against different algorithms.

Seznam doporučené literatury:

[1] Game-theoretic library, available at http://jones.felk.cvut.cz/repo/gtlibrary/
[2] B.Bosansky, V. Lisy, M. Lanctot, J. Cermak, M. Winands; ?Algorithms for Computing Strategies in Two-Player Simultaneous Move Games?, Artificial Intelligence (AIJ), 2016
[3] T. Ambler, N. Cloud ?JavaScript Frameworks for Modern Web Dev?, Apress, 2015
[4] C. de la Guardia ?Python Web Frameworks?, O'Reilly Media, 2016

Jméno a pracoviště vedoucí(ho) diplomové práce:

Mgr. Branislav Bošanský, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **27.06.2018** Termín odevzdání diplomové práce: ______________

Platnost zadání diplomové práce: **30.09.2019**

Mgr. Branislav Bošanský, Ph.D.		[illegible] prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

CVUT-CZ-ZDP-2015.1 Strana 1 z 2 © ČVUT v Praze, Design: ČVUT v Praze, VIC

Appendix B
Symbols

Java EE	Java Enterprise Edition
Java SE	Java Standard Edition
CD	Compact disc
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
WS	WebSocket
IDE	Integrated development environment
URL	Uniform Resource Locator
API	Application programming interface
CRUD	Create Read Update Delete
JVM	Java Virtual Machine
DSL	Domain Specific Language
DTO	Data Transfer Object
JAR	Java Archive
MCTS	Monte Carlo Tree Search
DOM	Document Object Model
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
CSS	Cascading Style Sheets
ES	ECMAScript

Appendix C
Code listings

Appendix D
Content of attached CD

The content of a CD has the following structure:

- diploma_thesis – folder that contains diploma thesis source tex files and images.
- gt-app.zip - contains back end application source files.
- gt-app-web.zip - contains front end application source files.
- Web application for demonstrating domain-independent game-playing algorithms.pdf - diploma thesis in pdf format.