BACHELOR THESIS

Martin Červeň

Artificial Intelligence for Go on Non-standard Topologies

Department of Software and Computer Science Education

Supervisor of the bachelor thesis:	Mgr. Jakub Gemrot, Ph.D.
Study programme:	Computer Science
Study branch:	IOI

Prague 2018

Title: Artificial Intelligence for Go on Non-standard Topologies

Author: Martin Červeň

Department of Software and Computer Science Education: Department of Software and Computer Science Education

Supervisor: Mgr. Jakub Gemrot, Ph.D., Department of Software and Computer Science Education

Abstract: Go is a popular strategic game for two players. It is usually played on a squared board of 19x19. The aim of this thesis is to create an application allowing a user to play Go on any board defined by a graph, such as sphere and torus. We created a web based client-server application, written in JavaScript and Node.js, that is using protocol Websockets for fast communication. Application allows user to play against other players on the Internet. Server can support multiple concurrent games at the same time. Boards can be created by modelling tool Blender and then imported into the application. Our application supports 3D rendering of these boards in web client using WebGL. It has responsive control, allowing changes of view by rotating, moving and zooming. Users can also play against artificial intelligence.

Keywords: game tree search, Monte Carlo Tree Search, board game, Go, non-standard topology

I would like to thank to my supervisor Jakub Gemrot for answering my questions and for his patience. I would also like to thank to my family for continuous support throughout my study.

Contents

1. Introduction

Go is an ancient strategic board game for two players. It is played approximately by 40 millions players, mainly in Asia. It is interesting to the field of artificial intelligence because it has simple rules, but is difficult to master. In May 2017 AI called AlphaGo developed by Google beat No.1 ranking player in the world Ke Jie [1]. Go is played on square wooden board using black and white stones. Rules of the game are further described in section **2.1**.

While there are applications allowing a user to play Go on squared board, there is no accessible way of playing Go on other than square board. In our thesis we are going to develop an application allowing users to design their own Go boards, visualise them, allow them to play Go on these boards against other players and develop basic artificial intelligence that will be able to play against user.

We developed a web based application that satisfy these needs. It consists of client and server, and has 3D visualisation of Go boards. It is written in a modular way, which could be used for development and playing of other games than Go, for example, we implemented game Connect Four. You can play our application at `http://3dgo.io:3000` against other human players or AI.

1.1 Goal of the thesis

The goal of our thesis is the development of an application that would allow user to play Go on boards defined by undirected graph. Playing on these boards requires creating them, and visualising them. It would also be useful to be able to play against other players across the Internet. We also develop Artificial Intelligence that players can play against. To summarise these needs, we develop an application that would satisfy following goals.

Goal 1.) allow users to play Go on boards defined by undirected graph,

Goal 2.) support multiplayer playing of Go against other users,

Goal 3.) allow users to play against artificial intelligences,

Goal 4.) visualise Go defined by undirected graphs,

Goal 5.) let users design their own Go boards, and play on them in this application.

1.2 Structure of the thesis

In the second chapter, **Background**, we describe the game of Go, it's rules and history. We also define Go on non standard topologies and give some examples of non-standard boards.

In the third chapter, **Related work**, we describe several Go applications and their features. We also mention Go AI history.

In the fourth chapter, **Analysis**, we describe decision process that we used to select architecture of our application, which frameworks and technologies we used and which algorithm we chose for AI.

User documentation will be presented in the fifth chapter. We will describe how a user connects to the website, how to start a server, how can user create a game and how to play a game using our application.

In the sixth chapter, **Development documentation**, we present how we developed our application, what classes we created and we will show state diagrams of client and server and give an example of communication between client and server.

In the last, seventh chapter, **Conclusion**, we summarise what we have achieved and we will outline the future work that could be done and was not part of our goals.

2. Background

In this chapter we will introduce Go game and its rules. We then describe Go on non standard topologies, which we will call **3DGo**

2.1 Rules of Go

In this section we will introduce rules of Go. We chose to use rules as found on Wikipedia [2] , which are essentially rules of the American Go Association [3] in somewhat digestible form. We added examples of rules in figures for additional clarity.

Game of Go is a board game played by two players, called **Black** and **White**. Go is played on a squared **board**, composed of 19 vertical and 19 horizontal lines.

Point where horizontal line meets vertical line is called an **intersection**. Two intersections are adjacent, if they are connected by horizontal or vertical line with no other intersections between them. Go is played with tokens known as **stones**.

At any time, each intersection on the board is in one of the following states: empty, occupied by black stone, occupied by white stone. A **position** is an arrangement of the states of all the intersections on the board. Two placed stones of the same color (or two empty intersections) are **connected** if between them exists path of the same state.
Liberty of a stone is an empty intersection adjacent to that stone or adjacent to a stone which is connected to that stone. **Position** is a Example of stones with various number of liberties is in figure **2.1**.

Play: Board is empty at the beginning of the game. The players alternate in moving, with Black playing first. Move is either a **pass**, or a **play**. Play consists of following steps:

1. Placing a stone of their color on an empty intersection.
2. Removing from board any stones their opponents color that have no liberties, this is called capture. Capture of one stone is shown in figures **2.3** and **2.3** and capture of multiple stones is shown in figures **2.4** and **2.5**.
3. Removing from board any stones of their own color that have no liberties, this is called self-capture.

Prohibition of suicide: Play is **illegal** if one or more stones of that players color would be removed in step 3 of that play. Suicide is shown in figure **2.6**.

Positional superko rule: A play is **illegal** if it would have the effect of creating a position that has occurred previously in the game (pass is allowed).This rule is shown in figure **2.7**.

End: The game ends when both players have passed consecutively. The final position is the position on the board at the time the players pass consecutively. Example of final position is shown in **2.8**

Territory: In the final position, an empty intersection is said to belong to a player's territory if all stones adjacent to it or to an empty intersection connected to it are of that player's color.

Area: In the final position, an intersection is said to belong to a player's area if either: 1) it belongs to that player's territory; or 2) it is occupied by a stone of that player's color.

Score: A player's score is the number of intersections in their area in the final position.

Winner: If one player has a higher score than the other, then that player wins. Otherwise, the game is drawn.

Optionally, a non integral value called **komi** is added to the white's score to break the tie. Because Black player moves first he has an advantage. Komi, which was nonexistent before 20th century, was progressively increased over time to offset this advantage and is now usually 6.5 or 7.5.

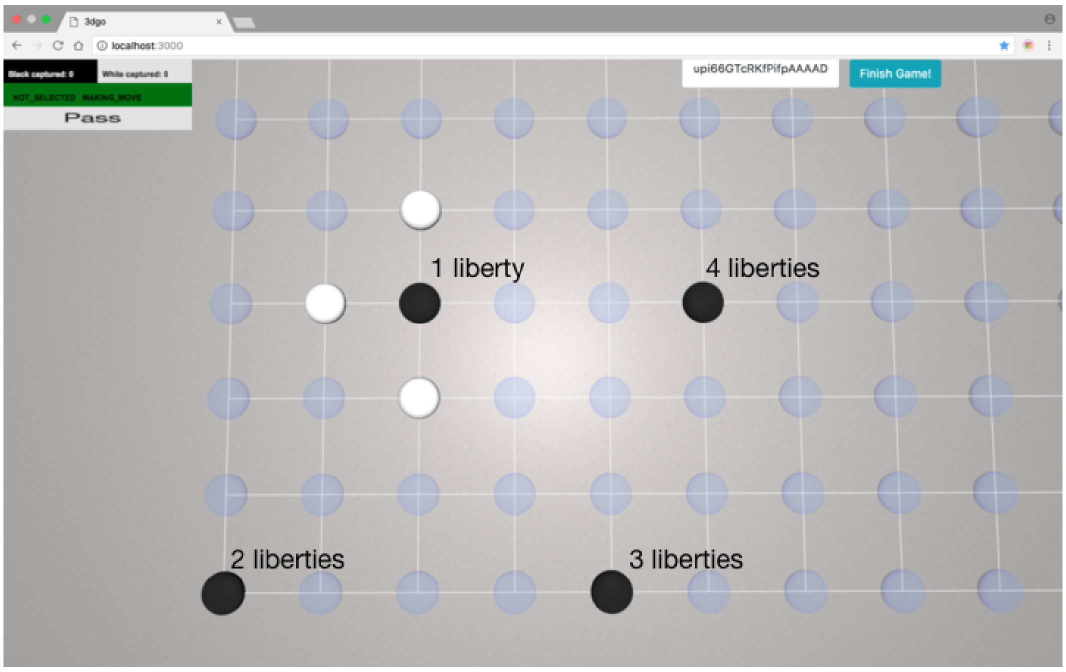


Figure 2.1: Stones with various liberties

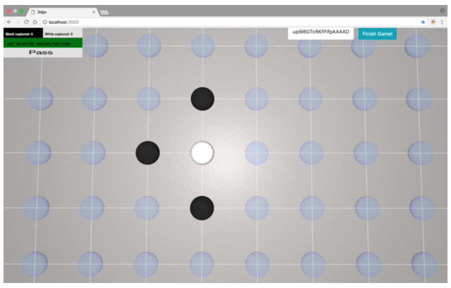


Figure 2.2: White stone has one liberty

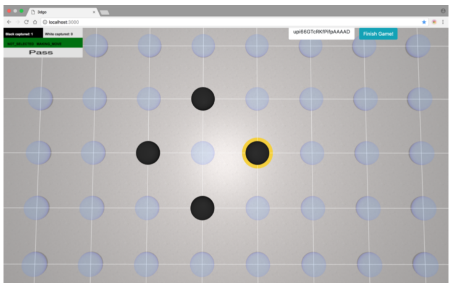


Figure 2.3: Black captured white stone

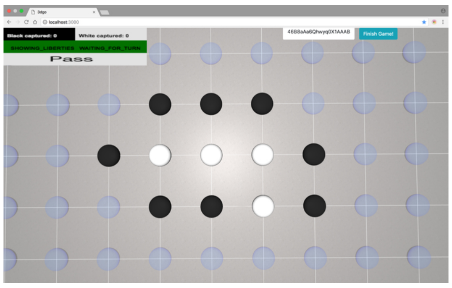


Figure 2.4: White stones has one liberty

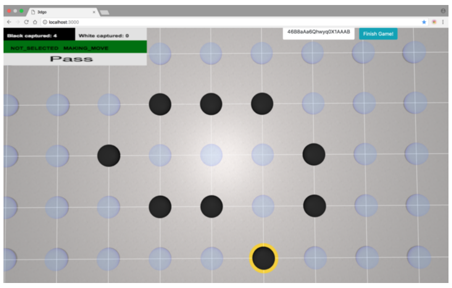


Figure 2.5: Black captured white stones

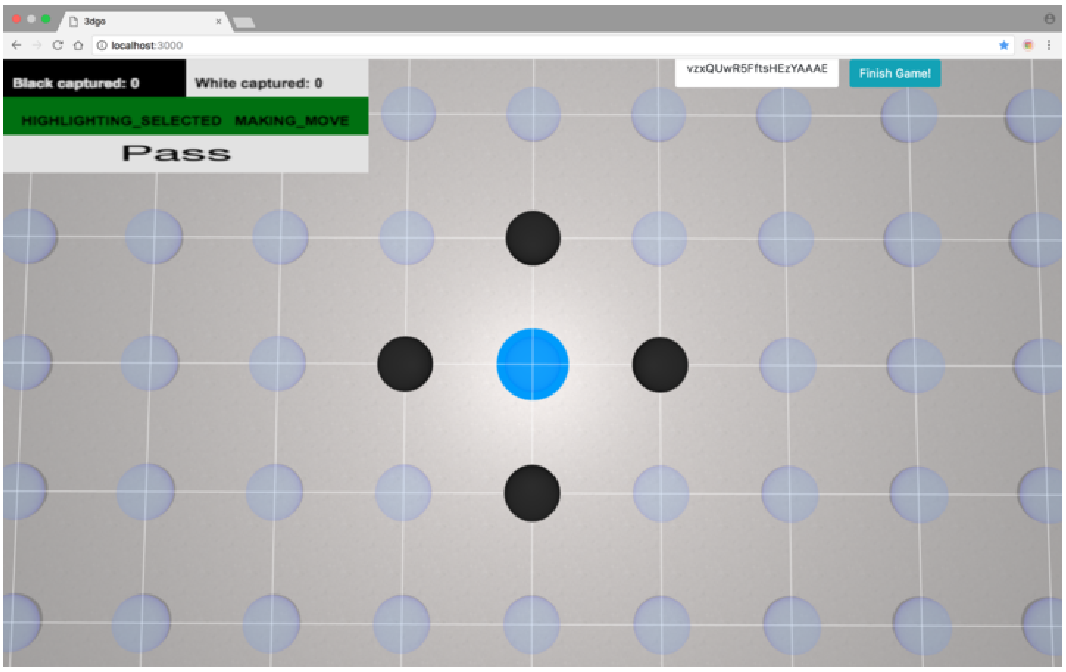
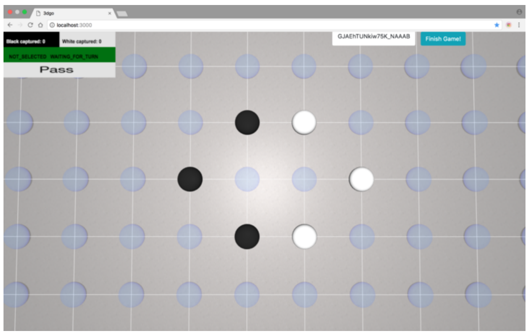
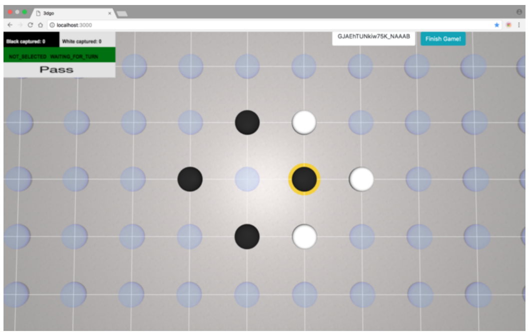
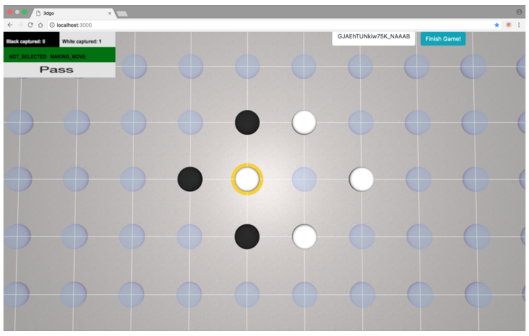


Figure 2.6: Example of suicide, White cannot play at highlighted position.


(a) Initial position


(b) Black plays


(c) White captures

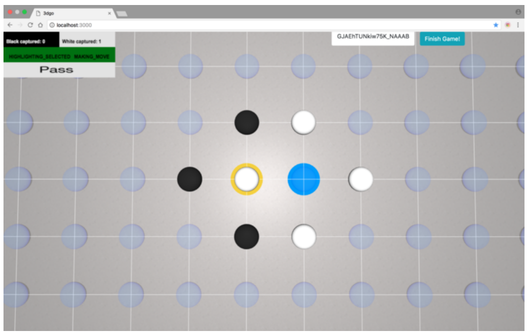

(d) Black can not play the same move

Figure 2.7: Example of positional superko rule.

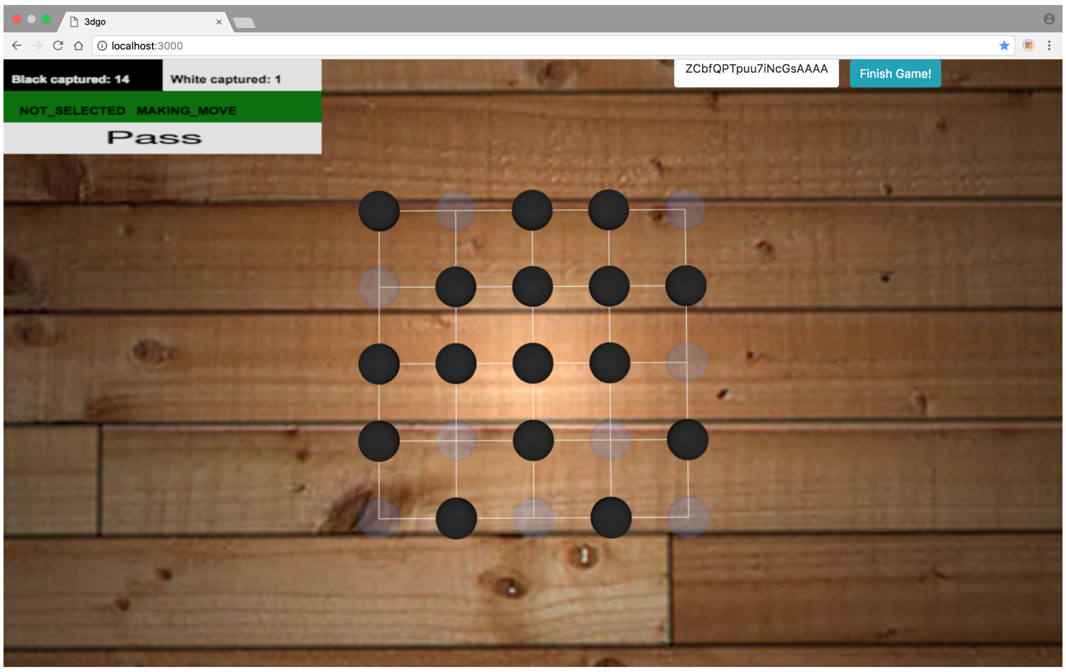


Figure 2.8: Example of a final position

2.2 Go on non standard topologies

We extend Go by not playing only on a squared board, but also on boards defined by an arbitrary undirected graph, as stated in goal **Goal 2.)**. The game mechanics and rules stays the same as in traditional Go. But since in traditional Go, each stone has four, three and two liberties depending on position, in 3DGo it can have less or more. This will lead to different strategies than on a squared board.

There were a few tendencies to extend Go for other boards than squared or change dimensions from ordinary. Few programs exists that allow this. However, most of them are only available for one platform, does not support current operating systems, or are only single-player. One such example is shown in section **3.1.3.**

Our application allows creation of board with arbitrary "topology". In other words, each vertex can have arbitrary neighbours, and boards can be arbitrary large, and vertices together can be arbitrarily connected.

This allows creation of interesting boards such as torus, as show in figure **2.12**, which has no edges or corners like square board, mobius strip, which is like band of rubber but with inverted edges, this creates interesting play as it disallows capture along the edge.

The other interesting examples are circle, as shown in figure **2.11** or sphere with cut off top and bottom, in figure. **2.10**. These are examples of seemingly different boards, but are topologically the same.

Our thesis also creates new material for development of artificial intelligence. To date, advances in Go AI are oriented for 19x19 squared boards. It would be interesting to observe play of artificial intelligences on other boards than squared.

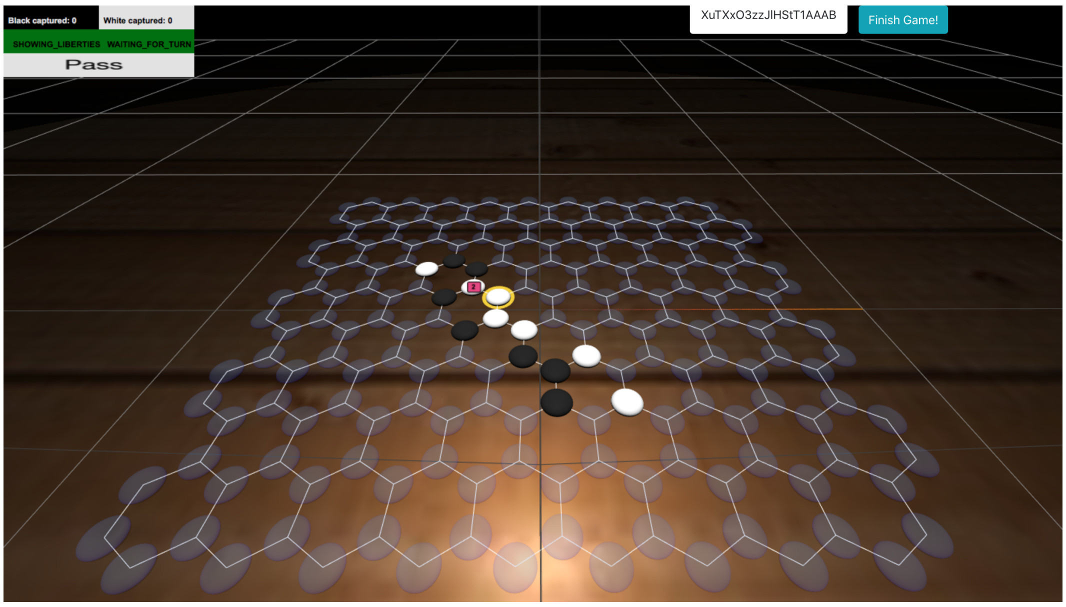


Figure 2.9: Hexagonal grid, screenshot from a game

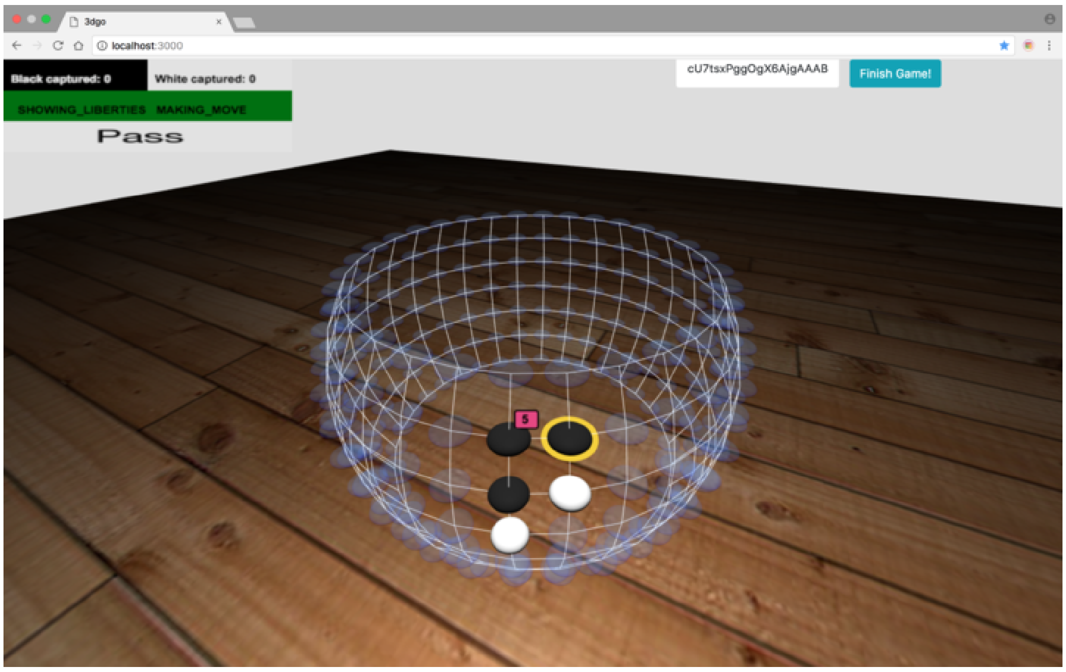


Figure 2.10: Spehre with cut off top and bottom

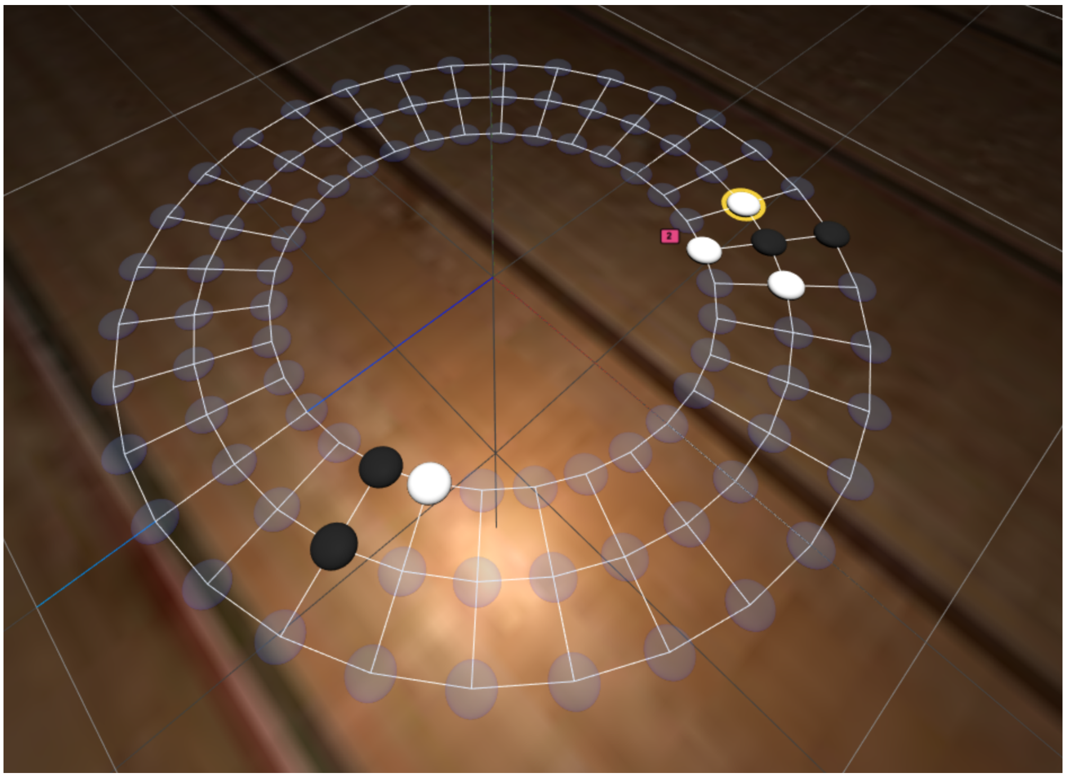


Figure 2.11: Circle board, screenshot from a game

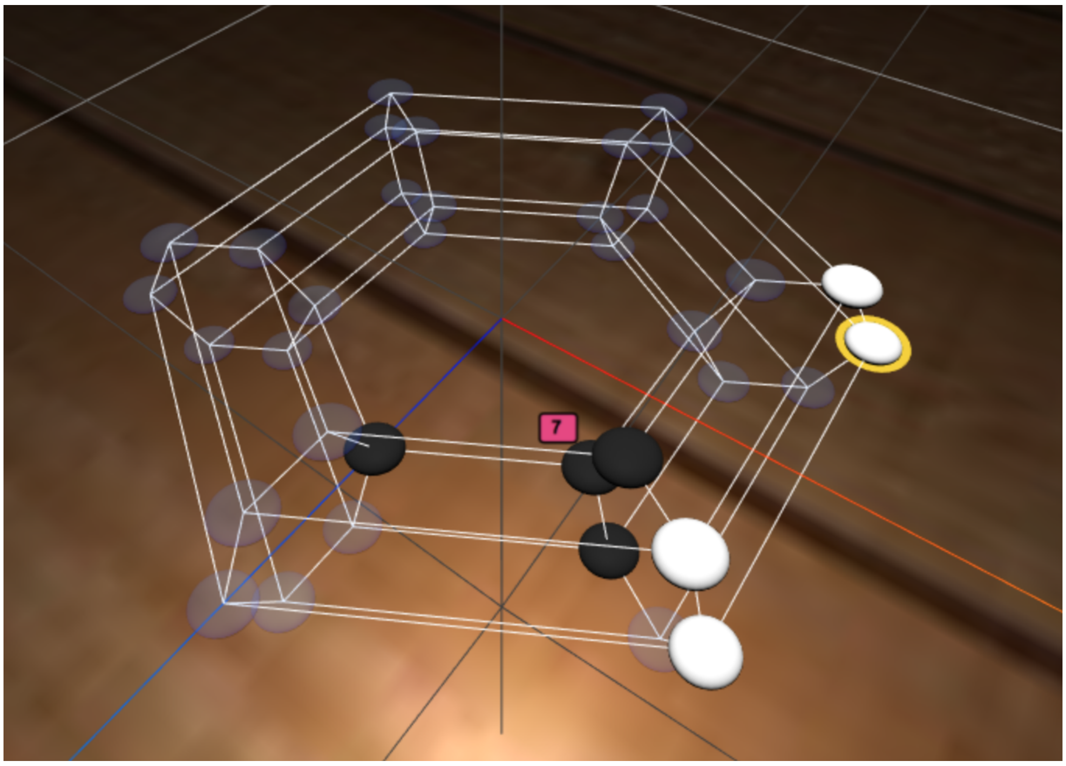


Figure 2.12: Torus board, screenshot from a game

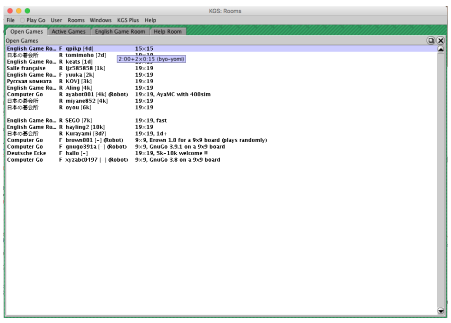
3. Related work

In this chapter we present related work in two sections. First section describes several applications that allow user to play Go. Second section describes history of Go AI.

3.1 Applications for playing Go

3.1.1 KGS

KGS [4] is a client server application written for playing Go. It was created by William M. Shubert in 2000, and is written in Java. It has a lobby, chat rooms, and support play of squared Go boards of sizes 2x2 to 38x38. It also supports viewing files in Smart Game Format (SGF) [5] and connecting AIs in Go Text Protocol (GTP) [6]. AlphaGo was trained on database of games from this server. In 2017 it was sold to American Go Foundation, and several new clients not based on Java are being developed.



```
English Game Ro... F  qpikp [4d]                15×15
日本の碁会所       R  tomimoho [2d]             19×19
English Game Ro... R  keats [1d]                19×19
Salle française    R  ljz585858 [1k]            19×19
English Game Ro... F  yuuka [2k]                19×19
Русская комната    R  KOVJ [3k]                 19×19
English Game Ro... R  Aling [4k]                19×19
Computer Go        R  ayabot001 [4k] (Robot)    19×19, AyaMC with 400sim
日本の碁会所       R  miyane852 [4k]            19×19
日本の碁会所       R  oyou [6k]                 19×19

English Game Ro... R  SEGO [7k]                 19×19, fast
English Game Ro... R  hayling2 [10k]            19×19
日本の碁会所       R  Kurayami [3d?]            19×19, 1d+
Computer Go        F  brown001 [-] (Robot)      9×9, Brown 1.0 for a 9x9 board (plays randomly)
Computer Go        F  gnugo391a [-] (Robot)     9×9, GnuGo 3.9.1 on a 9x9 board
Deutsche Ecke      F  hallo [-]                 19×19, 5k-10k welcome !!
Computer Go        F  xyzabc0497 [-] (Robot)    9×9, GnuGo 3.8 on a 9x9 board
```

2:00+2×0:15 (byo-yomi)

Figure 3.1: Screenshot of CGoban, the KGS client.

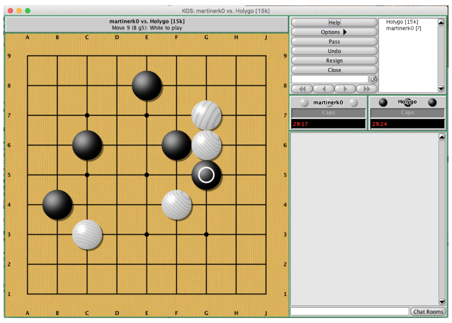

KGS: martinerk0 vs. Holygo [15k]
martinerk0 vs. Holygo [15k]
Move 9 (B g5): White to play
Help
Options
Pass
Undo
Resign
Close
martinerk0
Holygo
Chat Rooms

Figure 3.2: Screenshot of a game played on KGS.

3.1.2 Online-go.com

Online-go server [7] is a popular Go server that is available from the browser. It has appealing user interface, hosts regular tournaments and users can play against AI. It also has ranking system, database of the played games and real time chat.

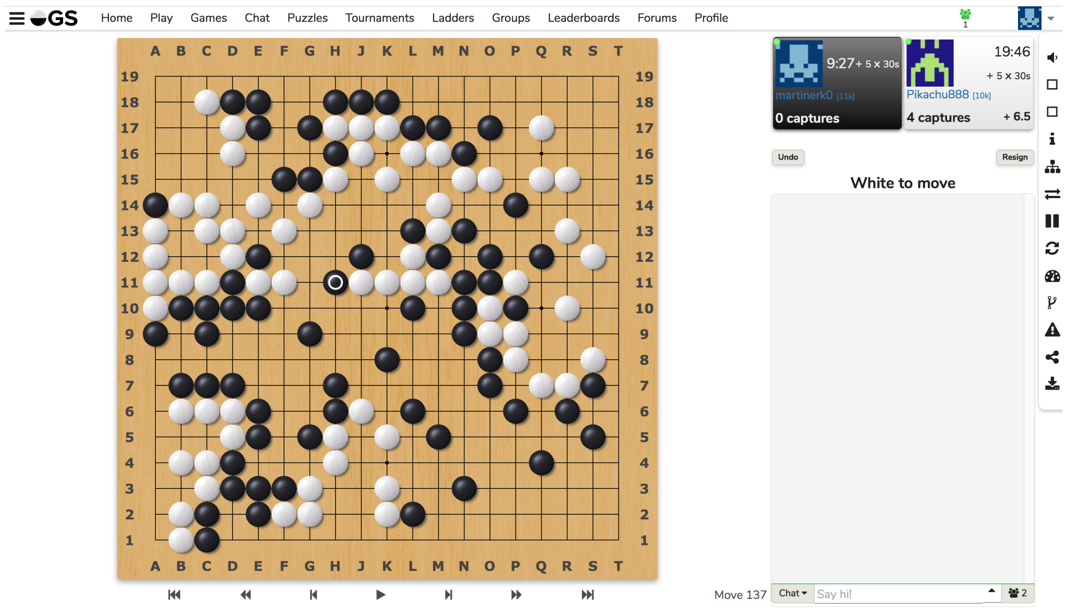


Figure 3.3: Online-go web client, screenshot from a game.

3.1.3 Freed Go

Freed Go is a Go application by Lewey Geselowitz [8]. Users can play Go on boards like Sphere, Torus, Mobius Strip and else. It renders these boards in 3D and supports user to user network play. It has also several drawbacks, such as, it does not have central server, it is only for windows platform, does not allow play against AI, is not open source, and its controls support only rotation of view. Figures 3.4 and 3.5 shows boards in Freed Go.

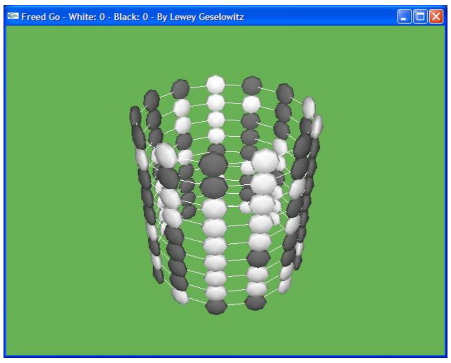


Figure 3.4: Screenshot of cylindric board in Freed Go.

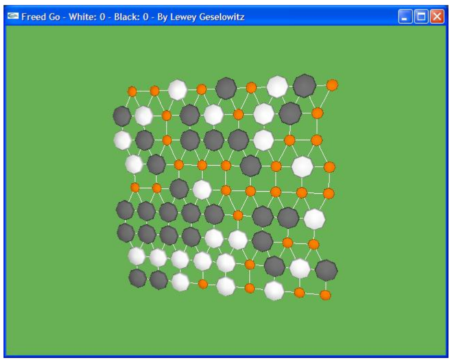


Figure 3.5: Screenshot of a irregular board in Freed Go.

3.2 Computer Go and AI

Chess was for a long time main topic of study of Artificial intelligence community [9]. Focus of AI researchers shifted to computer Go after Deep Blue defeated reigning chess champion Garry Kasparov in 1997.
Computer programs playing chess at that time were using Alpha beta pruning and were run on large supercomputers. Deep blue searched on average 126 million positions per second in match with Kasparov [10].

It is not easy to design a Go playing program that plays well, because it is not easy to design evaluation function of moves [11]. Go has up to 361 legal moves and more than 10^{170} states. This is too much for alpha beta searches used succesfully in chess. [9].
In 2006, Coloum proposed new algorithm using monte carlo simulations combined with tree search, called **Monte Carlo tree search**[12]. Then, in 2007 UCT was added, which also imporved performance of Go playing programs. Neural networks was used in 2000s but they have not met performance of MCTS algorithms

In October 2015 AlphaGo beat european Go champion Fan Hui on 19x19 board.
AlphaGo was trained on millions of games from professional games. [13]

In 2016 Alpha Go beat one of strongest players in history, Lee Sedol [14]. Next version of Alpha Go beat Next stage in computer Go came next year. Google Deepmind created new program AlphaGo Zero that did not learn from human games, but played against itself over many iterations. At each iteration, next generation of program would be stronger than previous.
In 2017, AlphaGo Zero,that was trained without human knowledge, beat the previous version of AlphaGo trained on human expert moves. [13]

4. Analysis

In this chapter we analyse possible application architectures we could use. We also consider how we are going to render boards of our application and what are our options. We also consider what programming language could be used to develop our application, what libraries and frameworks we could use, and what tools would be beneficial to the development of our application. We also consider possibilities of creating custom boards, and what are advantages and disadvantages of various approaches. Lastly, we describe several algorithms for possible AI, and chose appropriate one for our problem.

For each choice we list several options, their advantages and disadvantages, and pick one we deemed the best. We will guide our choices by the goals we set up earlier in the introduction. At the end of the chapter we will repeat choices that we made along the way.

Let us reiterate the goals:

Goal 1.) allow users to play Go on boards defined by undirected graph,

Goal 2.) allow users to play Go on these boards against other users,

Goal 3.) allow users to play against artificial intelligences,

Goal 4.) visualise Go defined by undirected graphs,

Goal 5.) let users design their own Go boards, and play on them in this application.

4.1 Application architecture

Our application can be programmed by different approaches. In this section we describe two types of application architectures, we consider their advantages and disadvantages and describe which one is better suited for our problem.

4.1.1 Monolithic application

Monolithic application [15] is an application performing all the tasks itself. It has these advantages:

- it is easy to design and program,
- it is not dependent on the Internet connection.

But it also has disadvantages, such as:

- user has to download and install it,
- if application is updated, he has to download update,
- user cannot play with other players on the Internet,
- AI will run on the same machine, taking computing time.

4.1.2 Client server architecture

Client server architecture is a distributed application structure that allows splitting tasks between multiple machines. [16].

These are its advantages:

- user can play against other players on the Internet,
- AIs can connect to the server as clients, so we can use the already-programmed system, and agents can run as a separate process or on another machine,
- according to [17], 54,4% of all people on earth is connected to the Internet.

It also has few disadvantages:

- it is harder to design and program than monolithic application,
- user has to download client,
- if client is updated, he has to download update,
- connection to the Internet is required.

Goal 1.) could be implemented in both architectures, but because we want to support multiplayer play as stated in **Goal 2.)**, thus we choose **client server architecture**.

4.2 Type of client server architecture

We chose client server architecture in the previous section as the go to application architecture. Now we need to specify how clients would look like. We have several options.

4.2.1 Platform-dependent client server

Our first option is to build client for every architecture that we target. It is easier with languages such as Java and C#, but still, the user has to have installed Java virtual machine (JVM) [18] or Common Language Runtime (CLR)[19] to run the application.

These are its advantages:

- user can play with other players on the Internet,
- AI can connect to the server as client.

Its two disadvantages are:

- For every platform have specific application, in case of Java or C# have installed JVM or CLR respectively. For example Apple's iOS devices does not support Java, which is large portion of user base.
- In case of updates, rebuild and download updated application.

4.2.2 Multi-platform game framework/engine

Another choice that is often used is using some kind of game framework/engine that builds application for many platforms. Great example of this is Unity game engine [20] ,shown in Figures **4.1** and **4.2**.

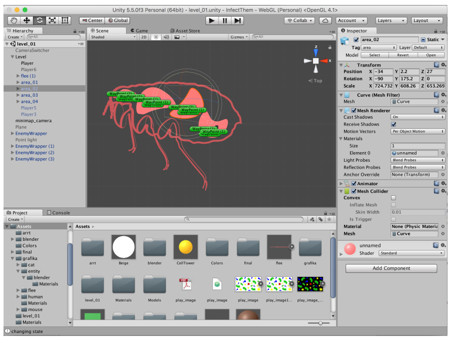


Figure 4.1: Screenshot of Unity editor

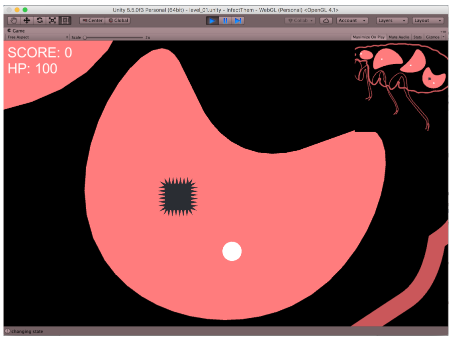


Figure 4.2: Game running in Unity editor

It has same advantages as desktop client server, plus it can build to many different platforms. While it may seem as perfect solution, it also has several disadvantages:

- necessity of using engine's/framework's editor,
- necessity to build application for each platforms,
- necessity of using some predetermined language, such as C# in Unity,
- it is usually not free,
- necessity to study particular framework in depth.
- Other limitations, for example, free version of Unity limits player number for multiplayer to 20, and building for WebGL is not supported on mobile devices [21].

4.2.3 Web client server

This design will use web browser as a client. Web browser is an application allowing user to connect to the Internet and browse web pages. First browser, shown in figure **4.3**, was written in 1990 by Tim Berners-Lee at CERN [22] and since then browser's capability dramatically increased. Rich web applications [23] rival capabilities of their desktop counterparts. Figure **4.4** shows an example, Google Docs [24]. Almost every personal computer today has some sort of browser capability.

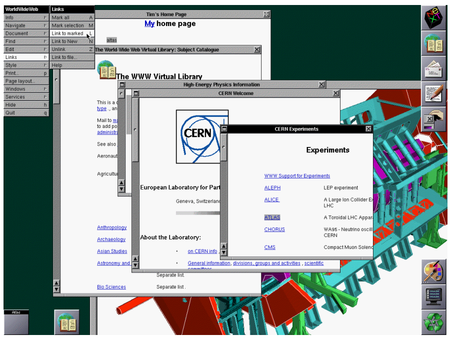
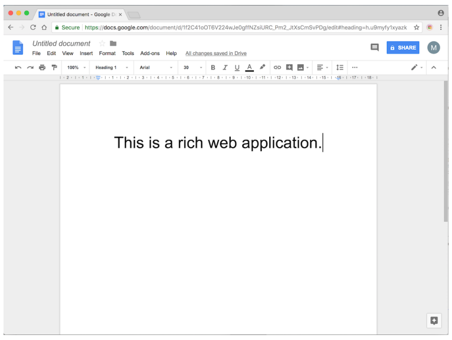


Figure 4.3: The first web browser by Tim Berners-Lee



This is a rich web application.

Figure 4.4: Google Docs web application

Advantages of this architecture for our application are:

- user does not have to install application,
- large user base, 55.4% [17],
- user can play with other players on the Internet,
- AI can connect to the server as client,
- user does not have to update application.

It also has disadvantages:

- it is hard to design and program,
- user has to download client every time.

Web applications have large user base and are inherently multiplatform. Users do not have to install application, they only connect to the website. Web applications offer easy updating and are not artificially limited in any way as game engines. We therefore decided to choose **Web client server** architecture for developing our application. Our choice satisfies our first three goals:

Goal 1.) because this could be implemented in any architecture,

Goal 2.) becuase we chose client server architecture,

Goal 3.) because AI can connect to server as clients, and play against human clients or other AI via server.

4.3 3D rendering in web browsers

Because we want to play Go on boards defined by non directed graph, our graphs can be badly represented in 2D. We could project other shapes into 2D plane, for example torus, but the result would look very messy and chaotic. Because of this, we will render boards in 3D. There could be boards that could look chaotic even in 3D too, but for sake of simplicity of gameplay we are not going to consider that. Next we consider technologies which we could use for 3D rendering, and select one which we will use.

4.3.1 Choosing the right technology

While **Adobe Flash** [25] has been dominant force in interactive web gaming industry for many years [26], it started to lost ground with Apple's decision to not support it in iOS in 2010 [27]. Adobe said that it discontinues use of Flash in favour of open standards like HTML5 and WebGL [28]. Figures 4.5 and 4.6 show games built with Adobe Flash.

Figure 4.5: Civilization Wars [29], a real time strategy game built with Adobe Flash

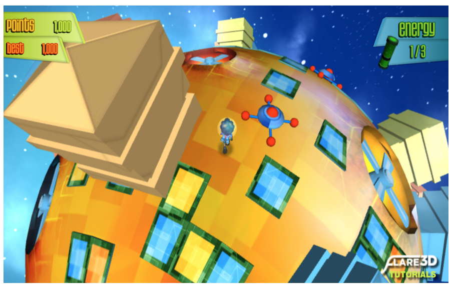


Figure 4.6: Yellow Planet [30], a 3D running game built with Adobe Flash.

Microsoft Silverlight [31] is a similar application framework as Adobe Flash for writing rich web applications. It was deprecated in favour of HTML 5 in 2015 [32].

WebGL [33] is a **JavaScript** API for hardware-accelerated 2D and 3D graphics rendering within compatibile web browsers without use of plug-ins like Flash and Silverlight. It can be used with HTML5 <canvas> elements. Its API is modelled after OpenGL ES 2.0. It is cross-platform and widely implemented in major browsers[33]. Figure 4.7 shows Quake [34], 3D game ported to WebGL and figure 4.8 shows Experience Curiosity, an 3D interactive application developed by Nasa [35].

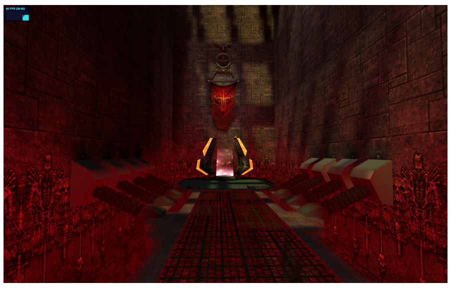


Figure 4.7: Quake game ported to WebGL

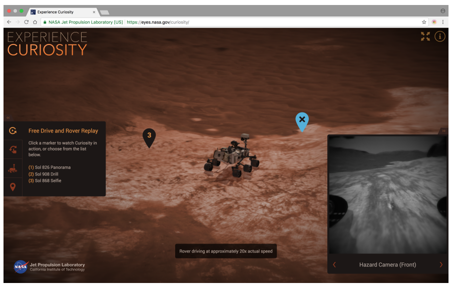


Figure 4.8: Experience Curiosity interactive application

WebGL offers many advantages, such as support in all major browsers, is fast, and allows rendering of 3D graphics. It is standard 3D graphics API for the Web [36]. **We will use WebGL** for 3D rendering of our application.

4.3.2 Using graphics library

Now that we established that what technology we will use for rendering, we need to consider what other functionality we need for board rendering and comfortable gameplay of our application. We need to be able to:

- render stones,
- render edges between stones,
- manipulate view of board, e.g. zooming, panning, rotating,
- select stones we wish to play by clicking or touching.

But since WebGL is just a low-level drawing API that works with arrays of data and shaders, it is very tedious to work with [36]. We need several more things to be able to render boards and play our application such as:

- have common math operations with vectors and matrices,
- import objects and create basic shapes,
- have a scene that will contain all objects that we wish to draw,
- have multiple views on the scene,
- have a way to manipulate objects,
- have a way to manipulate cameras, such as rotation, zoom and panning.

All this functionality that we listed, has been already implemented in graphics libraries, which are WebGl under the hood. We will pick one, **Three.js** [37], which has most stars on github, and is has very good documentation, code examples and friendly community. Figures 4.9 and 4.10 show examples of various projects build with Three.js.

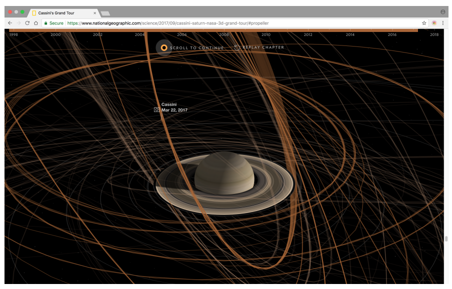


Figure 4.9: Cassini's Grand Tour by National Geographic [38].

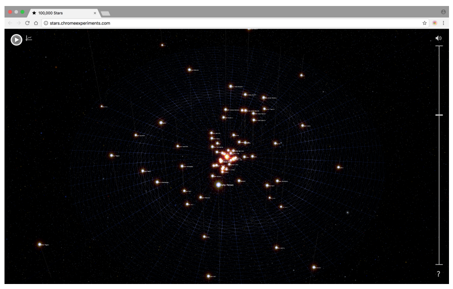


Figure 4.10: Interactive visualisation of Milky Way by Google [39].

Three.js has these advantages:

- it hides the details of 3D rendering and has common objects like scenes, cameras, meshes, materials, lights,
- it is fast,

- it has basic geometric shapes and supports import of common 3D file formats,
- it has common math operations,
- it supports interaction with objects such as clicking and selecting,
- it supports interaction with camera, such as rotation, zoom and panning,
- it can render to 2D Canvas, SVG, and CSS in case some browser does not support WebGL,
- it is open source.

There are also game engines built on top of Three.js, offering additional functionality, but since Three.js fulfils all our needs for rendering boards and playing Go, **we will directly use Three.js**. This choice satisfies **Goal 4.)**, because we have all the tools necessary to visualise Go boards in 3D.

4.4 Programming language of client

Because we decided that we will use WebGL, we will program logic of our application in JavaScript.

4.5 Programming language of server

We already decided that we will use JavaScript for client side programming. Now we need to decide what language we will use to program server.

Since we are going to use JavaScript for client side programming, we decided to program server in JavaScript too. This will save us time because we don't have to think in two languages, and we can reuse parts of code between client and server.

JavaScript can be run on server using JavaScript runtime environment, such as very popular **Node.js** [40]. Javascript has been most popular language for 5 years in a row according to [41], and Node.js most popular framework in 2017 [41].

4.6 Choice of programming tools

In this section we identify what we need to effectively program our application and explore various options for programming tools that could help us with development, and select most suitable ones.

We would like to have tools that would have features such as:

- ability to write code efficiently,
- syntax highlighting of code,
- code linting,

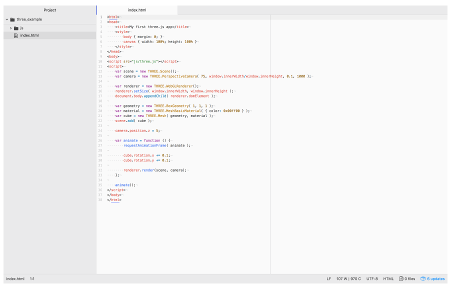
- debugging of JavaScript on both client and server side,
- code refactoring,
- custom hotkeys and code snippets,
- source control such as Git,
- ease of use.

4.6.1 Basic text editor

Basic text editor usually is able only to read/write text to file, but some programmers use it although it does not offer any help or functionality that we listed above.

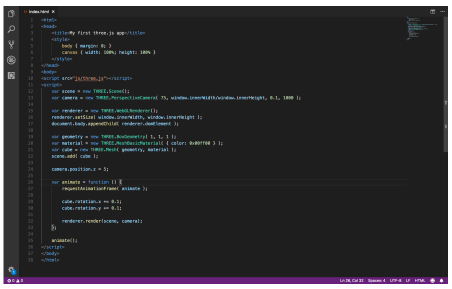
4.6.2 Advanced text editors

Text and source code editors such as **Atom** [42] and **Visual studio code** [43] are open source programs that besides text editing abilities offer additional functionality through use of plug-ins. They support syntax highlighting, linting, code refactoring, custom hotkeys and even can be used to debug Node.js applications.



```
<html>
<head>
    <title>My first three.js app</title>
    <style>
        body { margin: 0; }
        canvas { width: 100%; height: 100% }
    </style>
</head>
<body>
<script src="js/three.js"></script>
<script>
    var scene = new THREE.Scene();
    var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1, 1000 );

    var renderer = new THREE.WebGLRenderer();
    renderer.setSize( window.innerWidth, window.innerHeight );
    document.body.appendChild( renderer.domElement );

    var geometry = new THREE.BoxGeometry( 1, 1, 1 );
    var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
    var cube = new THREE.Mesh( geometry, material );
    scene.add( cube );

    camera.position.z = 5;

    var animate = function () {
        requestAnimationFrame( animate );

        cube.rotation.x += 0.1;
        cube.rotation.y += 0.1;

        renderer.render(scene, camera);
    };

    animate();
</script>
</body>
</html>
```

Figure 4.11: Atom editor



```
<html>
<head>
    <title>My first three.js app</title>
    <style>
        body { margin: 0; }
        canvas { width: 100%; height: 100% }
    </style>
</head>
<body>
<script src="js/three.js"></script>
<script>
    var scene = new THREE.Scene();
    var camera = new THREE.PerspectiveCamera( 75, window.innerWidth/window.innerHeight, 0.1, 1000 );

    var renderer = new THREE.WebGLRenderer();
    renderer.setSize( window.innerWidth, window.innerHeight );
    document.body.appendChild( renderer.domElement );

    var geometry = new THREE.BoxGeometry( 1, 1, 1 );
    var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
    var cube = new THREE.Mesh( geometry, material );
    scene.add( cube );

    camera.position.z = 5;

    var animate = function () {
        requestAnimationFrame( animate );

        cube.rotation.x += 0.1;
        cube.rotation.y += 0.1;

        renderer.render(scene, camera);
    };

    animate();
</script>
</body>
</html>
```

Figure 4.12: Visual studio code editor

4.6.3 Debugging in browser

For client side debugging, all major browsers has some sort of debugging capability. In figure 4.13 and 4.14 we show two examples of debugging in Google Chrome and Apple Safari respectively.

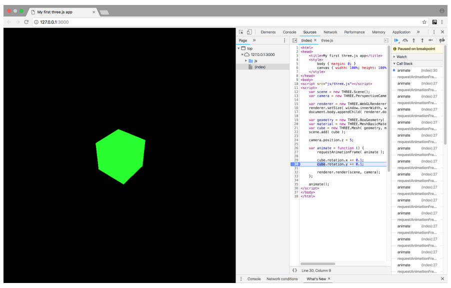


Figure 4.13: Chrome DevTools [44] in Google Chrome

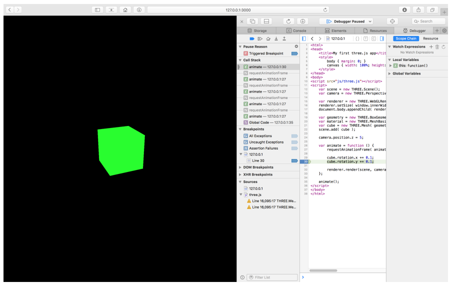


Figure 4.14: Web Inspector [45] in Apple Safari.

4.6.4 WebStorm

WebStorm [46] is a Integrated development environment (IDE) by JetBrains and is free for students. It offers syntax highlighting, code linting, code refactoring, Node.js debugger, integrated source control, refactoring tools and it can also use debug plug-in within the browser so we can also debug client side code with it. It has also same user interface as another popular IDEs we are familiar with, such as IntelliJ IDEA and PyCharm, made by the same company. WebStorm offers all the tools we need, has familiar UI, and for us, programming in it is faster than other two editors we tried, Atom and Visual Studio Code.

We therefore decided to use **Webstorm** for development of our application.

4.7 Choice of other frameworks and libraries

In this section, we describe three additional libraries that we will need to create responsive, good looking application.

4.7.1 Bootstrap

We chose web client architecture in the section 4.2. We will be rendering Go games in 3D, but we need a way to connect players together. That could be done for example by creating HTML table of available games, sent to client by server. Since plain HTML is not particularly eye catching in the 21st century, we will use front-end library **Bootstrap** [47] to get more polished look for our application. We chose it because it is most popular library on GitHub and we have some experience with using it.

4.7.2 Express.js

Another library that we will use, is **Express.js** [48]. It is an web framework for Node.js. While Node.js itself is a JavaScript runtime environment with networking capability, Express.js makes it easier to set up and run server with the Node.js.

4.7.3 Socket.io

Because we will need to communicate from server to client, and from client to server, we will use **Websockets** [49], which is a TCP-based protocol. It has several advantages, such as full duplex communication, it has lower overhead than HTTP and it is providing a stateful connection. There are several libraries for easy Websocket client to Node.js programming, we will use **Socket.io** [50], which has most GitHub stars.

4.8 Creation of user defined boards

In this section we describe several possible ways, how could a user create custom board and play it in our application.

4.8.1 Create custom board editor

We could create custom board editor in the client. This has several advantages such as:

- it is user friendly,
- user directly sees the board he\she is creating.

But it has also several disadvantages, for example:

- it is hard to implement,
- it has limited capability to create complex shapes.

4.8.2 Use existing software

Using existing software to create boards has many advantages:

- we do not have to program additional part,
- we could leverage existing capabilities of modelling software such as procedural generation on shape generation,
- different softwares could be used.

It has one disadvantage:

- user has to export board from modelling tool and import it into our application.

By using existing modelling software, we do not have to program custom editor. Existing modelling softwares have vast capacity to create complex shapes, the only disadvantage to this is that we have to implement importer to our application. We therefore choose to **use existing modelling software** to create boards, and **create importer** to import them into our application. This choice satisfies **Goal 5.)**, and we have thus covered how are we going to satisfy all goals.

4.9 Choice of AI algorithm

In this section we describe few AIs and approaches used in computer Go, and then we select most suitable approach.

4.9.1 Use existing Go AI

There exists open source Go AIs such GNUGo by Free Software Foundation [51] or Pachi made by Petr Baudiš [52]. These AIs are used only for squared boards, which prevents us from using them. They are also developed by large teams of people and their source code is using lot of heuristics used specifically for squared 19x19 boards such as databases of moves and patterns. Furthermore, they are using GTP [6] for communication. Since we are using Websockets protocol for communication, it would be additional thing to implement.

There are also Go AIs such as AlphaGo [14] and Crazy Stone [53] which are proprietary and we cannot use them.

4.9.2 Use existing AI technology

We therefore need to build our own 3DGo AI, which will be inspired by algorithms that are used by existing Go AIs. Our two candidates are **Monte Carlo tree search** [9] and **neural networks** [54].

Neural networks that are used by AIs such as AlphaGo are trained on large databased of moved or by self play. Either way, they are specified for squared boards and require extremely large amounts of computing power, 1700 years of computation on common hardware [55].

Furthermore, we would require separate training for each board, making this approach impractical. MCTS does not require training and is not confined on squared board only. We decided to use MCTS algorithm for our AI.

4.9.3 Monte Carlo tree search

MCTS is an algorithm that iteratively builds search tree that assigns value to the moves. Root node is current state a game, and its children are next moves from current state. Children node with highest value is move which algorithm thinks its best move in that situation. Algorithm is anytime, that means if stop it prematurely, it gives us current best result. If we give it more time, it should give us better result.
MCTS algorithm consist of four steps:

1. Selection: Start from root R and select successive best child nodes down to a leaf node L.
2. Expansion: Add a child node C to the L according to available actions.
3. Simulation: From node C do a random playout until terminal state and compute outcome.
4. Backpropagation / Backup: Go from C to R and update nodes with information from playout along the way.

These steps are shown in figure **4.15**

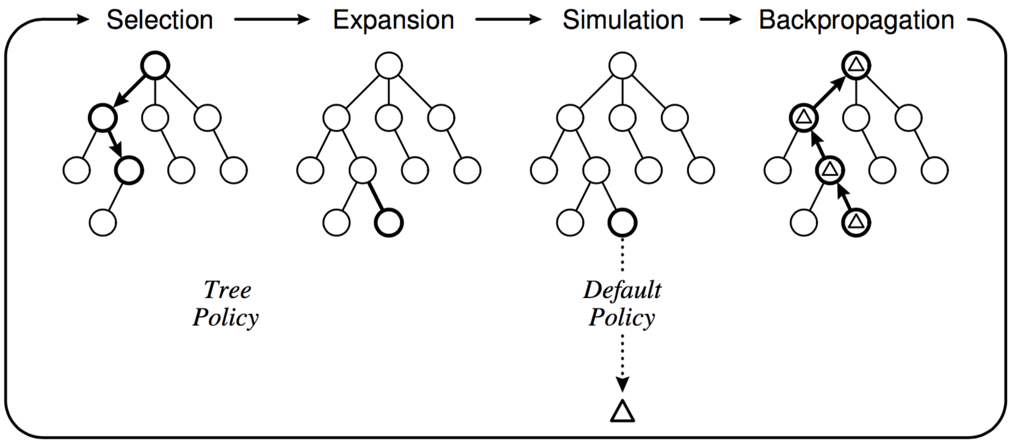


Figure 4.15: Steps of MCTS algorithm, figure from [56].

Upper Confidence Bounds for Trees (UCT)

Since MCTS pick up best children in selection, it is called greedy MCTS. There can be nodes with immediate bad values, but in the long run could be best. Thus we need a way to explore even nodes with immediate bad values. We will give each node a bonus based on uncertainty we have about that node, and thus we will explore even nodes that are not immediately of high value. Thus we want to maximize:

$$n_{val} + c\sqrt{\frac{\ln(p_{visits})}{n_{visits}}}$$

where n_{val} is value of a node n, c is a tunable constant, we will use $\frac{1}{\sqrt{2}}$ according to [56], p_{visits} is number of visits of parent node of n and n_{visits} is number of visits of node n.

5. User documentation

This chapter describes how to use our application. It outlines elements of web interface, parts of GUI, and shows tutorial on how to play the game of 3DGo. It shows how to use the web interface to join an already created game, how to create a new game and how to play against artificial intelligence. Moreover, we show how to create an arbitrary board using free, open source modelling tool **Blender** [57] and how to import it into our application to play it.

5.1 User documentation of client

5.1.1 Requirements for client

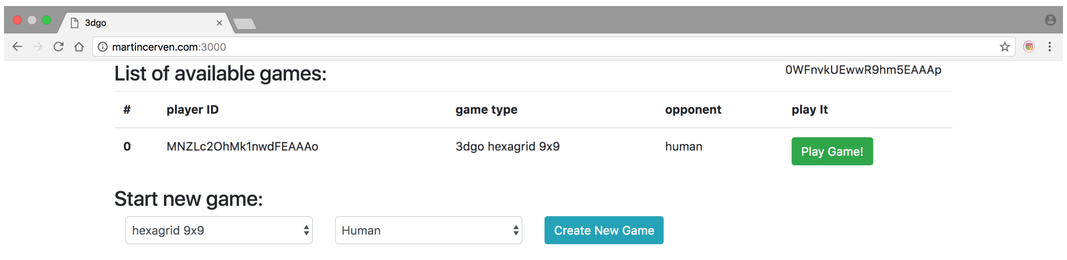
User who wants to play our application needs browser Google Chrome version 63.0.3239.108 or higher, or Apple Safari version 11.0.1. or higher.
Mozilla Firefox version 57.0.1 (64 bits) is also supported, but user needs to go to **about:config** page and set **dom.moduleScripts.enable** to true.

5.1.2 How can user connect to the application

User can connect to the website clicking on following link:

http://3dgo.io:3000

He then selects already created game from a list of games by clicking on *Play game!* button, as shown in figure 5.1



List of available games:

0WFnvkUEwwR9hm5EAAAp

#	player ID	game type	opponent	play It
0	MNZLc2OhMk1nwdFEAAAo	3dgo hexagrid 9x9	human	Play Game!

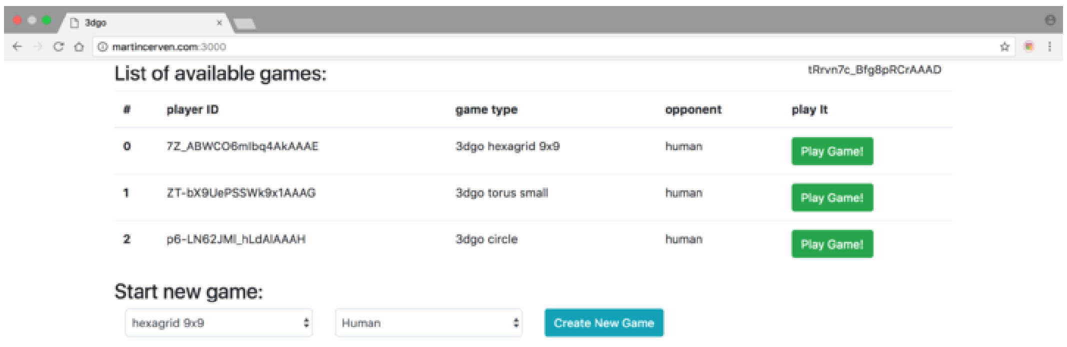
Start new game:

hexagrid 9x9 | Human | Create New Game

Figure 5.1: Selected game starts when user clicks on "Play game" button

5.1.3 Lobby

When user goes to website http://3dgo.io:3000 he is sent to a lobby page. It's purpose is to let users create a new game or to play other games, available in the table in the center of the screen. Table lists available games, sorted by date of their creation.



List of available games:

tRrvn7c_Bfg8pRCrAAAD

#	player ID	game type	opponent	play It
0	7Z_ABWCO6mlbq4AkAAAE	3dgo hexagrid 9x9	human	Play Game!
1	ZT-bX9UePSSWk9x1AAAG	3dgo torus small	human	Play Game!
2	p6-LN62JMI_hLdAlAAAH	3dgo circle	human	Play Game!

Start new game:

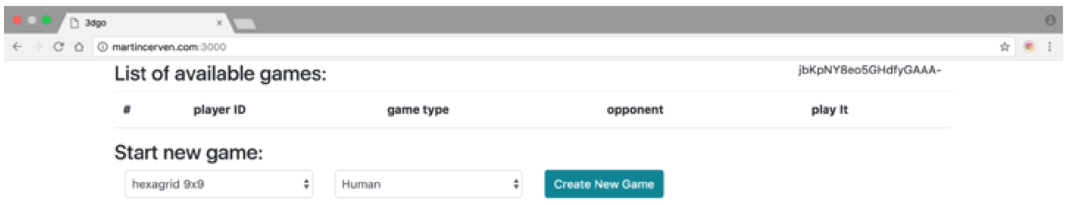
hexagrid 9x9 | Human | Create New Game

Figure 5.2: Lobby, screenshot

5.1.4 How can user create new game

User can create new game by selecting which board he wants to play from drop-down list. He also selects which user he wants to play against, human, random agent or MCTS agent.

He then clicks on play game button.



List of available games:

jbKpNY8eo5GHdfyGAAA-

#	player ID	game type	opponent	play It

Start new game:

hexagrid 9x9 | Human | Create New Game

Figure 5.3: How user create game, screenshot

5.1.5 How user joins already created game

User can see all information about games in lobby. If he wants to play desired game, he needs to click on green button Play Game! which initiates the selected game. There is additional info in the table as **player ID**, **game type**, **opponent**.

5.1.6 How user plays 3DGo game

3DGo is played similarly to classic Go. In the figure **5.4** we depict finite state machine of 3DGo game.

Liberty counting

Player can click on any stone that is already on board to display its liberties. The stone or whole group will be highlighted, and sprite will be displayed with number of liberties of that stone or group.

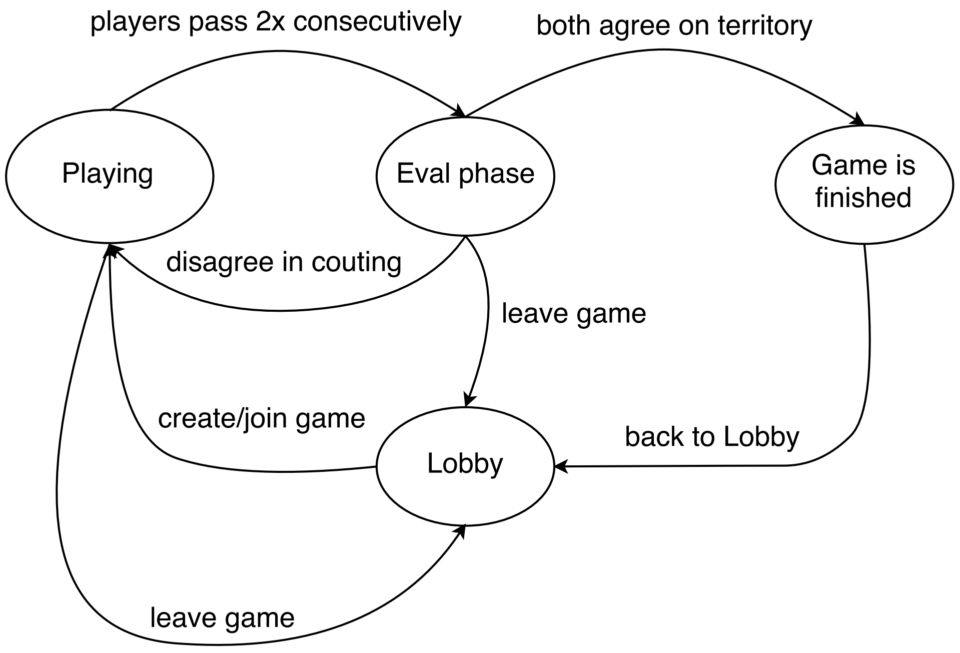


Figure 5.4: Finite state machine of user interaction with client.

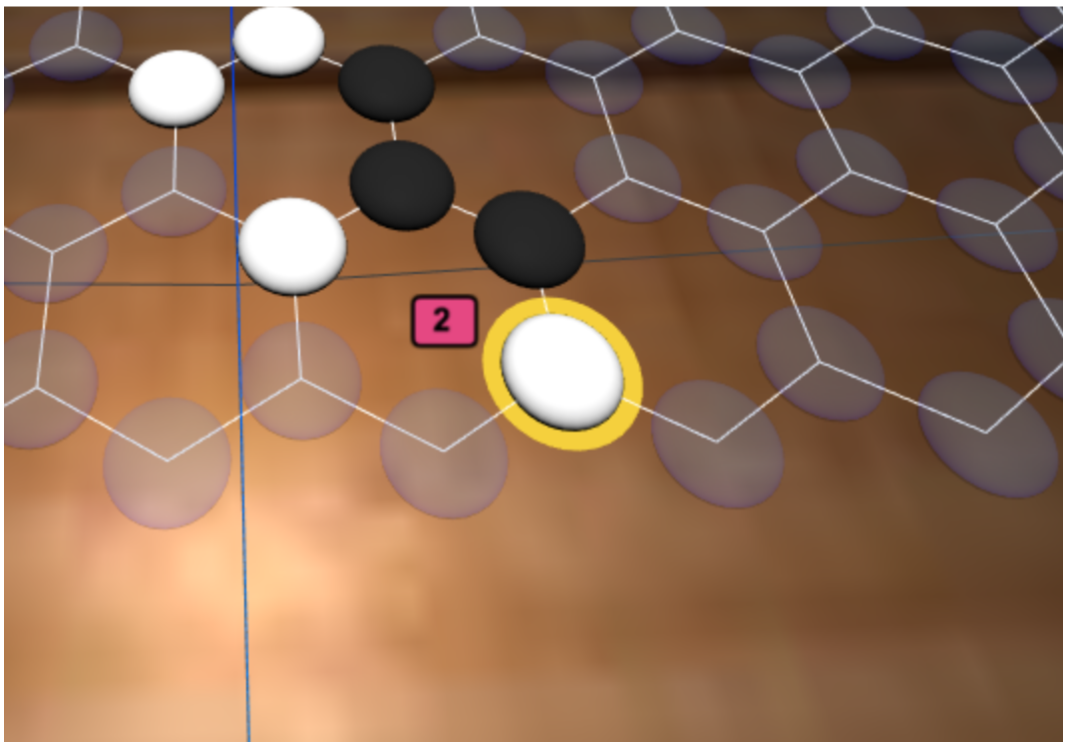


Figure 5.5: Librety counting of stones

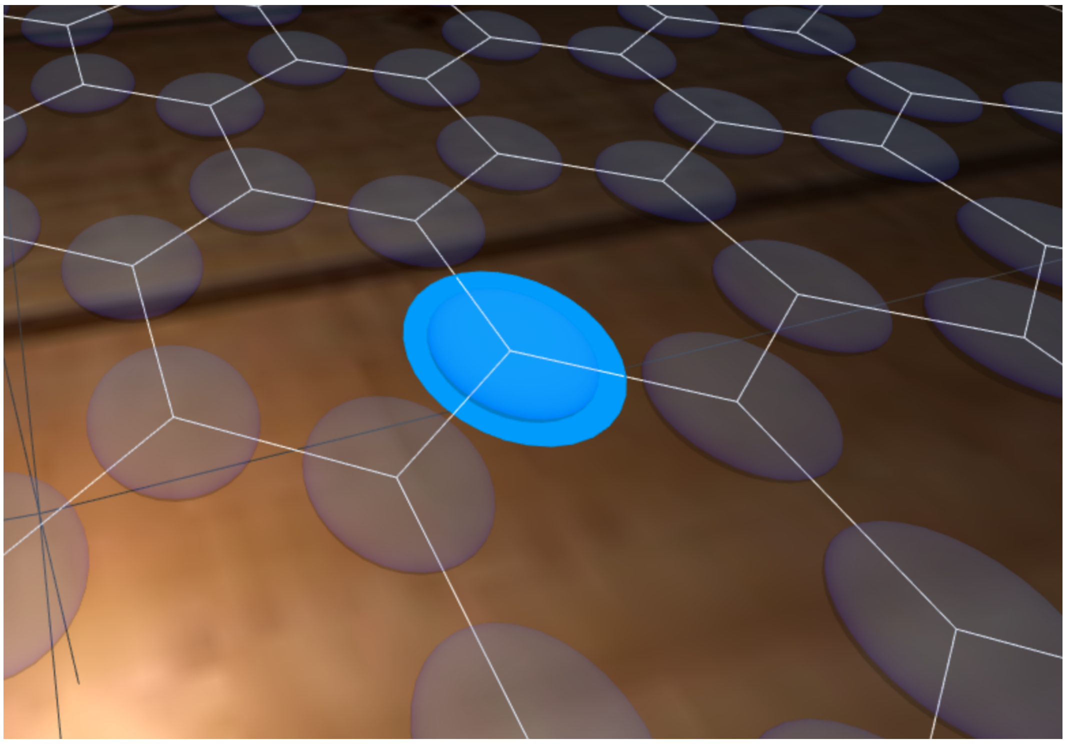


Figure 5.6: Stone selected to play on board

To prevent misclick, when user wants to play a move, he has to click on some blank space. Blue color is highlighting that selected stone. If he clicks on that stone again, it is sent to server and processed.

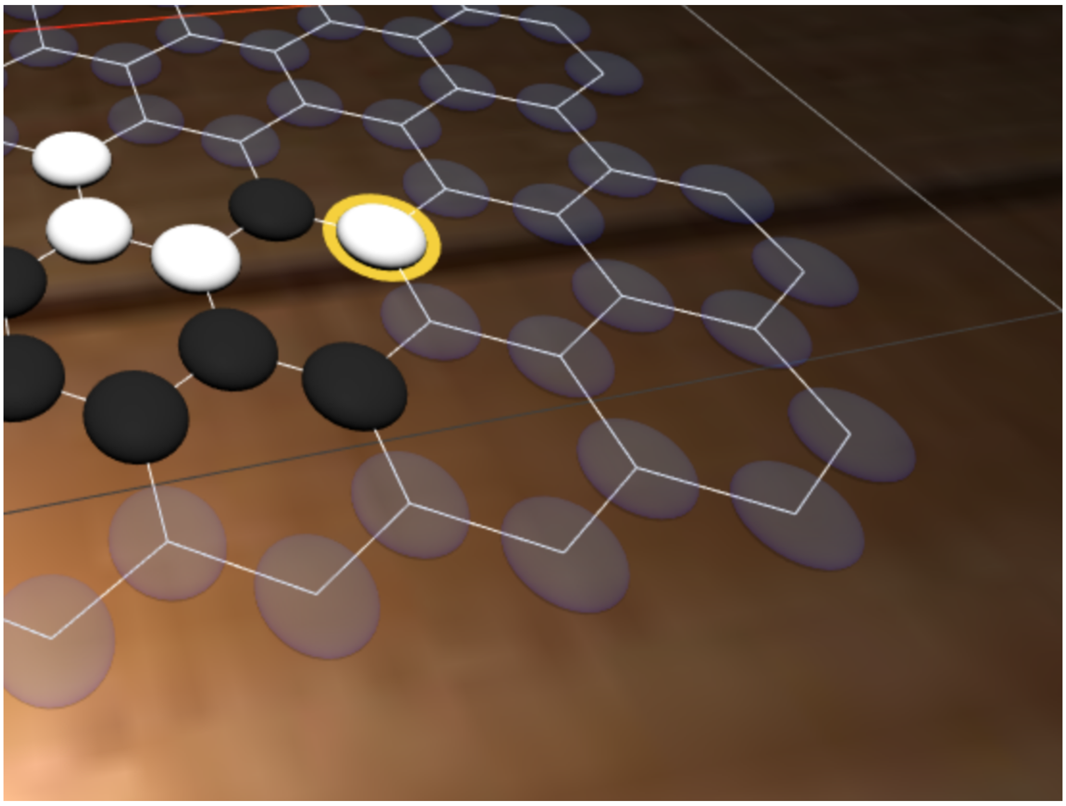


Figure 5.7: Last played stone on board

5.1.7 How to create new boards for 3DGo

Boards used by **3DGo** are stored as undirected graphs. We needed a way to create any board that a user could imagine. We could either implement our own editor, or use some existing software. Editor has to be **multiplatform**, **easily accessible** and **free**. Having considered these options, we settled on second one, because existing software is more flexible and has more creative options. Boards created using third party editor must be converted to undirected graph that **3DGo** uses.

Blender [57] is a very popular modelling tool satisfying our requirements. We have implemented loader which converts models in **.obj** [58] file format to our undirected graph data structure. Loader only reads lines beginning with **v** and **l**.

The .obj file we wish to convert has to have following format:

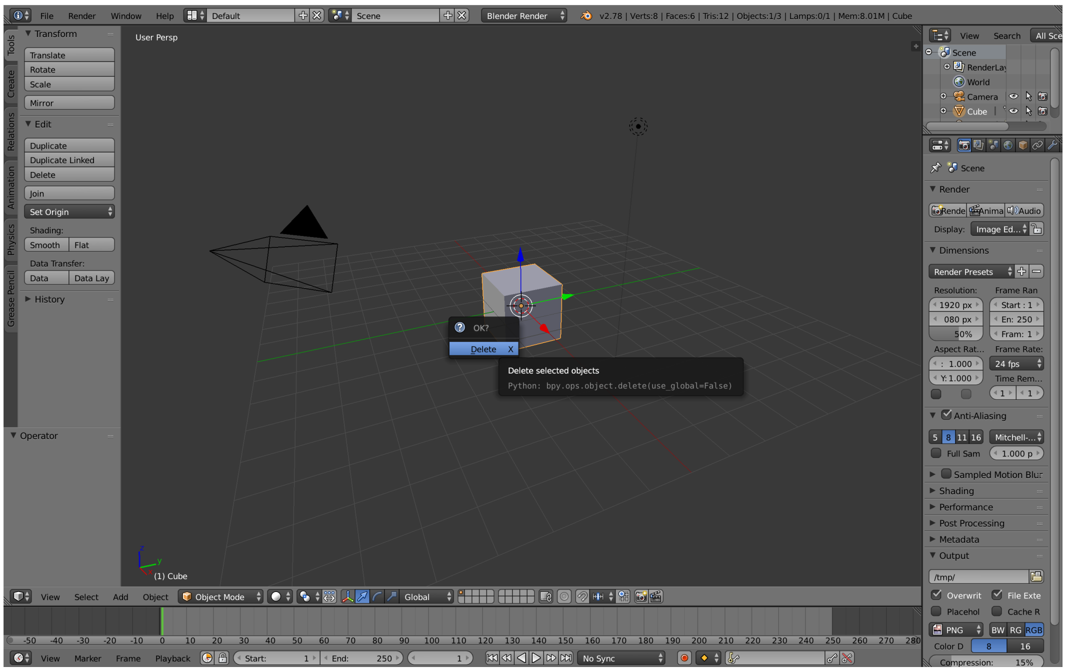
```
# Blender v2.78 (sub 0) OBJ File: ''
# www.blender.org
v -1.000000 0.000000 1.000000
v 1.000000 0.000000 1.000000
v -1.000000 0.000000 -1.000000
v 1.000000 0.000000 -1.000000
l 3 1
l 1 2
l 2 4
l 4 3
```

Lines begining with # are comments. Lines which begin with *v* means vertex and its x,y,z coordinates. Lastly, lines in format *l from to* are edges.

Board creation workflow

First, ensure you have installed up-to-date Blender. You can install latest Blender from https://www.blender.org. Below we describe how to create compatible .obj file for our loader.

1.) Open the Blender and delete default cube.

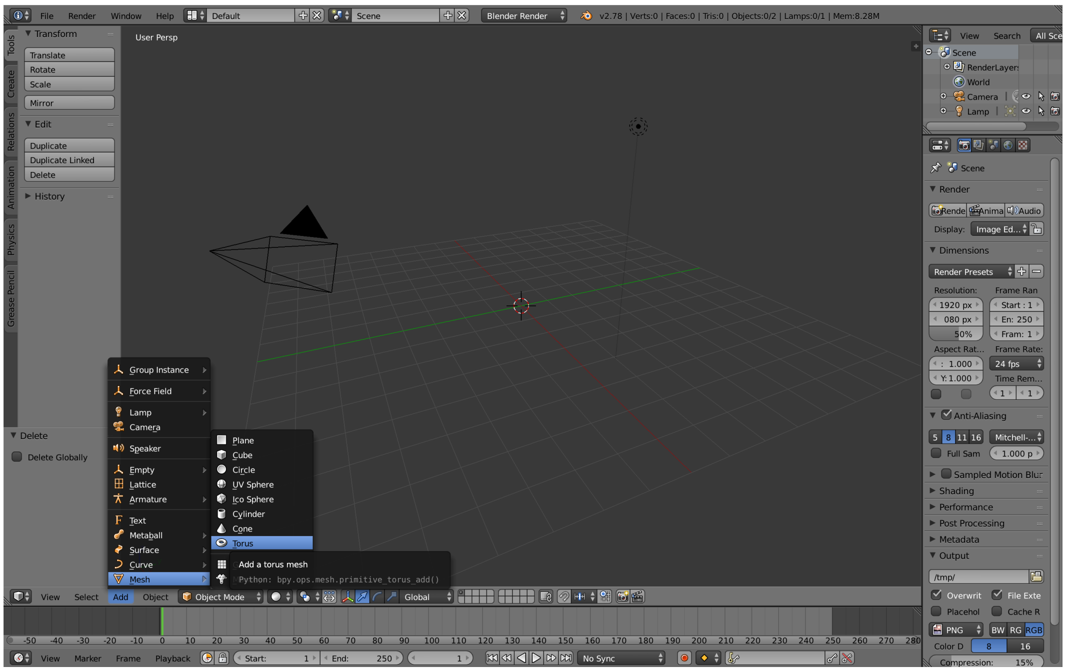


Delete selected objects

```
Python: bpy.ops.object.delete(use_global=False)
```

Figure 5.8: Blender, screenshot

2.) Add or create any mesh you want.

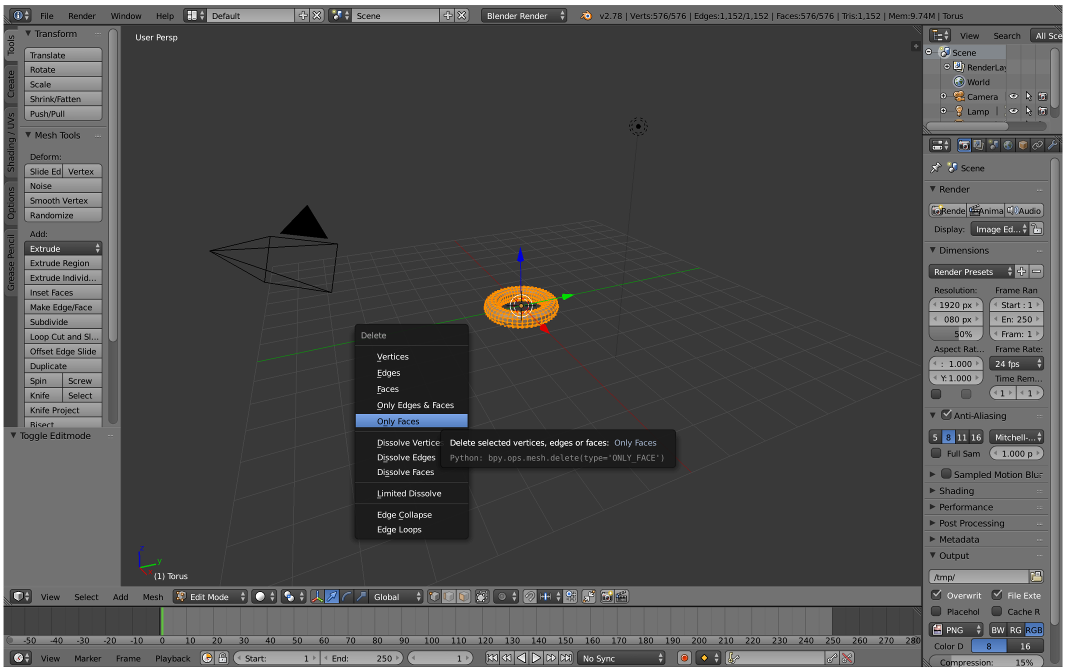


Add a torus mesh

```
Python: bpy.ops.mesh.primitive_torus_add()
```

Figure 5.9: Blender, screenshot

3.) After you are content with your creation, go to edit mode and press x and then delete **only faces**.



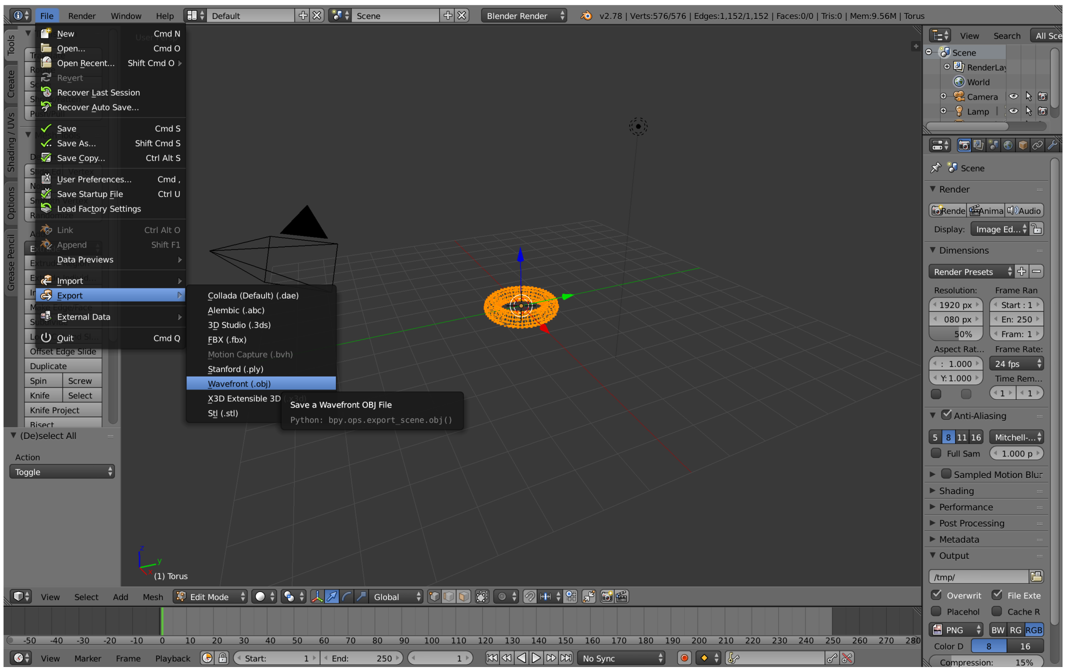
v2.78 | Verts:576/576 | Edges:1,152/1,152 | Faces:576/576 | Tris:1,152 | Mem:9.74M | Torus

Delete selected vertices, edges or faces: Only Faces

```
Python: bpy.ops.mesh.delete(type='ONLY_FACE')
```

Figure 5.10: Blender, screenshot

4.) Then, select *File* → *Export* → *Wavefront (.obj)*



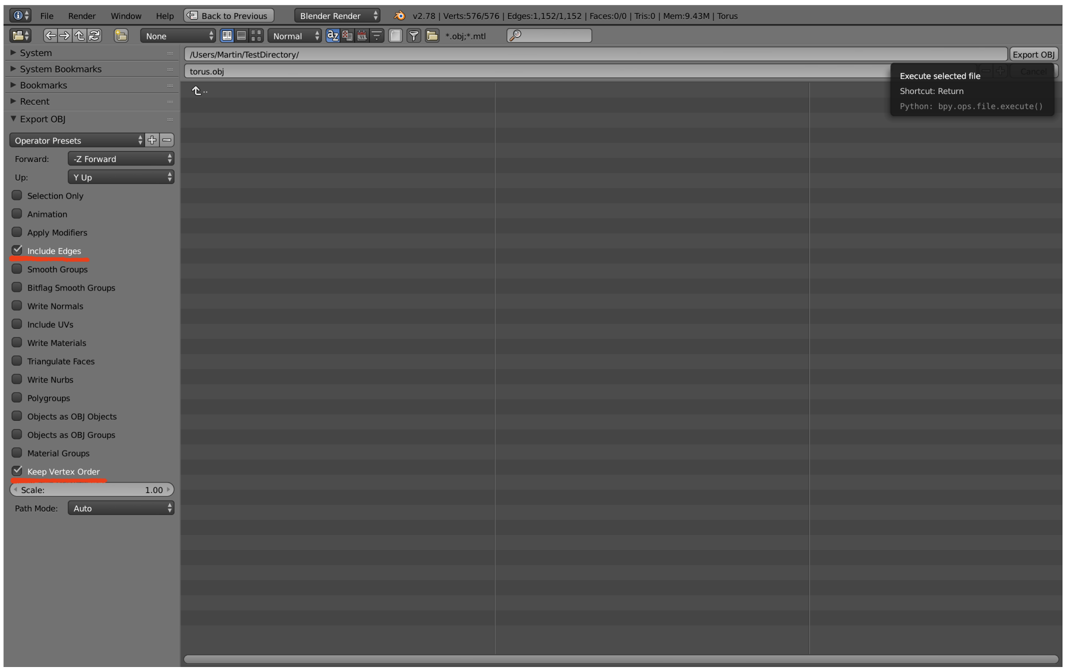
v2.78 | Verts:576/576 | Edges:1,152/1,152 | Faces:0/0 | Tris:0 | Mem:9.56M | Torus

Save a Wavefront OBJ File

```
Python: bpy.ops.export_scene.obj()
```

Figure 5.11: Blender, screenshot

5.) Save the file with **only** following options: *Include Edges*, *Keep Vertex Order* to the **/app/serverSide/boards**



v2.78 | Verts:576/576 | Edges:1,152/1,152 | Faces:0/0 | Tris:0 | Mem:9.43M | Torus

.obj;.mtl

/Users/Martin/TestDirectory/

torus.obj

Execute selected file
Shortcut: Return
Python: bpy.ops.file.execute()

Forward: -Z Forward
Up: Y Up
- [ ] Selection Only
- [ ] Animation
- [ ] Apply Modifiers
- [✓] Include Edges
- [ ] Smooth Groups
- [ ] Bitflag Smooth Groups
- [ ] Write Normals
- [ ] Include UVs
- [ ] Write Materials
- [ ] Triangulate Faces
- [ ] Write Nurbs
- [ ] Polygroups
- [ ] Objects as OBJ Objects
- [ ] Objects as OBJ Groups
- [ ] Material Groups
- [✓] Keep Vertex Order

Scale: 1.00
Path Mode: Auto

Figure 5.12: Blender, screenshot

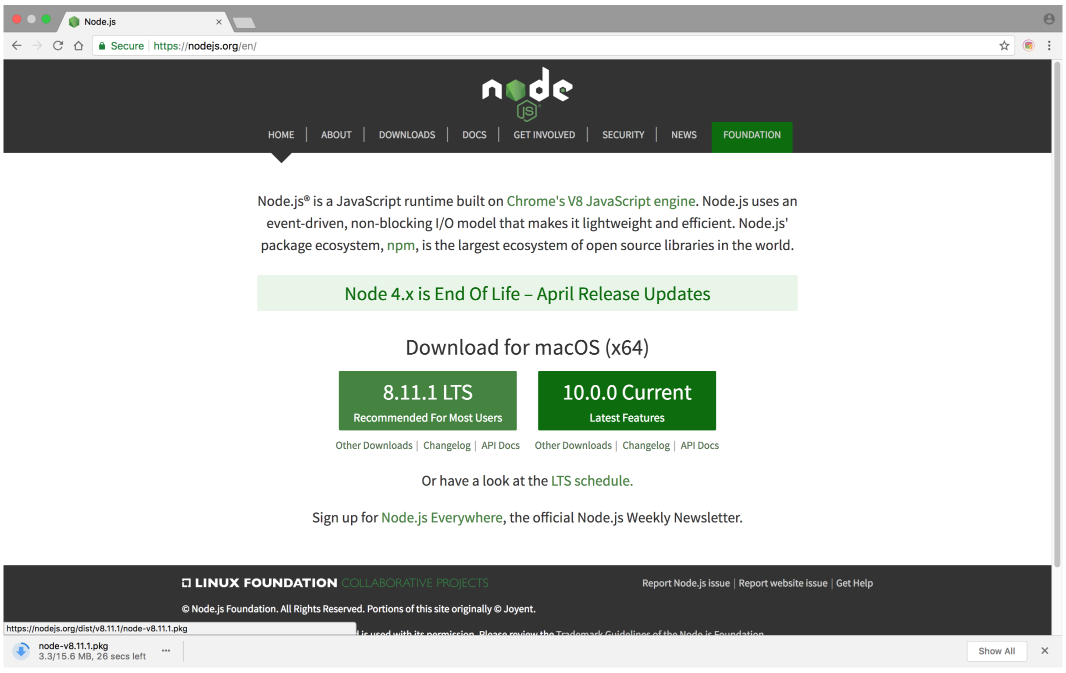
5.2 User documentation of Server

In this section we will describe how to set up Server. You can download up-to-date version of the application at: https://github.com/martinerk0/3dgo

5.2.1 How to install and run app

We show how to install and run our application on one platform, Mac OS X 10.11.6 (El Capitan).

1. Download current version of Node.js from website https://nodejs.org/en/


Node.js
Secure | https://nodejs.org/en/
HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | NEWS | FOUNDATION

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Node 4.x is End Of Life – April Release Updates

Download for macOS (x64)

8.11.1 LTS
Recommended For Most Users

10.0.0 Current
Latest Features

Other Downloads | Changelog | API Docs   Other Downloads | Changelog | API Docs

Or have a look at the LTS schedule.

Sign up for Node.js Everywhere, the official Node.js Weekly Newsletter.

LINUX FOUNDATION COLLABORATIVE PROJECTS   Report Node.js issue | Report website issue | Get Help

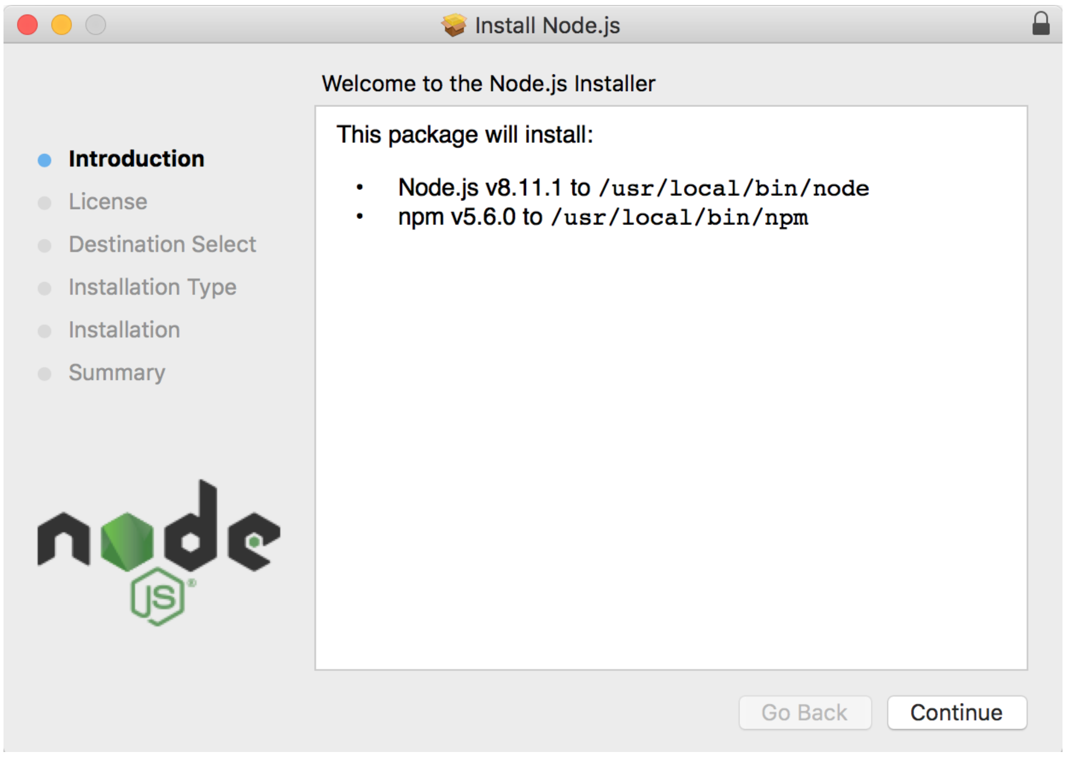
© Node.js Foundation. All Rights Reserved. Portions of this site originally © Joyent.

https://nodejs.org/dist/v8.11.1/node-v8.11.1.pkg

node-v8.11.1.pkg
3.3/15.6 MB, 26 secs left

Figure 5.13: Node, download

2. Run installer



Welcome to the Node.js Installer

This package will install:

- Node.js v8.11.1 to /usr/local/bin/node
- npm v5.6.0 to /usr/local/bin/npm

Figure 5.14: Node, download

3. Download the latest application from github repository, https://github.com/martinerk0/3dgo.

4. Unzip downloaded folder, and use terminal to cd to <name of downloaded directory>

5. Type in terminal: node app.js <number of port>, so for example node app.js 3000, server is now running on specified port.

6. open localhost:<number of port> or <your ip adress>: <number of port>, and play!

6. Development documentation

In this chapter we describe how we implemented our application. We strived for modular design by splitting each functionality in separate classes, and separate files. This was possible using Node.js import/export and es6 export/import recently introduced to javascript.

Our application consist of four main parts, which are split into respective parts:

- Server
- Games
- User Client
- AI Client

In next sections we will describe each part.

6.1 Server

Server is a Node.js application implemented in file app.js.
Node.js libraries are called modules, Server app depends on these modules:

- express [48]
- path [59]
- socket.io [50]

Server is the central part of application. It listens for clients to connect. Each client firstly connect to shared **room** [60] called lobby where they can create games or join games.
Game is represented by classes AvailableGame,RunningGame which contains info such as hostPlayer,joinPlayer and particular controller of the game. Players are identified by unique socket id assigned by library socket.io.

When client creates game, AvailableGame object is created and added to the list of available games and is displayed in the table. After another client joins this particular game, it is removed from list of available games.
Game controller is created, which creates model of the game and handles all the traffic between clients.
Game controller represents running game, and is added to the list of running games.

Relaying messages to the Game controller by server is useful, because we can have different game controllers and thus different running games on the same server at the same time, this is shown in figure **6.1** . Controllers of course must have same API.

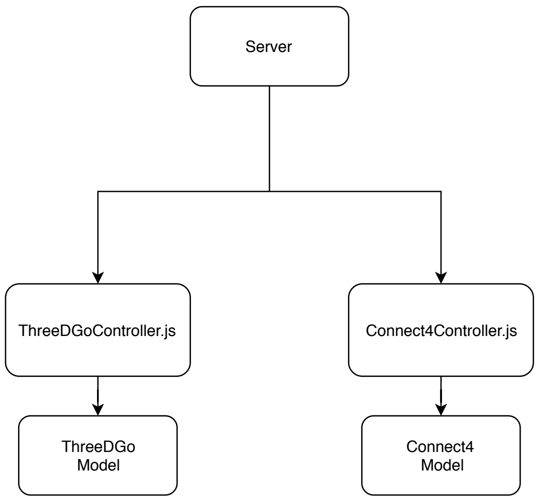


Figure 6.1: Example of two games

We also pass reference to the server object so that controller can call methods of server. This is useful for example when game is finished, and game controller needs to tell server to remove game from list of running games, so controller calls endGame method.

```
endGame(winner){
        let runningGame={
            hostPlayer: this.agent1.id,
            joinPlayer: this.agent2.id
        };
        let data = JSON.stringify(runningGame);
        this.server.endThisRunningGame(data);
    }
```

Figure 6.2: Example of end game method, which tells server to end this particular game

Usual lifecycle of a game is:

1. Server starts, no client is connected
2. Client 1 connects to the server
3. Client 1 creates a game
4. Client 2 connects to the server
5. Client 2 joins the game created by Client 1
6. Game has finished, clients are put to Lobby

This is illustrated in figure **6.3**

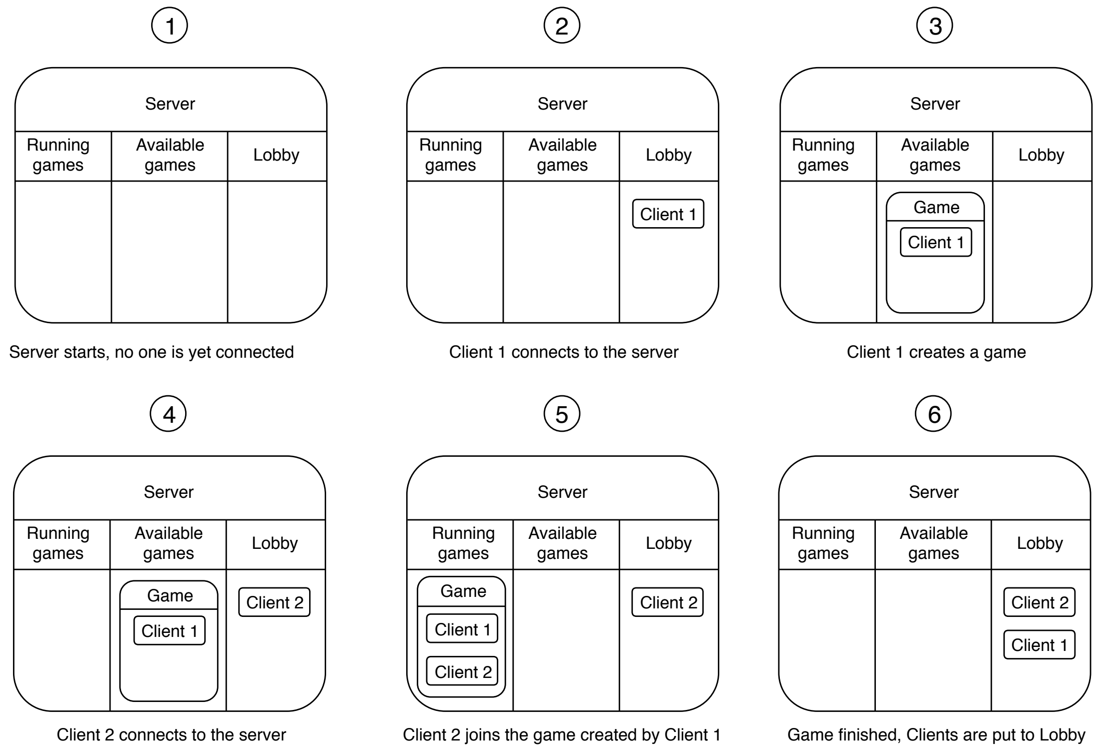


Figure 6.3: Example of lifetime of a game

6.1.1 Events

Server is using library **Socket.io** which is **event-driven**[61, p. 38] When server wants to send message to client, it emits event with some data

```
this.io.emit('nameOfEvent', someDataForRecipient);
```

Server has to listen for events from clients too:

```
socket.on('someEventFromClient', dataFromClient => {
        // process data
});
```

In figure **6.4** we have example of code that listens for event 'gameMessage', used in ClientController.

In figure **6.5** we have example of code emit event 'gameMessage', used in ClientController.

```
socket.on('gameMessage',message => {
    if(this.state==="playingGame"){
        this.GameController.processMessageFromServer(message)
    }
    else{
        alert("ERROR");
        console.log(this.state);
        console.log(message);
    }
});
```

Figure 6.4: Example of code that listens for 'gameMessage' events

```
sendGameMessage(message){
        message.senderID = this.socket.id;
        this.socket.emit('gameMessage',message);
}
```

Figure 6.5: Example of code that listens for 'gameMessage' events

6.1.2 Layered design of events

Socket.io has a few default events such as 'connection' and 'disconnect' . In addition to this, we defined our own events. We split these events into two layers for client-server part and for actual game playing messages. This modular design enables us to use any game with our client-server architecture, not just 3DGo. We used while developing our application on test game, Connect4. First layer is used for client - server communication, such as creating new game, joining game, updating game list in lobby and ending the game. Second layer is used for passing game-specific messages such as "play at 4th collumn" in game Connect4. This design is shown in figure **6.6**.

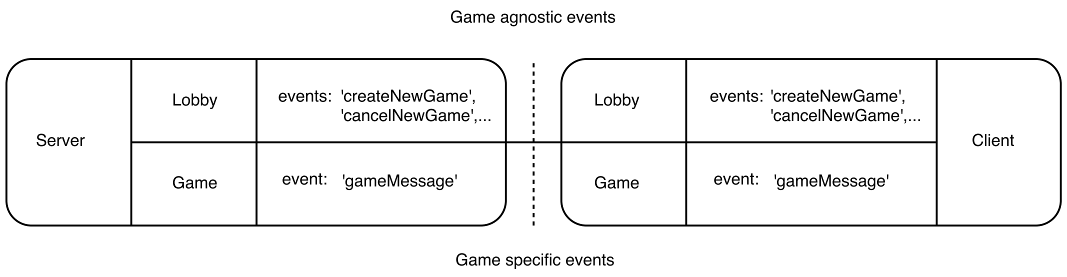


Figure 6.6: Layered communication between server and clients

Server events

These are all events that server emits and listens to, with their meaning. Server emits following events:

- ’gameHasFinished’ - this event is sent to clients in a running game and signals that game is finished
- ’userDisconnected’ - notifies clients that someone has disconnected
- ’updateList’ - notifies clients and gives them updated list of available games
- ’playGame’ - is sent to clients and signals them that game begins

And listens for following messages:

- ’createNewGame’ - is used when client wants to create a new game
- ’cancelNewGame’ - is used when client wants to cancel new game
- ’finishedGame’ - is used when client wants to exit game, I guess it should be called exit game because it suggest peaceful end of the game
- ’joinGame - is used when a client wants to join a particular available game
- ’gameMessage’ - event used by games, it is relayed to specific game controller

6.2 Games

Although our main focus is on implementing 3DGo game functionality, we designed our application to be very modular and support multiple games. This allowed us to streamline class design and also implement out testing game, Connect4 with much ease. We will show how additional games could be added to our application, either for research or casual purposes. Thus our application can be used for testing various AIs in one place for various games for example. Each game roughly correspond to MVC patterns and thus has three parts, Model, View and Controller.

6.2.1 Controller

Controller takes care of:

- processes messages from clients
- processes messages from server
- processes input from user
- sends messages to and from client and server

6.2.2 Model

Model takes care of:

- state of the game,
- applying user moves to model,
- has support functions for AI and View, e.g. get all valid moves.

6.3 Client

In this section we describe client, its functions and components. Client is the part of the application that is running in the web browser. User interacts with client by mouse or touch events. Client displays HTML pages served by server and displays 3d scenes using javascript library Three.js. Javascript files are imported by following HTML statements:

```
<script  src="socket.io/socket.io.js"></script>
<script  src="lib/popper.js"></script>
<script  src="lib/jquery-3.2.1.min.js"></script>
<script  src="lib/bootstrap.min.js"></script>
<script  type="module" src="js/ClientController.js"></script>
```

Notice last line, type="module", that means it is es6 import.

The parts of the client are:

- ClientController
- Controller
- View
- Model

6.3.1 ClientController

Client Controller is the main javascript client program. It:

- manages user input
- draws list of games
- initialized game controller.
- communicates with server

Client emit these events:

```
'joinGame','createNewGame','cancelNewGame','finishedGame'
```

and listens for these events:

```
'connect','gameMessage','updateList','playGame',
'gameHasFinished','userDisconnected','disconnect'
```

6.3.2 GameController

Game controller is responsible for managing input from user.

It sends only one event:

```
'gameMessage'
```

with message:

```
let  message = {
    name: 'makeMove',
    move : move
};
```

6.3.3 Model

Model on Client is the same as on Server. It manages state of the game. Which Model is selected depends on user choice
We further describe ThreeDG:

It can have following states:

```
"NOT_GAME_STATE", "MAKING_MOVE", "WAITING_FOR_TURN",
"EVAL_PHASE", "GAME_ENDED"
```

ThreeDG is using classes:

- UnorderedGraph
- Node
- Spot

6.3.4 View

View provides graphical output to the user. Each game has its respective View. We implemented 3D rendering for 3DGo with JavaScript library Three.js and simple text based View for Connect4 to showcase usage of different Views. We redraw View when something changes, such as user rotates scene by mouse or clicks.

ThreeView

ThreeView is class encapsulating rendering of the 3DGo. Since graph is represented by vertices and edges, we need a way to draw them. We use Three.js library for that. We firstly create scene, renderer and then add objects to the scene. Objects are stones from graph.Stones are there from the beginning, changing color and transparency signalises that position is blank, black or white. Edges are drawn by lines.

6.4 AI Client

Our AI agent is programmed using client server approach. One way would have been to run AI as some server based subroutine. Other way, which we chose is to have AI agent as separate client, spawned on demand by server. It thus consist of similar parts as human client.

- ClientController
- Controller
- Model

This lets us use the same architecture on server. For sever, the AI client is just another ordinary client. We use client part of socket.io library for Node.js apps.

6.4.1 How AI client connects to the game

When player chooses to play with AI, server:

1.) creates new available game

2.) forks new node AI process with ID of game to join

3.) does not add it to list of visible games

4.) AI connects to the game

Here is a diagram describing this with respect to time.

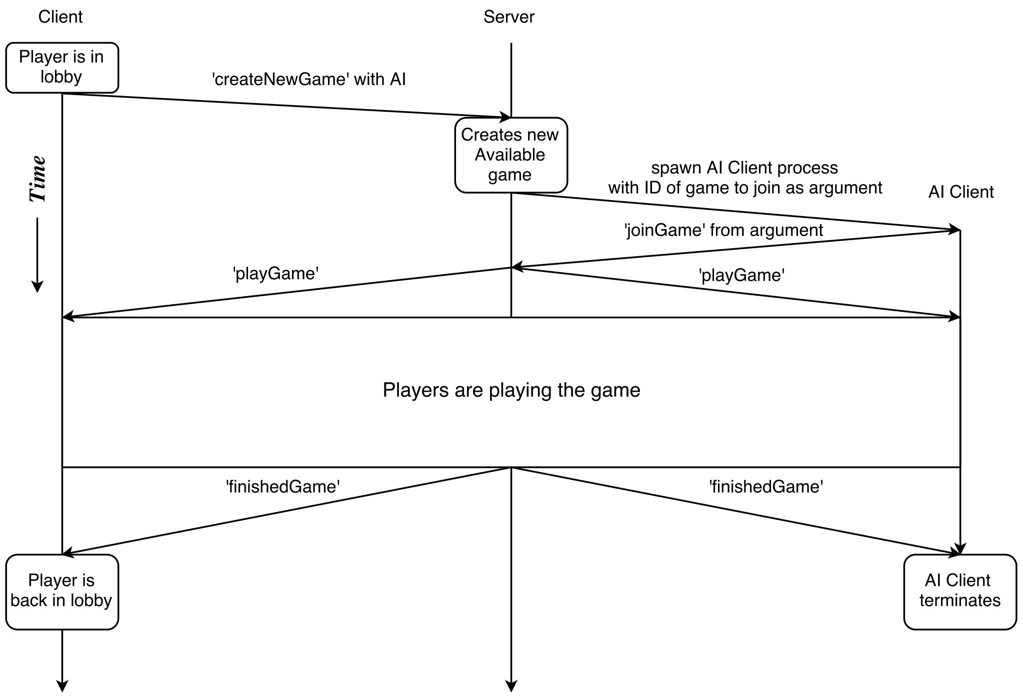


Figure 6.7: AI client connection to server

6.5 Addition of other games by other developers

If a developer wants to add another game to our app, he must implement the following classes.

- Game Controller
- View
- Model

7. Conclusion

In this thesis we developed an application that lets to user play Go defined by arbitrary graph, as we defined in **Goal 1.)** from introduction.

Our application also allows a user to play Go against other users on the internet, as was required by **Goal 2.)**. For this task, we selected web based client server architecture. This approach has many advantages. Today, more than 50% of population of Earth is connected to the Internet, and most computers have installed web browsers. Our application is multiplatform, and does not require installation of the client.

By **Goal 3.)**, we implemented a way to play against AIs. Because we are using client-server architecture, AI can connect to the server as a client and therefore it can run on another machine. We used protocol Websockets, that allow fast duplex communication without much overhead, which is especially suited for web games. We tried to implement an artificial intelligence based on Monte Carlo tree search, but this did not produce satisfying results. We can improve this in the future work.

We implemented rendering of Go boards represented as graphs, as we stated in **Goal 4.)**. For this task we selected 3D rendering. Because we are using web browser as client, we used modern technology WebGL. We also implemented controls that allow the user to rotate, zoom and pan the view, and thus achieving better user experience.

Lastly, we wanted to give the user the ability to create his own non standard Go boards and to play them in our application. For this task we chose free open source modelling tool Blender. User can use all the functionality of Blender to design boards and then exports them in the suitable subset of .obj format. These exported boards are then loaded into our application and thus we fulfilled **Goal 5.)**.

Future work

Because computer Go and artificial intelligence is a fairly complex topic, we present some possible improvements that could be done in the future.

Since our implementation of MCTS algorithm did not produce satisfying results, we could look for the reasons why.

Because we have designed our application modularly, it is be possible to add other games to the already implemented client-server architecture. Since any artificial intelligence can be developed for our application, additional artificial intelligences, regardless of platform or programming language or game may be added.

In our next work, we could also focus on extending the application to the database of users. Users will be able to log in, log off, save game history. For example, a profile might be showing rank relative to the other players.

Bibliography

[1] Wikipedia. (Mar. 20, 2018). AlphaGo versus Ke Jie, [Online]. Available: https://en.wikipedia.org/wiki/AlphaGo_versus_Ke_Jie (visited on 05/16/2018).

[2] ——, (2018). Rules of Go, [Online]. Available: https://en.wikipedia.org/wiki/Rules_of_Go#Reference_statement (visited on 05/15/2018).

[3] American Go Association. (2018). Official AGA Rules of Go, [Online]. Available: http://www.cs.cmu.edu/~wjh/go/rules/AGA.html (visited on 05/15/2018).

[4] W. M. Shubert. (2000). KGS Go Server, [Online]. Available: http://www.gokgs.com/help/index.html (visited on 05/15/2018).

[5] British Go Association. (Oct. 23, 2017). Definition of the Smart Game Format, [Online]. Available: http://www.britgo.org/tech/sgfspec.html (visited on 05/15/2018).

[6] G. Farnebäck. (2002). Go Text Protocol, [Online]. Available: http://www.lysator.liu.se/~gunnar/gtp/ (visited on 05/15/2018).

[7] Online-go server. (2018). Online-go server, [Online]. Available: https://online-go.com (visited on 05/15/2018).

[8] L. Geselowitz. (2004). Freed Go 1.1, [Online]. Available: http://www.leweyg.com/lc/freedgo.html (visited on 05/12/2018).

[9] D. S. Sylvain Gelly, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Elsevier*, 2011.

[10] M. Campbell, A. J. Hoane, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[11] B. Bouzy and T. Cazenave, "Computer Go: an AI oriented survey," *Artificial Intelligence*, vol. 132, no. 1, pp. 39–103, 2001.

[12] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *International conference on computers and games*, Springer, 2006, pp. 72–83.

[13] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[15] Wikipedia. (2018). Monolithic application, [Online]. Available: https://en.wikipedia.org/wiki/Monolithic_application (visited on 05/11/2018).

[16] A. S. Tanenbaum and D. Wetherall, *Computer networks*. Prentice hall, 1996.

[17] Miniwatts Marketing Group. (2018). Internet Usage Statistics, [Online]. Available: https://www.internetworldstats.com/stats.htm (visited on 05/11/2018).

[18] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley. (2013). The Java Virtual Machine Specification, [Online]. Available: https://docs.oracle.com/javase/specs/jvms/se7/html/index.html (visited on 05/12/2018).

[19] Microsoft. (Apr. 16, 2018). Common Language Runtime, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/clr (visited on 05/12/2018).

[20] Unity Technologies. (2018). Unity, [Online]. Available: https://unity3d.com (visited on 05/13/2018).

[21] Unity. (2018). Unity WebGL Browser Compatibility, [Online]. Available: https://docs.unity3d.com/Manual/webgl-browsercompatibility.html (visited on 05/15/2018).

[22] T. Berners-Lee. (2018). The WorldWideWeb browser, [Online]. Available: https://www.w3.org/People/Berners-Lee/WorldWideWeb.html (visited on 05/12/2018).

[23] J. Peterka. (2015). Rich Internet application, [Online]. Available: http://www.earchiv.cz/l226/slide.php3?l=10&me=30 (visited on 05/12/2018).

[24] Google. (2018). Google Docs, [Online]. Available: https://www.google.com/docs/about/ (visited on 05/12/2018).

[25] Adobe. (2018). Adobe Flash Player, [Online]. Available: https://www.adobe.com/products/flashplayer.html (visited on 05/13/2018).

[26] C. Warren. (Nov. 19, 2012). The Life, Death and Rebirth of Adobe Flash. Mashable, Ed., [Online]. Available: https://mashable.com/2012/11/19/history-of-flash/#YIwDjkGKoPqI (visited on 05/13/2018).

[27] S. Jobs. (2010). Thoughts on Flash, [Online]. Available: https://www.apple.com/hotnews/thoughts-on-flash/ (visited on 05/13/2018).

[28] Adobe Corporate Communications. (Jul. 25, 2017). Flash & The Future of Interactive Content, [Online]. Available: https://theblog.adobe.com/adobe-flash-update/ (visited on 05/13/2018).

[29] C. of Wonders. (2015). Civilization Wars, [Online]. Available: https://www.bubblebox.com/play/adventure/1603.htm (visited on 05/13/2018).

[30] Flare3D. (2018). Yellow Planet, [Online]. Available: http://www.flare3d.com/demos/yellowplanet/ (visited on 05/13/2018).

[31] Microsoft. (May 13, 2018). About Silverlight, [Online]. Available: https://www.microsoft.com/silverlight/what-is-silverlight/ (visited on 05/13/2018).

[32] J. Smith. (Jul. 2, 2015). Moving to HTML5 Premium Media, [Online]. Available: https://blogs.windows.com/msedgedev/2015/07/02/moving-to-html5-premium-media/ (visited on 05/13/2018).

[33] Mozilla. (Jan. 27, 2018). The WebGL API: 2D and 3D graphics for the web, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (visited on 05/13/2018).

[34] B. Jones. (Jul. 2, 2014). Quake, [Online]. Available: http://media.tojicode.com/q3bsp/ (visited on 05/13/2018).

[35] NASA. (2018). Experience Curiosity, [Online]. Available: https://eyes.nasa.gov/curiosity/ (visited on 05/13/2018).

[36] T. Parisi, *Programming 3D applications with HTML5 and WebGL: 3d animation and visualization for web pages.* " O'Reilly Media, Inc.", 2014.

[37] R. Cabello, B. Ulicny, *et al.* (2018). Three.js library, [Online]. Available: https://threejs.org (visited on 05/13/2018).

[38] N. Drake and B. T. Jacobs. (2017). Cassini's Grand Tour. N. Geographic, Ed., National Geographic, [Online]. Available: https://www.nationalgeographic.com/science/2017/09/cassini-saturn-nasa-3d-grand-tour/ (visited on 05/13/2018).

[39] Google. (2014). 100,000 Stars, [Online]. Available: http://stars.chromeexperiments.com (visited on 05/13/2018).

[40] Node.js Foundation. (2018). Node.js, [Online]. Available: https://nodejs.org/en/ (visited on 05/14/2018).

[41] StackOverflow. (2017). Developer Survey Results 2017, [Online]. Available: https://insights.stackoverflow.com/survey/2017#technology-programming-languages (visited on 05/14/2018).

[42] GitHub. (2018). Atom text editor, [Online]. Available: https://atom.io (visited on 05/14/2018).

[43] Microsoft. (2018). Visual Studio Code, [Online]. Available: https://code.visualstudio.com (visited on 05/14/2018).

[44] Google. (2018). Chrome DevTools, [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/ (visited on 05/14/2018).

[45] Apple. (2018). Safari Web Development Tools, [Online]. Available: https://developer.apple.com/safari/tools/ (visited on 05/14/2018).

[46] JetBrains. (2018). Webstorm, [Online]. Available: https://www.jetbrains.com/webstorm/ (visited on 05/14/2018).

[47] Bootstrap Core Team. (2018). Bootstrap, [Online]. Available: https://getbootstrap.com (visited on 05/15/2018).

[48] StrongLoop and IBM. (2018). Express framework, [Online]. Available: https://expressjs.com (visited on 05/15/2018).

[49] Internet Engineering Task Force. (2011). The WebSocket Protocol, [Online]. Available: https://tools.ietf.org/html/rfc6455 (visited on 05/14/2018).

[50] Automattic. (2018). Socket.IO library, [Online]. Available: https://socket.io (visited on 05/15/2018).

[51] F. S. Foundation. (2009). Gnu go, [Online]. Available: https://www.gnu.org/software/gnugo/ (visited on 05/15/2018).

[52] P. Baudiš, "MCTS with Information Sharing," Master's thesis, Charles University in Prague Faculty of Mathematics and Physics, 2011.

[53] R. Coulom. (2018). Crazy Stone, [Online]. Available: https://www.remi-coulom.fr/CrazyStone/ (visited on 05/15/2018).

[54] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[55] G.-C. Pascutto. (2017). Estimate of computing power required to train AlphaGo, [Online]. Available: http://computer-go.org/pipermail/computer-go/2017-October/010307.html (visited on 05/15/2018).

[56] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[57] Blender Foundation. (2018). Blender, [Online]. Available: https://www.blender.org (visited on 05/15/2018).

[58] Wavefront. (2018). Wavefront .obj documentation, [Online]. Available: http://www.martinreddy.net/gfx/3d/OBJ.spec (visited on 05/15/2018).

[59] Node.js Foundation. (2018). Node Path API, [Online]. Available: https://nodejs.org/api/path.html (visited on 05/15/2018).

[60] Socket.io. (2018). Socket.io documentation, [Online]. Available: https://socket.io/docs/ (visited on 05/15/2018).

[61] D. Flanagan, *JavaScript: the definitive guide.* " O'Reilly Media, Inc.", 2006.

[62] JSDoc. (2018). Jsdoc, [Online]. Available: http://usejsdoc.org/index.html (visited on 05/15/2018).

List of Figures

Attachments

Appendix A

The attachment contains three directories. Firstly, directory **app**, the application directory, which contains:

- **__JSDoc**, directory containing support files for generation of documentation
- **clientSide**, directory containing code used by client
- **node__modules**, directory with node.js packages and libraries
- **serverSide**, directory containing code used by server
- **app.js**, main script of the application
- **package.json**,**package-lock.json**, files used for installing packages
- **README.TXT**

It also contains directory **jsdoc** which contains documentation generated from source code by **JSDoc** [62]. You can view it by opening index.html file inside the **jsdoc** directory.

Lastly, attachment contains directory **thesis**, which contains PDF document of this thesis.