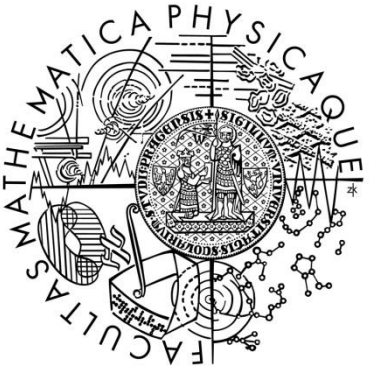
Charles University in Prague

Faculty of Mathematics and Physics

# MASTER THESIS




Bc. Matyáš Brenner

## Artificial Intelligence for Quoridor Board Game

Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Jakub Gemrot

Study programme: Computer Science

Specialization: Software Systems

Prague 2015

First let me thank to Mgr. Jakub Gemrot who kindly agreed to supervise this work. His guidance and counselling were invaluable to me.

Big thanks also go to my proof-readers: My dearest friend Simone Schuster and Ing. Lenka Matějíčková. Their fields of interests are completely different, so I can imagine that going through this text might have been exhausting, at best. Thank you girls for your big help and your friendship! I owe you one.

Finally let me thank my girlfriend Silvie Hrubá who did a great job of keeping me sane during the work on this thesis. Together with my mother she took a good care of me while suffering from my moods.

Finally a big thanks goes to my employer, company Z.L.D., s.r.o., who kindly let my work on the thesis while being under pressure generated by our clients.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In …..... date …..... signature

Název práce: Umělá inteligence pro hru Quoridor

Autor: Bc. Matyáš Brenner

Katedra / Ústav: Katedra softwaru a výuky informatiky

Vedoucí diplomové práce: Mgr. Jakub Gemrot

Abstrakt:
Cílem práce je navrhnout umělou inteligenci pro hru Sector 66, což je desková hra založená na hře Quoridor. Sector 66 umožňuje oproti Quoridoru navíc používat kouzla a políčka se speciálními efekty.

Umělá inteligence je založena na algoritmu Monte Carlo Tree Search a je použitelná pro 2 až 4 hráče. Představená umělá inteligence pracuje s vysokým větvícím faktorem hry Quoridor/Sector 66 a umí si poradit s neznámými prvky v podobě uživatelských zásuvných modulů.

Hra a umělá inteligence je vyvinuta za použití platformy .NET, XNA a jazyka C#.

Klíčová slova: Umělá inteligence, Monte-Carlo Tree Search, Desková hra Quoridor

---

Title: Artificial Intelligence for Quoridor Board Game

Author: Bc. Matyáš Brenner

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Jakub Gemrot

Abstract:
The aim of this work is to design an Artificial Intelligence for Sector 66, which is a board game based on Quoridor. In Sector 66 there is a possibility to use spells and fields with some special effects.

The Artificial Intelligence is based on Monte Carlo Tree Search. It can be used for 2 to 4 players. The Artificial Intelligence introduced in this work can work with the high branching factor of Quoridor/Sector 66 Game and can also handle unknown elements represented by user defined plug-ins.

The game and the Artificial Intelligence has been developed using .NET Framework, XNA and C#.

Keywords: Artificial Intelligence, Monte-Carlo Tree Search, Quoridor board game

# Content

# 1 Introduction

This chapter contains a description of bachelor thesis "Sector 66 – A Modular Board Game" [1] which is used as a base of this thesis. It also contains an overview of the Quoridor board game developed by Gigamic and the game Sector 66, developed in mentioned bachelor thesis, which was based on Quoridor.

## 1.1 Structure of the thesis

This thesis is divided into six chapters. This first one describes the games on which the thesis is based and also the goals of this thesis.

Chapter 2 gives a general overview of methods used in the field of Artificial Intelligence.

Chapter 3 describes how the goals of the thesis are met and what alternatives may be used.

In Chapter 4 there is the development documentation which further describes how the decisions made in Chapter 3 are implemented.

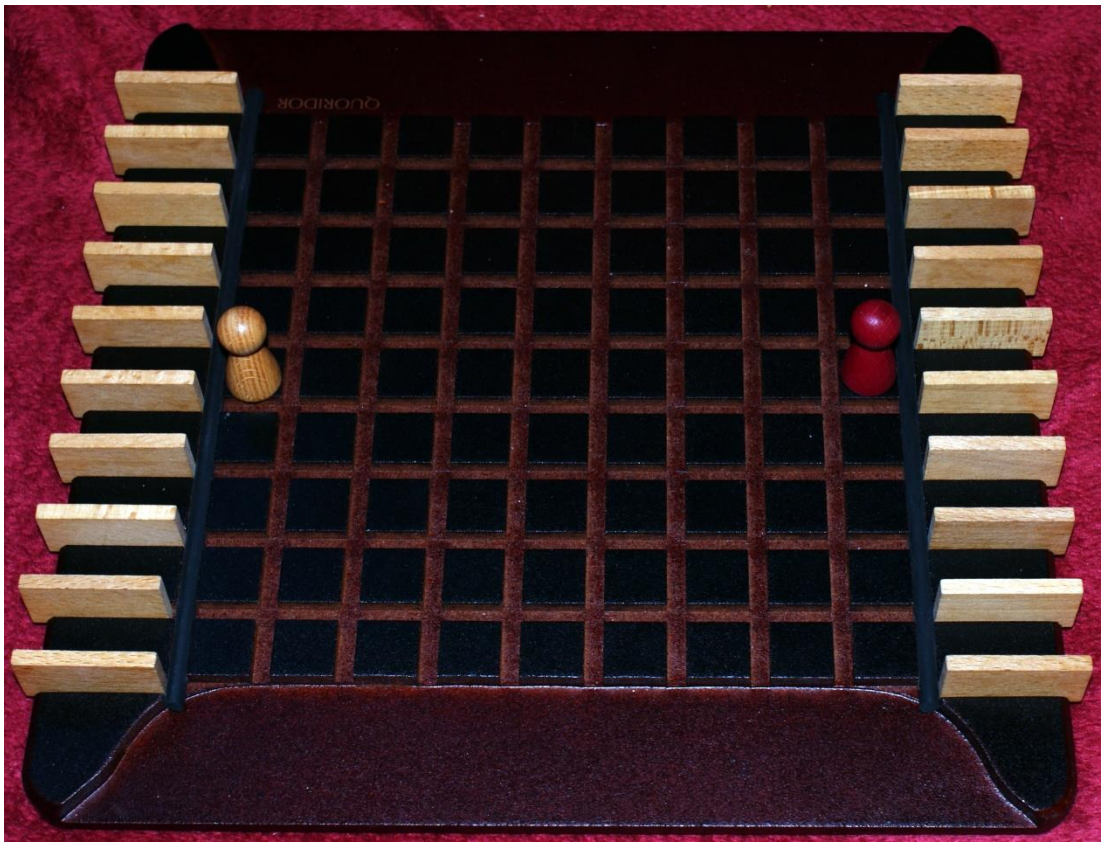
In Chapter 5 there is a user documentation which serves as a manual for the end user of the game of Sector 66.

Chapter 6 sums up the whole thesis and proposes some conclusions.

## 1.2 Quoridor

Quoridor is a board game developed by Gigamic. It is a game for two or four players played on a board of $9 \times 9$ fields. Between the fields there is a gap. Players are playing in turns and can either move their pawn or place a wall in the gap between the fields. The wall occupies a space equal to the length of two fields. The pawns cannot move over a built wall. The goal of the game is to cross the board of the game sooner than the opponent(s).

How the board looks like and the initial figure placement is shown in Picture 1. There are also unused walls there.




*Picture 1. Quoridor*

## 1.3 Sector 66

Sector 66 is a computer board game introduced in bachelor thesis "Sector 66 – A Modular Board Game" [1], which is based on Quoridor.

The basic concept of the game is the same as in Quoridor, but there are some additional features. It is possible to use spells. A spell is an object which first must be picked up by stepping on a field where the spell is placed. After picking it up the player can use it as a third kind of move instead of moving a pawn or placing a wall. The spells can be targeted up on a field, a player or a wall. A player can hold up to four spells at the same time. The spells are developed as plug-ins, so any user with programing knowledge can develop their own and use it in the game.

The second extra feature are field plug-ins. Users can implement not only spell plug-ins but field plug-ins as well. These fields can then be placed in the game board instead of the original ones. These special fields can have different effects, for example there can be a field on which a player is not allowed to step on.

The last extra feature introduced in the bachelor thesis is a template editor. The game is based on templates. The templates can modify some rules. It can define where the

players start, where they must get to win, if spells are allowed in the game and if there are some spells forced in the game. It is also possible to state for how many players the game is intended. Generally the game can be played by two to four players, but by using this feature it can be further limited, so for example it is possible to create a template of a game meant exactly for three players. The last thing which can be defined in the template is the initial look of the game board. The types of the fields can be changed using the plug-ins and also some walls can be prebuilt or disabled.

Sector 66 can be played locally on one computer or over LAN (local area network).

There is also an Artificial Intelligence introduced in the bachelor thesis to play against the computer, but it is practically unusable in every way. It consumes a lot of resources and the result is definitely not worth it. This Artificial Intelligence is so unintelligent that anyone can win against it.

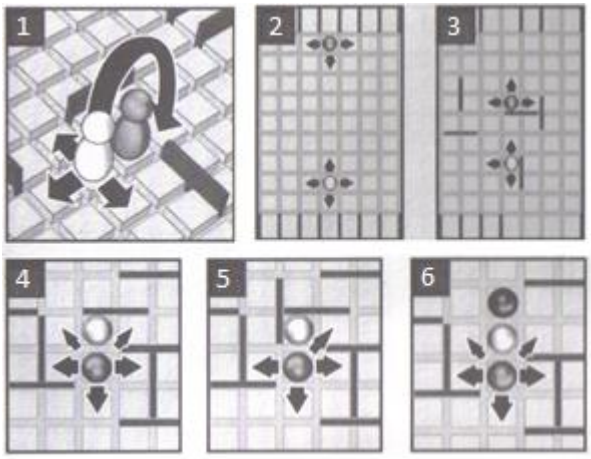
### 1.3.1 Game rules

In this chapter there are stated the rules of Sector 66 game as they are in the bachelor thesis [1].

The game is played on a board consisting of 9 × 9 fields. The goal of the game is to get to one of the player's target fields sooner than any opponent reaches one of his/hers target fields.

Target and start positions of the players are defined by the game template chosen when creating a game.

Players make moves in rounds one by one. To achieve the goal of the game they can do one of three actions in each round. A player can move their figure according to the diagram shown in Picture 2, build a wall or cast a spell if allowed by the template used. The wall can be built according to the same rules as in Quoridor game. This means that it can be placed between fields and after placing it there must be left at least one way to at least one of the target fields for each player.




*Picture 2. Pawn moves*[1]

The spells are generated according to the settings during the game. The users can specify the probability of spell generation and also the generation interval measured in rounds. A spell is picked by stepping on one. A player can hold four spells tops at the same time.

The player who reaches a target field first is the winner of the game.

All these rules are actively enforced by the game.

## 1.4 Bachelor thesis

In the bachelor thesis [1] following steps has been done.

1. Game design
2. Game graphics
3. Analysis and implementation of the Sector 66 Game
4. Analysis and implementation of the Template Editor
5. Analysis and implementation of service for hosting LAN games
6. Analysis and implementation of the Translation Tool
7. Analysis and implementation of insufficient Artificial Intelligence
8. Plug-ins design and implementation of several for proof-of-concept purposes

[1] Picture 1 is scanned from the official Quoridor manual.

## 1.5 Goals

The aim of this thesis is to redesign the original Sector 66 game with all the components introduced in the bachelor thesis, if necessary, to implement sufficient Artificial Intelligence. This Artificial Intelligence should be able to handle unknown plug-ins and games with up to four players.

Namely these points should be met to achieve that:

1. Revise, redesign and reimplement the original Sector 66 Game
2. Revise, redesign and reimplement the original Sector 66 Template Editor
3. Revise, redesign and reimplement the original Sector 66 Service
4. Revise, redesign and reimplement the original Sector 66 Translation Tool
5. Revise, redesign and reimplement the original plug-ins design and implement several for proof-of-concept purposes
6. Analyse, choose a suitable algorithm and implement Artificial Intelligence for the game which would be usable with previously defined goals.

# 2 Artificial intelligence overview

This chapter contains a brief description of the field of Artificial Intelligence. There is also a description of used method and its alternative – Monte Carlo Tree Search and Minimax with alpha-beta pruning.

There are several ways how to approach the field of Artificial Intelligence using computers. There will be some of these methods described in this chapter.

The first complete vision of Artificial Intelligence was published by Alan Turing in "Computing Machinery and Intelligence" in 1950 [2].

Since those days the field of Artificial Intelligence has evolved a lot and now there are two basic approaches to the field. We can either "teach the computer to think" or we can let it generate and test certain situations.

The method mentioned first is inspired by the human brain and nature processes. It is covered by things like artificial neural networks and cellular automata. For the purposes of this work it will be called *cognitive approach*.

The other method is inspired by an ability of all thinking beings to experiment. It is covered by methods like tree searching. In this work it will be called *exploration approach*.

## 2.1 Cognitive approach

Cellular automata and neural nets are exemplary techniques of this approach. Artificial Neural network is a set of neurons connected to some others. Each neuron has multiple inputs but only one output. These neurons pass signals using the connections. When the computation starts there are some signals brought to the input of the network then it processes the signals and after the network stabilizes an output of the network can be read.

This method can be used for example to classify some qualities or to distinguish certain phenomena or behaviour.

## 2.2 Exploration approach

The second approach described is not inspired by brain and thinking, but by behaviour of intelligent beings. When a person wants to learn some new things by

their own, there are two things to do so. The person can explore or exploit already mastered knowledge. This leads to new situations which are then evaluated and we decide whether the new state is good and worth another exploration or not.

These search methods are used often in Artificial Intelligence for computer (board) games.

More formally we are generating situations, which are encoded in states of the game. These states are generated according to the rules of the game. In each state we can try each possible move. Like this we are creating a tree of the game – in mathematical sense. That is a structure of game nodes connected together by edges which represent the moves. It is impossible to check all the nodes of such a tree in most of the games in the time which is available, so we need to decide more intelligently how we will explore the tree and how we will exploit the knowledge already contained in the tree. Some games (like Go) have more valid game states than there are atoms in the whole universe. Exploiting in this sense means to use the data which is already present – to examine promising branches of the tree and not to generate new ones.

There are two basic methods how to go through the tree in a more efficient way. The first is called *Branches and Bounds*. In this method we will cut off the branches which are worse than some others, because if we have a better result in one branch, it makes no sense to go through any branch which we know to be worse. This method will be described in chapter 2.2.4.

The other method would be random sampling of the tree and according to these samples we will evaluate the nodes (representing states of the game) of the tree. One of such sampling method is Monte Carlo Tree Search which will be described later in chapter 2.2.2 but first Monte Carlo methods will be described in general in chapter 2.2.1.

We can also include genetic algorithms to this approach.

*Genetic algorithms* use evolution to find the best possible population. It starts with initialization of the population which consists of specimens. In each specimen there are some values encoded which we want to be as good as possible. What is good and what bad is decided by a fitness function which evaluates each specimen.

After initialization, some specimens are selected as parents then they are recombined and mutated which will create some new specimens. Then from these new specimens

a new generation is raised. Selection, mutation and recombination and selection of a new generation is being done repeatedly till we either have a population which is good enough or till we use up all the time we have at our disposal for the calculation.

### 2.2.1 Monte Carlo

Monte Carlo methods are one of the fortunate outcomes of a very unfortunate event – World War II. But the first note about such a method is much older.

In 1777 a French mathematician Comte de Buffon introduced an experiment. In this experiment there is a needle of length `L` thrown on a board with parallel lines drawn on it in distances of `d`. By repeated throws this experiment is trying to approximate a probability `P` that the needle will fall onto one of the lines on the board. As Comte de Buffon calculated, this equation holds for the probability: $P = \frac{2L}{\pi d}$. [3]

The Monte Carlo method as such was first stated in the 1940's by Stanislaw Marcin Ulam and John von Neumann in Los Alamos National Laboratory where the behaviour of neutrons was studied. This method was then used when the atomic bomb was being developed in the same facility by a team of scientists led by Robert Oppenheimer. [4]

The heart of the Monte Carlo method is to do a large number of simulations or random experiments which can be used to study a solution of a method. Performing such a number of experiments was hardly possible before computers where invented. That is probably the reason why the method, which is considered to be one of the greatest and most important algorithms of the 20$^{th}$ century, was not used before the 1940's [5]. Due to massive computation power of today's computers this method is now used a lot to approximate results of various mathematical problems, such as results of integrals which can be sometimes hard to calculate precisely. The method is used widely since the 1950's.

**Example: Monte Carlo integration in 2D**

Let's have a function $f : \mathbf{R} \to \mathbf{R}$. Our goal is to calculate the definite integral $I = \int_a^b f(x)\,dx; \quad a < b$.

Let's assume, without loss of generality, that $f(x) > 0 \quad \forall x \in \langle a, b \rangle$.

Let's have a constant function $c(x) = v \quad v \in \mathbf{R}$ where $v > f(x) \quad \forall x \in \langle a, b \rangle$

Then $V = \int_a^b c(x)\ dx = (b-a)v$

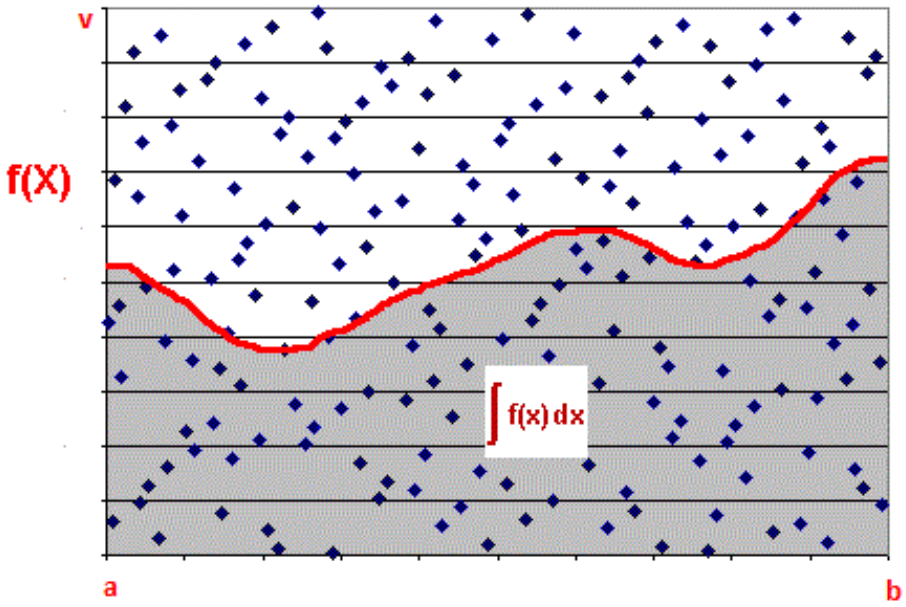
Now we will generate random samples for the Monte Carlo method. The samples will be points in a rectangle given by `V`. So the samples will be a set of points $S = \{(x, y) \in \mathbf{R}^2 \mid x \in \langle a, b \rangle, y \in \langle 0, v \rangle\}$.

From this set we will pick points which are in the area represented by `I`. These points can be distinguished easily, so let's have a set of positive samples $P = \{(x, y) \in S \mid y \leq f(x)\}$.

We can now approximate the value of the integral $I \approx Q_{|S|} = \frac{V}{|S|} \sum_{(x,y) \in S} e(x, y)$ where $e(x, y) = \begin{cases} 1 & y \leq f(x) \\ 0 & \text{otherwise} \end{cases}$. The function `e` is a classifier which returns `1` when the point `(x, y)` is in the region of the integral `I` and `0` otherwise.

For $\lim_{|S| \to \infty} Q_{|S|} = I$. This basically says that if we have enough samples, the approximation will be good enough.

This example is illustrated in Picture 3.




*Picture 3. Monte Carlo integration*

### 2.2.2 Monte Carlo Tree Search

Monte Carlo Tree Search is a method of sampling a tree in Artificial Intelligence. This method is nowadays being successfully used in the game of Go to implement Artificial Intelligence. Go is considered to be one of the most difficult game, it has about $10^{171}$ states [6].

Unlike a common tree search which explores all the nodes in the tree of the game to some depth deterministically, the Monte Carlo Tree Search explores just some branches, but to the leaf of the tree. As leafs are nodes which can be evaluated the easiest, it might be advantages. Leafs in the tree of a game are such nodes where the game is finished, which means that either one of the players won, or the game finished in a draw. These situations can be evaluated trivially.

The result of the search for the best possible move for given player of a game is picked from the set of the descendants of the root of the game tree according to the ration of wins and explorations count. We will either pick the possibility with the best ratio or the node which has been explored the most times. It has been proved that Monte Carlo Tree Search converges to minimax [7].

### 2.2.3 Upper Confidence Bounds applied for Trees

Upper Confidence Bounds applied for Trees (UCT) is an important extension of Monte Carlo Tree Search (MCTS). UCT was formalized by Kocsis and Szepesvári in 2006 and first introduced by Rémi Coulomb [8]. It is used in the most of the cases when Monte Carlo Tree Search is being implemented. UCT was used in the program MoGo, which was introduced as an Artificial Intelligence for the game of Go in 2007.

UCT is basically a Monte Carlo Tree Search used with Upper Confidence Bound (UCB) formula [9]. UCB formula has the following form:

(1) $$v + C \cdot \sqrt{\frac{\log N}{n}}$$

where `v` is an estimated value of the node, `n` is the number of the times the node has been visited and `N` is the total number of times that its parent has been visited. Parameter `C` is a weight constant. The value of `C` controls the balance between

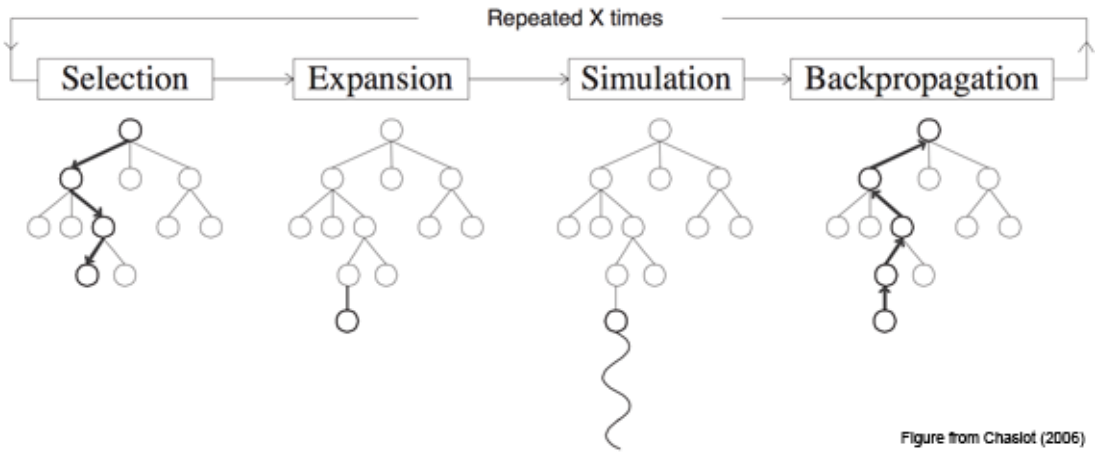
exploration and exploitation. Higher C means wider search – more exploration, less exploitation.

The idea of UCB is that the moves which were more successful than others should be explored more. UCT is keeping this idea. During the calculation it builds a partial tree of the game consisting of the explored states of the game. In each node there is also kept information about how successful the node is and how many times it was explored. The tree is being continuously built during each loop of the algorithm.

The calculation is finished after some amount of time or after exploring a certain amount of nodes. After the exploration is finished the node with the best win rate is picked as the result of the calculation of the Artificial Intelligence.

The build of the tree goes in loop consisting of these steps, which are also shown on Picture 4. Each phase of the algorithm will be described in following part of the thesis.

1. Node selection
2. Expansion
3. Simulation
4. Backpropagation




*Picture 4. MCTS UCT algorithm schema*

**Selection**

In this phase of the algorithm, we are searching for a node which will be expanded during the next phase. This node is being selected using UCB formula (1) described previously in this chapter. As a result of using UCB formula (1) the algorithm is searching subtrees under more successful nodes more than the subtrees under the less

successful nodes. But on the other hand we must sometimes search even the less successful nodes to avoid missing a good move. Thus we must balance exploration and exploitation. This balance is kept by a proper value of constant `C` in UCB formula (1).

**Expansion**

In this phase we will add a new node as a descendant of a node picked in the previous phase of the algorithm. We will randomly, or by a suitable heuristics, choose a move which was not used till now (from the given node) and is possible to be made from the state defined by the picked node.

**Simulation**

After adding a new node we must now evaluate it. That is done by simulating the rest of the game from the newly added node. The game is played to the end in the simulation. The node is evaluated according to who is the winner. If we won, the value is `1`, if the opponent won, the value is `0`. We do not keep any information from this random game for the next phases or repetition of the loop, apart from the value coding the result of the simulation.

In some games the simulation can be endless. In Go this is not possible because after a certain amount of moves there will be no free space to place a new piece. But in some other games, like Quoridor, we can move a figure between two fields of the board for eternity. In such a case some maximum number of moves must be set up. After reaching this limit there are two options. We can either restart the simulation and hope for the best, or we can approximate the value of the final state of the simulation by some heuristics, which is sometimes hard or not precise enough.

**Backpropagation**

In this final step of the algorithm loop we will propagate the result `r` of the simulated game back to all the nodes on the path on the way from the root to the newly added node. We will start with the newly added node in which we initialize the visit counter `n` to `1`, because it has just been searched once and we will also set the value of the node to the result `r` of the simulated game. Win rate of this node will be the ratio of these values:

(2) $$\text{win rate} = \frac{\text{value of the node}}{\text{exploration count}}.$$

After that we will chose as a current node the parent of the newly added node.

This new node has already some value and exploration count from the previous runs. We will add the result of the simulation `r` to the value of the current node and we will increase the exploration count of the node by one. Then we will recalculate the win rate according to the same formula as in the case of the newly added node. After that we will choose as current node the parent of this one. This is being done till we reach the root of the tree.

Backpropagation as described is illustrated by the following Algorithm 1.

```
Node node ← newly added node
node.n ← 1
node.value ← r
node ← node.parent
while node is not null
     node.value ← node.value + r
     node.n ← node.n + 1
     node ← n.parent
```

*Algorithm 1. Backpropagation of a result of a random game in Monte Carlo Tree Search with UCT*

Win rate of each node can be calculated according to the formula as stated before:

(3) $$\text{win rate(node)} = \frac{\text{node.v}}{\text{node.n}}.$$

### 2.2.4 Alpha-Beta pruning

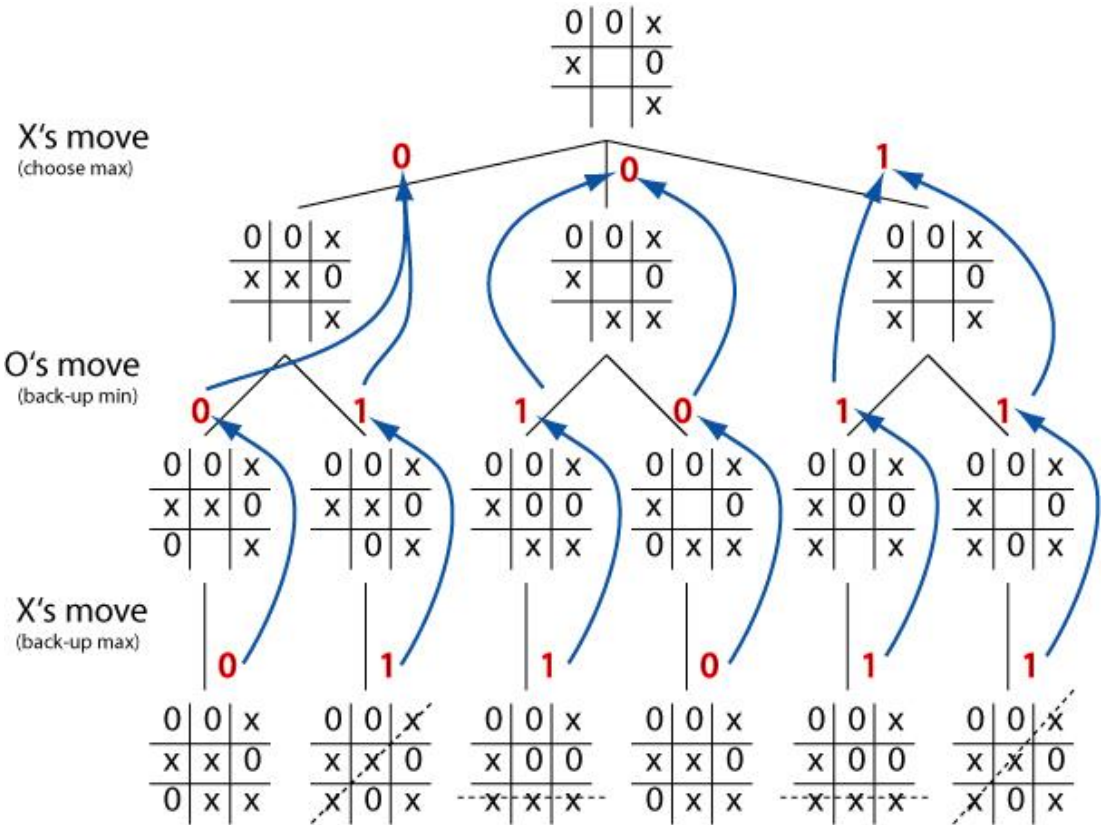
There is minimax with alpha-beta pruning described in this chapter as an alternative way to search a tree in a deterministic way. Monte Carlo Tree Search and Alpha-beta pruning are probably the two most used algorithms to search a game tree in the field of Artificial Intelligence.

Alpha-Beta pruning is an enhancement of minimax algorithm. So we will briefly describe minimax first.

Minimax [10] is a famous algorithm which is used as a base for playing strategical games for two or more players. It was originally introduced for two-player zero-sum

games. The main idea of the algorithm is to explore the game tree of the game and minimize possible loss for a worst case scenario.

The typical use of the algorithm is to find the best possible move from the given game state. That is done by simulating all the possible moves for the subtree and then backpropagating the results to the root of the game. After that we can choose a descendant of the root. At this time all of the nodes are searched and evaluated. The schema of the algorithm is shown in Picture 5. The game of tic-tac-toe is used as a demonstration there. The move evaluated by `1` would be chosen as the next one.

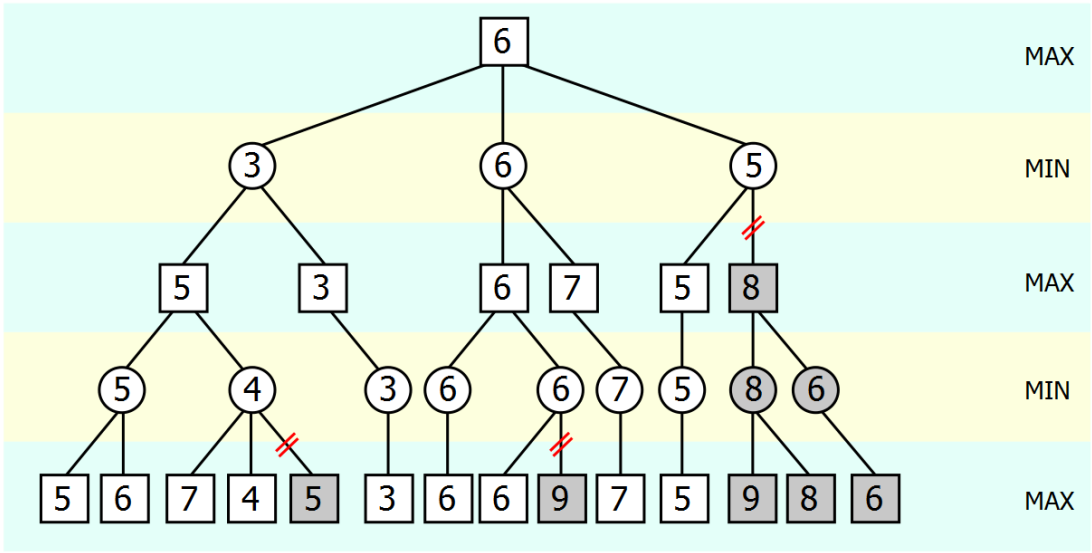



*Picture 5. Minimax algorithm*

Minimax is obviously doing some extra work which might not be done. In some cases it is clear that the move which is being currently explored cannot be better than other which was already explored, so it is pointless to explore the move further.

Alpha-Beta pruning does exactly that. It uses Branch and Bound method to reduce the amount of nodes of the game tree to explore. There are two bounds in this algorithm: `alpha` and `beta` – each for one player. `Alpha` represents the maximum score of the maximizing player and `beta` represents the minimum score of the minimizing player. If a situation where `alpha ≤ beta` occurs during the exploration of the game tree, then the node should not be chosen by the parent node

because it will make the score of the parent node even worse than it already is. Thus the other branches of the node do not have to be searched because it cannot influence the outcome for the subtree. The algorithm is illustrated in Picture 6. In this picture the tree is searched from the left to the right.




*Picture 6. Alpha-Beta pruning*

The efficiency of the algorithm can be further improved by exploring the branches of the tree in appropriate order. The best performance can be achieved when we explore at first those branches, where we expect the best results, but to do that we need to have some heuristics which would approximate the value of the branches.

This algorithm will return the same result as more naive minimax [10].

# 3 Analysis

This chapter analyses how to realise the goals of the work summed up in chapter 1.5. There also are some alternative solutions described. There is one subchapter for each analysed component of the thesis.

The main goal of this thesis is to analyse, design and implement Artificial Intelligence for the game of Sector 66 introduced in bachelor thesis "Sector 66 – A Modular Board Game" [1]. The Artificial Intelligence would also work for Quoridor, because Sector 66 gives the users more freedom. All the rules of Quoridor are also in Sector 66, which adds some extra features. The extra features can be used optionally.

To achieve the goal, some parts of Sector 66 had to be redesigned to allow the Artificial Intelligence to be implemented or to improve the user's experience of the game. These changes are described later in this chapter. Some parts of the original Sector 66 were used without changes or with minor refactoring as it is also stated in this chapter.

Another reason for the redesign is that the design of the original Sector 66 was not perfect. These facts resulted in a need to reimplement a major part of the application because the design used in the bachelor thesis did not allow much changes and especially implementation of Artificial Intelligence. The original design is rather monolithic and thus mostly unusable for the purposes of this work. Mostly the idea of the game and graphics from the original one are used in this thesis.

## 3.1 Game Design and Game Rules

First we will discuss changes in the design of the game and its rules.

In the Sector 66 Game introduced in the bachelor thesis the spells are generated randomly during the game according to the parameters set before the start of the game. That led to too much randomness in a game where no random elements existed originally. Let's imagine a situation where one player is clearly winning, then because of the random spells generation the other player will get a spell which would change the odds dramatically. The first player who did well during the whole game would be rightfully disappointed.

Because of this the rules are changed and the spells are generated only at the beginning of the game, so each player can take the placement of the generated spells in count when developing a strategy.

Before the start of the game there can be specified which spells will be used and how many of them will be generated at the beginning of the game. There may be $k \times n$ spells ($n$ is the number of players in the current game and $k$ is a positive integer), which are generated justly. That in this case means, that if there is a spell $m$ fields away from a player, then there is for every other player the same spell in the same distance $m$.

In the bachelor thesis there was probability and interval of spell generation specified along with allowed spells.

The design of the spells and plug-ins in general was also revised as described later in chapter 3.8.

In the bachelor thesis it was also necessary to finish a move by clicking on a button, which turned out to be unnecessary. In the bachelor thesis this button was disabled until the player finished the move and then the button was enabled. The only thing that the player could do was to click on that button. In this thesis this button is no longer present and the next player is selected automatically. The only situation in which the button might be useful is when there would be a possibility to undo the move and do some other but that also does not make much sense. Usually when the move is done, it cannot be undone in board games.

Both these changes should improve the user's game experience. The first one will lead to certain justice and will level the chances for each player and the other change will remove the possibility of unnecessary clicking.

## 3.2 Common features

In the bachelor thesis a rather monolithic design was used and some of the code was copied between the game itself and the Template Editor, which is a bad practice. It leads to poor maintainability of the code. In this thesis the design is much more modular. There are many common projects which are used by multiple parties. Some code is still necessarily the same in some of the applications. This is on a certain level inevitable, because even if the best possible design is used, there still must be

some objects initialized on the side of the caller, so if multiple callers are using the same object, the initialization will be the same in all of them. The code is in this thesis carefully divided into modules compiled to their own assemblies, so a minimum amount of code is repeated across the modules. These assemblies are then referenced and used by multiple parties.

This design leads to good maintainability, reusability and possible exchange of one module for a newer implementation.

This design will also allow to fully separate the graphical user interface from the business logic of the application. That is necessary for the purposes of the Artificial Intelligence, which is further discussed in chapter 3.9.

Another benefit of this design is that a different graphical user interface could be implemented easily over the core of the game without any changes in the business logic. So for example a lightweight less demanding graphical user interface could be developed to run the game on some devices without DirectX support which is needed by the XNA framework.

The design of the applications is described in chapter 4.

## 3.3 Sector 66 Game

The implementation of the game itself follows the design described in the previous two chapters and realises the introduced changes. To meet the design, the main executable of the application does not contain any business logic. The logic is implemented in common libraries which are referenced.

The graphics of the game and used graphical framework and controls of graphical user interface are kept the same as in the bachelor thesis. It is built on XNA framework with Neoforce controls. This choice made in the bachelor thesis turned out to be a good solution.

Since 2012 when the bachelor thesis was developed there was some progress in the field of 3D hardware accelerated frameworks. For example Unity Framework was introduced. It would serve as a good alternative to XNA, also Mono Game was further improved since those days.

The performance and features of XNA are sufficient for this projects and has no impact on Artificial Intelligence, thus there is no reason to choose another framework.

Localization of the game has been changed a little, as described later in chapter 3.7. Plug-ins design changed a lot, as described in chapter 3.8. Also the way of network communication has been changed, as described in chapter 3.5 and 3.6.

To sum it up, the only change made in the game itself from the point of view of the analysis and implementation is the design, which was changed dramatically. It changed from monolithically, hardly maintainable to modular. Also some of rules of the game were changed a little, as described previously in chapter 3.1, but that is not important from the point of view of software analysis and implementation.

## 3.4 Sector 66 Template Editor

As in the case of the Game described in the previous chapter there are some changes made to follow the described design in order to change from monolithic to more flexible and maintainable modular design.

Apart from design some other changes were made to follow the alterations in the game design. The spells are now generated differently – at the beginning of the game and not during it. Thus the settings of the spell generation had to be changed slightly.

The graphics and graphical framework is the same as in the case of the Game. The same reasons hold here too. Also it would not be a good design if the Game would look differently from the Template Editor. Both are parts of one software so the graphics should be the same to avoid confusing the end user.

Also the same changes of localization were made for the same reasons as in the case of the Game. These changes and reasons are described in chapter 3.7.

## 3.5 Sector 66 Service and network communication

The service for hosting LAN games is a smaller piece of software than the Game or the Template Editor. There is no graphical user interface for example and the only functionality is keeping some information about running games and handling message distribution among the clients. Therefore the redesign is not as significant as it is in other components.

A significant change was made in the communication channel. LAN communication of the bachelor thesis was based on HTTP (hypertext transfer protocol), which is a simplex communication channel used a lot in the Internet. The disadvantage of this solution was that the communication was opened only from the client to the service and the service could not communicate with any of the clients.

In this thesis there is another communication protocol used – TCP (transmission control protocol). TCP channels are by design duplex and thus both client and service can communicate with each other. The communication must still be initialized from the client's side, because there can be some NAT (network address translation) devices in the way so the client might be unreachable from the service when used in the Internet or a complex network.

The advantage of choosing TCP is that the clients do not need to check the updated game state periodically. If a change occurs, one of the players makes a move for example, this change is sent to the service of the game and then the service contacts all the clients connected to the same game and resends them the move of the player or any other change. The periodic checks used in the bachelor thesis created unnecessary overhead. The choice of TCP does not bring any limitation in comparison to the HTTP used in the bachelor thesis.

Both bachelor thesis and this thesis use Windows Communication Foundation (WCF) as a framework for network communication. Using WCF is the best practice when using .NET and network or inter-process communication, so this choice is quite obvious. Implementing low level TCP communication is unnecessary work, which has already been done many times.

Using WCF is not perfect by itself, as described in the next chapter in which the solution for the described issues is also presented.

## 3.6 Windows Communication Foundation Library

Using WCF is a best practice but still the solution provided by WCF is not high level enough for a typical usage. There are still some things which must be handled.

**Exceptions**

Exception distribution from service to client is in WCF possible by using `FaultException`. This method is unfortunately quite clumsy because all the calls

would have to be wrapped by `try-catch` statements. The same is true for handling other exceptions in the communication. For example when the service is unreachable and we will call it, the call would raise an exception, so again the `try-catch` statement would be needed.

**Exception localization**

Another unfortunate thing about `FaultException` is, that if we describe the error state in some text, it is localised according to the language environment of the service. This is impractical if we have clients in multiple culture or language environments. It would be nice to have a way to send just an error code, which would be localized at the client's side according to the environment of the client.

**Reconnection**

Also when the communication is aborted for some reason, we usually want it to be restored automatically without any further programming, which is also not done by WCF automatically.

**Proxies**

Also when communicating over WCF there must be a proxy, or something similar, implemented which looks usually like this:

```
class Proxy: ClientBase<IContract>
{
      public void Method()
      {
            base.Channel.Method();
      }

      public string AnotherMethod(string parameter)
      {
            return base.Channel.AnotherMethod(parameter);
      }

      …
}
```

*Algorithm 2. WCF Proxy*

After writing this, the service method is then called as follows.

```
using (Proxy proxy = new Proxy())
{
      try
      {
            string result = proxy.AnotherMethod(“parameter”);
            //do something with result
      }
      catch (FaultException fe)
      {
            //handle fault exception
      }
      catch (Exception e)
      {
            //handle general exception
      }
}
```

*Algorithm 3. Using WCF proxy*

In the code above `IContract` is the contract interface of the called service.

Writing such a code over and over is again clumsy and unnecessary.

**Channel disposal**

The WCF channels must be disposed properly, which is according to the best practices after each call, meaning that for each call we should create the proxy and then dispose it. This is often omitted by new users of WCF and some serious troubles can rise from that, because each service has a limited amount of channels which can be opened at the same time. If these channels are not disposed properly they are not freed and thus it might not be possible to call the service even if it seems unused.

**Solution**

Because of all these issues, library `Common.WCF` was created during the implementation of the program. How to use it and the design description is in chapter 4.4.12.

`Common.WCF` is basically another layer over the WCF which eases the usage of the framework.

This library allows us to do the same as the code described above (Algorithm 3) with a few lines of code, without implementing the proxy class (Algorithm 2):

```
ServiceResult<string> result = WcfClientProxy<IContract>.Do(a
=> a.AnotherMethod("parameter"));
if (result.IsSuccess)
{
     //do something with result.Result
}
else
{
     //handle error using result.InnerError
}
```

*Algorithm 4. Using Common.WCF.ClientProxies. WcfClientProxy*

The library was designed to be as easy to use as possible and to avoid all the discussed issues of WCF, which is otherwise a very flexible and easy to use solution.

Apart from the described `Common.WCF` also allows automatic reconnection if the channel is aborted for some reason.

This library was not introduced in the bachelor thesis. It is new in this thesis and it is based on years of experience of WCF usage.

## 3.7 Sector 66 Translation Tool

The design of the localization of the Game and the Translation Tool is the least changed part of the bachelor thesis. It is nearly taken as it was only with some minor refactoring.

The localization is realised with XML files with translated phrases. Each phrase is identified by a key. This design is completely the same as in the case of the bachelor thesis. There is one improvement though. The phrases may have a format-string `{0}` in them, which will be replaced with some string like a name of a player or something similar.

In the bachelor thesis this was not allowed. The phrases where a replacement format-string would be needed were split into two phrases and then concatenated according to the following schema: `phrase 1 + inserted word + phrase 2`. This design has a significant weakness. If a language has different word ordering and the

inserted word should be placed somewhere else, this would not be possible. Because of this weakness the format-string `{0}` was introduced in the diploma thesis.

There is no situation where multiple format strings would be needed in one phrase, but if that would be needed after all, the `0` in the format string can be replaced with any other number. So if we would need that, `{0}`, `{1}` and so on can be used to have multiple replacement masks. The introduced design allows that with no necessary changes.

To avoid any mistakes when creating or editing localization, the Translation Tool crosschecks each phrase of the translation with the reference translation for these format-strings. If there is a format-string in the reference translation and not in the translation or the other way round, a warning is displayed, so the user is aware of the possible problem.

There might be a situation where there is a format string in a reference translation and not in some new translation because of the rules of the target language. Thus the mismatch in the format-strings is just a warning and not an error preventing the localization from being saved.

## 3.8 Plug-ins

In this chapter there will be plug-ins discussed. The chapter is divided into parts – General description, requirements and their realisation.

The plug-ins in the bachelor thesis are not powerful enough. They cannot change the game mechanism and they can barely manipulate with any game objects. Plug-ins of spell have only two abilities. There is a way to specify if a spell can be casted on some object and when it is casted, it can access the pawn of the player who casted the spell and the objects, which are the target of the spell, but even these objects can be barely changed.

The situation is even worse with the plug-ins of fields where the plug-in can specify the action which occurs when a pawn steps on the field. The action can change only the pawn of the player who entered the field. Also it can be specified if it is possible to step on the field or not, but only by filling in a property, not by any code which would calculate the value according to some logic.

In the bachelor thesis the plug-ins are rather poorly designed. In this thesis the plug-ins are completely redesigned to allow more things to do. This decision allows the developers of the plug-ins to do a lot of things with the game state, which can even lead to some absurd situations. We assume that if the game was deployed publically, these absurd plug-ins would be refused by the community and thus not used.

There is no way to avoid some absurd situations while giving the developers enough freedom to do what is desired. There are two extremes. We can either give the developers unlimited power or we cannot let them do anything. In the second case, it might be better to not allow plug-ins at all. The optimum is probably somewhere between these two. The design of the plug-ins of this thesis is giving the developers a lot of freedom and at the same time it prevents them from corrupting the basic logic of the game. This is achieved by carefully exposing only some things in the interface of the game state and elements of the game. Thus it is impossible to delete a player for example.

The bachelor thesis was strongly focused on security of the plug-ins. The assemblies were loaded in a separate application domain with carefully limited privileges. So the plug-ins could not access any drive of the computer except reading from the path from which they were loaded and they could only use managed resources of .NET framework. They could not access registry of the system or network.

These features must be unfortunately omitted in this work, because communication between application domains requires Marshalling which is a technique similar to serialization. In the .NET framework Marshalling is used as the only way to communicate across the application domains. Unfortunately the overhead of this method is so significant that it would slow the Artificial Intelligence and move calculation a lot, which reduces the efficiency significantly. The Artificial Intelligence must simulate as many moves per second as possible thus any feature which is reducing the efficiency of the Artificial Intelligence significantly, is not desired. The dynamic nature of the plug-ins is making the situation hard even without further slowdown.

Another reason for omitting the application domain for plug-ins is that some data cannot be marshalled. There would have to be a proxy class just for the transfer and

then the original data type would have to be reconstructed according to the proxy class. This is again consuming unnecessary resources.

There are two kinds of plug-ins in the game. Field plug-ins which can be used as replacements of the default field of the game and spell plug-ins, which represent spells of the game. This is the same as in the bachelor thesis.

In the bachelor thesis the plug-ins are identified by their names. In this thesis the plug-ins are identified by `Guid`. `Guid`[2] is a structure in .NET framework used as a global unique identifier thus it is meant for exactly such a purpose. The name is not unique enough to be an identifier. There is a chance that two developers create two different plug-ins with the same names, but different features. If the plug-ins were identified by names, the game could not distinguish these two. The `Guid` is randomly generated and thus there is little chance that two plug-ins would have exactly the same `Guid`.

### 3.8.1 Requirements

Requirements of the plug-ins on the game elements are described in the following text separately for each game element. Apart from that each plug-in should have a name and a description, so the user would know what it does.

**Requirements on Fields**

1. There should be a way to change the logic controlling the ability of the field to receive a player.
2. There should be a way to change the actions performed after the player steps on a field.
3. There should be a way to store some data in the field.
4. There should be a way to change the type of the field by a spell.
5. Texture and shape of the field should be possible to change.
6. The position of a spell and a pawn on the field should be possible to change.

[2] Guid documentation can be found here: https://msdn.microsoft.com/en-us/library/system.guid%28v=vs.110%29.aspx. How to construct a new one can be found here: https://msdn.microsoft.com/en-us/library/b49320df%28v=vs.110%29.aspx

7. It should be possible to change the logic controlling if a spell can be casted on the field.

**Requirements on Walls**

1. It should be possible to change the state of the wall (to build it, unbuild it and disable it).
2. It should be possible to change the logic controlling if a wall can be changed.
3. It should be possible to change the action performed after built.
4. It should be possible to store some data in the wall.
5. It should be possible to change the logic controlling if a spell can be casted on the wall.

**Requirements on Pawns**

1. It should be possible to change the position of a pawn.
2. It should be possible to change the walls count of the pawn.
3. It should be possible to change the spells of the pawn.
4. It should be possible to change the logic controlling if a player can do a move.
5. It should be possible to change the after move action.
6. It should be possible to store some data in the pawn.
7. It should be possible to change the logic controlling if a spell can be casted on a pawn.

**Requirements on Spells**

1. A spell should have access to full game state.
2. The spell should be able to be cast on field, pawn and wall.
3. It should be possible to change the logic controlling on which game elements the spell can be casted.
4. The player who casted the spell should be distinguishable.

### 3.8.2 Realisation

To meet all the requirements of the plug-ins, game elements and also the game state and architecture of the game had to be changed in this thesis compared to the bachelor thesis.

Each element basically has 3 actions on it: first specifying if a spell can be casted on it, second specifying if a move can be done with it and third specifying the action done after the move. All these can be added and removed dynamically. To do that it might be possible to use events. But using it has two disadvantages.

The event is not identified by anything except the used handler, which is not enough because the developer of the plug-in might not have the handler at disposal at the time. Also the actions, at least the actions done after the move, must be reversible because of the Artificial Intelligence as described in chapter 3.9. If there would be "do action" and "undo action" events, the developers would probably forget to assign the undo action.

Therefore another technique is used to specify the actions. There is a dictionary of pairs (identified, action) and the action has two members "do action" and "undo action", so it is not possible to forget to fill in both "do" and "undo" parts of the action without knowing it. This design is used for all the action which can be specified.

To store some data in the game elements a dictionary of pairs (identified, object) is introduced on each game element. It works in the same way as the actions.

The rest of the requirements are met through the interface of the object. For example making clear who is the player who casted a spell is done by a parameter of casting methods, where there is the pawn of the player filled in. Changing the walls count and the spells is realised by exposing corresponding properties of the corresponding classes.

## 3.9 Artificial Intelligence

This chapter analyses the Quoridor and Sector 66 with respect to Artificial Intelligence. There is a reason for algorithm choice given along with its realisation.

The chapter is divided to following parts: General statements, Algorithm reasoning, its realisation and implementation and finally there are used optimizations described.

Quoridor is according to Table 1 [6] a complex game which can be compared with the game of Chess and other well-known games in terms of complexity.

| Game | State-space | Tree-space | Branching factor |
|---|---|---|---|
| Tic-Tac-Toe | $10^3$ | $10^5$ | 3.7 |
| Checkers | $10^{18}$ | $10^{31}$ | 2.8 |
| Backgammon | $10^{20}$ | $10^{144}$ | 250 |
| Reversi | $10^{28}$ | $10^{58}$ | 10 |
| Quoridor | $10^{42}$ | $10^{162}$ | 60 |
| Chess | $10^{47}$ | $10^{123}$ | 35 |
| Go | $10^{171}$ | $10^{360}$ | 250 |

*Table 1. Game complexity*

According to Table 1 Quoridor has a similar size of state-space as Chess and even a bigger tree-space. The branching factor of Quoridor is more than twice as large as in the case of Chess. Sector 66 is even more complex than Quoridor because there are the spells.

The tree-space of Quoridor is higher than in the case of Chess, because in Chess the figures are in time eliminated. In Quoridor there is no such rule. In Quoridor we can move a pawn between two fields of the board infinitely long. That means that the same states can repeat in the game tree multiple times and thus the game tree can grow a lot. In Chess some figures can be moved between two fields for a long time too, but according to the rules of Chess the other player would probably soon win the game and end it, or some figures would be eliminated and thus the growth of the game tree is limited.

### 3.9.1 Architecture

Because Sector 66 is a very complex game, the Artificial Intelligence for it might need a lot of resources. Therefore the Artificial Intelligence is designed as a stand-alone application which can potentially run on another more powerful computer than the Game. Another reason for this is, that the Game uses XNA framework which can be compiled only as 32bit application which limits the amount of used memory to

about 4GB. There might be a situation in which the limit of 4GB of memory might cause problems.

At this point it is clear that the design of the game which allows splitting the graphical user interface and the core of the game is necessary. The graphical user interface necessarily generates some overhead of used resources which might slow down the calculation of the Artificial Intelligence. Also if the Artificial Intelligence needs the reference to the user interface for its operation, it would generate some requirements on the hardware and runtime environment. Because the user interface uses XNA framework, these requirements would be rather hard. It would for example mean that the Artificial Intelligence could not run on a computer without the support of DirectX, which is not needed for the Artificial Intelligence at all.

### 3.9.2 Algorithm

There are two algorithms for the Artificial Intelligence considered: Monte Carlo Tree Search (MCTS) and Minimax with alpha-beta pruning. Both of these algorithms were introduced in chapter 2.2. Now the reasons for choosing Monte Carlo Tree Search will be discussed.

Unlike other algorithms, MCTS creates the tree asymmetrically. Because of this feature, MCTS works usually better in games with high branching factor. The reason is that classical algorithms are usually using depth search with depth limit to search the game tree. Because of that some branches of the tree might not be searched enough when the time for move calculation is up. In case of Quoridor and Sector 66 which both have high branching factor this is clearly an advantage.

MCTS also does not need any specific evaluation function. The nodes in MCTS are evaluated according to the result of a random game simulation done after a new node is added to the tree. The only thing required is the implementation of game logic, so it is possible to generate new game states according to a current one.

In the case of Quoridor this also might be an advantage. The evaluation function of the game states must be fast. If an evaluation function was used, it would be called many times during the exploration of the tree. If the evaluation function was slow, it would slow down the tree generation and thus the move generation a lot. But at the same time the evaluation function would have to be precise enough to give as good evaluation of game states as possible. If there was a bad evaluation function, which

could not distinguish a good game state from a bad one, the results of the artificial intelligence would be insufficient.

In Quoridor and Sector 66 it is not easy to evaluate game states, as stated in the bachelor thesis. To evaluate a state the distance of a player to the nearest target field is needed. To do that finding the shortest way is necessary and that means, in case of Sector 66, to simulate the moves, because there can be some fields with special features on the way. This simulation can potentially take a noticeable amount of time. Even if there were no special fields used, we must still find the shortest way to the closest target field. We might do that by Breadth-first search (BFS), which takes `O(E)` time, where `E` is the number of edges in a searched graph. The graph in this case consists of the fields of the game and edges represent the possible moves.

Even if we would do that, it does not take a strategical value of the evaluated game state into count.

The algorithm must also be able to handle unknown elements of the game – the plug-ins. The nature of MCTS is random thus if we bring another random element to in, it would not undermine its logic.

Because of all the described issues MCTS seems to be a good choice, because the evaluation function might not be used so much. Still it would have to exist, because in Quoridor and Sector 66 there is no limit in the depth of the tree. The game can be theoretically endless – both players would move between two fields each, or they would move in a bigger loop.

Minimax with alpha-beta pruning is usually used when we have a good evaluation function of game states and when the branching factor of the game is low. None of these recommendations are met in case of Sector 66 and Quoridor.

### 3.9.3 Realisation

According to the assignment of this thesis the Artificial Intelligence should handle more than two players, for which MCTS was originally designed. Also there are the plug-ins which need to be taken into count.

**Plug-ins**

With the plug-ins the solution is easy. Using a spell is just another kind of move, thus there is no need to change the design of the algorithm. The algorithm is just choosing

a move from a bigger set (and thus the branching factor of the tree is higher too). With the field plug-ins it is also easy. These plug-ins also just modify moves which are done when used, so there is still no change of the algorithm.

**Multiple players**

Multiple players are handled the same ways as minimax would do that. The minimax tree is sorted out to levels according to moves. Round is represented by $n$ moves, which corresponds with $n$ levels of the minimax tree ($n$ is players count). So the rounds start at levels $n \times k$ where $k$ is a positive integer or zero. The same idea is used in this thesis to modify MCTS for multiple players. The MCTS tree is designed the same way and the players are taking the moves in the same ways as they would do when minimax was used.

**Evaluation function**

As described earlier in Quoridor and Sector 66 a playout can be possibly infinite. This fact leads to a need of a limited depth of the random playouts of MCTS. There is another reason for the depth limitation of the random playouts.

Quoridor is a complex game and Sector 66 is even worse in this case because of the plug-ins. Usage of a plug-in in the game can be potentially time consuming even in the simulated random playouts, so without the depth limitations, the random playouts could take a considerably large amount of time. Therefore an evaluation function of game state is needed.

This evaluation function should definitely take into count the distance of current player to the closest target fields because getting to one of those is a goal of the game. Also the number of walls which are at the player's disposal should be taken into count, because the walls are strong tactical assets. The same is true for spells. Finally the evaluation function should do the same for the opponents.

To introduce the evaluation function we first need some convention:

| | |
|---|---|
| `w(player)` | walls count held by a player |
| `s(player)` | spells count held by a player |
| `d(player)` | shortest distance to closest target field of a player |
| `v(player)` | value of the game state for a player |
| `W` | wall value |
| `S` | spell value |

Value of a game state for a player `p` is then:

$$v(p) = W \cdot w(p) + S \cdot s(p) - d(p) . \tag{4}$$

So the value of a state for a player increases with the amount of held walls and spells and decreases with decreasing distance to a closest target field.

Weight constants `W` and `S` must be chosen carefully because they are relative to one field of distance to the target. The best seems `W = 0.5` and `S = 0.75`.

`w(p)` and `s(p)` can be evaluated trivially but `d(p)` representing a distance to the closest target field is not that easy and as described earlier it is a potentially expensive operation. To improve the performance of the calculation a heuristic is used in BFS. Examined nodes are taken in an order according to following formula:

$$\text{abs}(\text{target.x} - \text{player.x}) + \text{abs}(\text{target.y} - \text{player.y}) \tag{5}$$

To maintain this order a priority queue instead of a regular is used in the BFS. The implementation of priority queue is based on the heap data structure. The proposed heuristics narrows down the wave of the search toward the target of the player.

Now we have the value of the game state for a specific player. But we still need the value for the current player with respect to the others. If two players are present in the game the situation is easy. The value of a game state for the current player is then

$$v_c = v(p) - v(p_2) \tag{6}$$

where `p` is current player and $p_2$ is the other player.

If there are more players the values must be aggregated. There were two aggregation functions considered: diameter and maximum. In the introduced form of the evaluation function the results when used in the game are about the same.

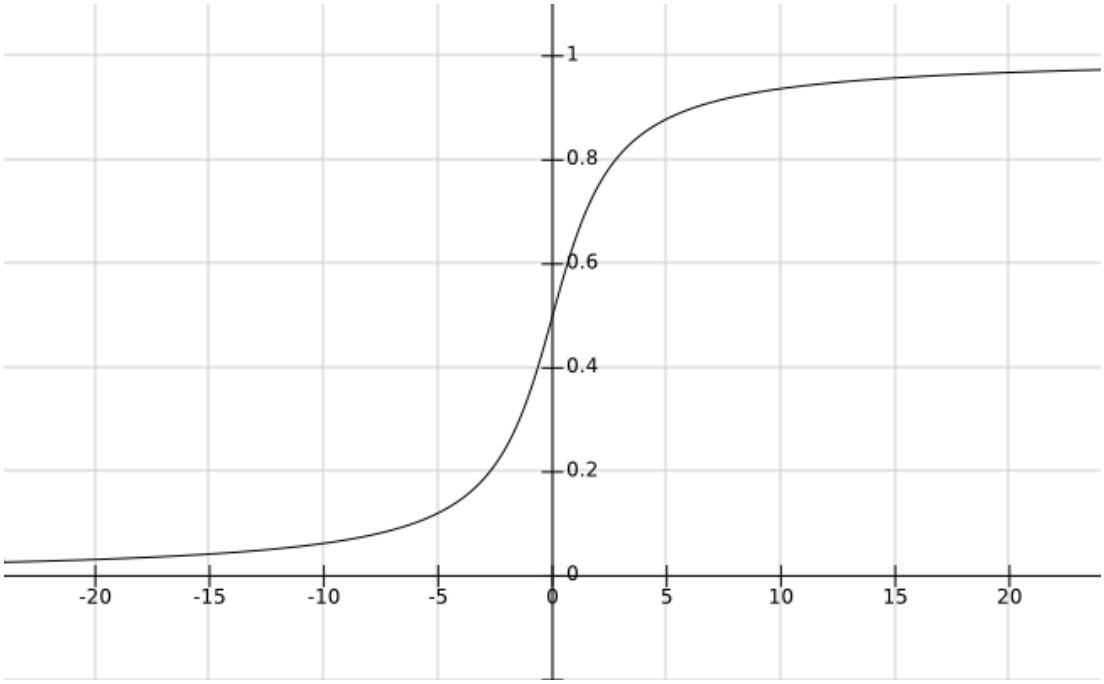
After a minor change in the evaluation function maximum seems to have better results in the game. The changes will be described later in this chapter.

Now we have an evaluation function of the game state but the value still must be transformed a bit because we need the value to be in the interval <0, 1>. 0 if the current player lost the game and 1 if the current player won. Winning and losing must be handled separately in the implementation because the proposed function would take in count also the walls and spells which are not relevant at that point.

There is a commonly used function in games used for these purposes – arcus tangents. Here is the function for calculating the final score from the evaluated game states:

$$score = \frac{\arctan(\alpha \cdot v_c)}{\pi} + 0.5, \tag{7}$$

$\alpha$ is a parameter which regulates how fast the function grows. The graph of the proposed function for $\alpha = 0.5$ is shown in Picture 7.




*Picture 7. Score function plot*

This function is defined in **R** and for our purposes has perfect qualities. From the plot it is clear that the score function (7) can distinguish values between about -15 and 15 well. That is about the same interval as function $v_c$ (5) provides. Also we can see that if the game state is as good for one player as it is for the other, the evaluation function $v_c$ (5) would be zero and then the score provided with this function would

be 0.5. Also small differences of $v_c$ (5) around 0 generate significant differences in the values of score, so we can easily distinguish between two game states.

So the score function described here when put together looks like this:

$$score = \frac{\arctan\left(\alpha\left(\mathrm{v}(p) - \max_{o \in \text{otherplayers}}\left(\mathrm{v}(o)\right)\right)\right)}{\pi} + 0.5 \tag{8}$$

where p is the current player and function v is the evaluation function of a game state for a player p introduced earlier (4).

As it turned out this score functions is quite good but it is not blocking opponent players enough. To do that another parameter V is introduced. It balances the value of the current player's state with respect to others. The final score function used in this thesis it looks like this:

$$score = \frac{\arctan\left(\alpha\left(V \cdot \mathrm{v}(p) - \max_{o \in \text{otherplayers}}\left(\mathrm{v}(o)\right)\right)\right)}{\pi} + 0.5\,. \tag{9}$$

In the implementation `V = 0.9` by default. The introduced score function turned out to be so good, that it is better than using the random playouts before the evaluation of a game state. Thus in the implementation of MCTS for purposes of this work the entire phase of random playouts was omitted. Instead of that the newly added node is immediately evaluated according to this function. Using the score function (9) might be beneficial because there is only a few iterations of MCTS done in the current implementation.

### 3.9.4 Implementation

Now we can finally introduce the exact implementation of MCTS used in this thesis.

There are just three phases of the original MCTS used: Selection, Expansion, Backpropagation. These phases are repeated in a loop. After a certain time elapses or enough nodes are generated one of the descendants of the root of the generated tree is selected as a result of the calculation of Artificial Intelligence. This node is selected according to the win rate (2) introduced in chapter 2.2.3 – The win rate is a ratio of subtree value and count of visits.

Now we will describe each phase of current MCTS implementation.

**Selection**

Selection starts in the root of the tree and is described by the following algorithm.

```
Node Selection(Node root)
     if root.descendants.count = 0
          return root
     else if randomly decide with probability p
          return root
     else
          Node child ← the best descendant of root according
                       to UCB applied on descendant.winRate
          return Selection(child)
```

*Algorithm 5. MCTS Node Selection*

The real implementation must handle some other technicalities. For example in some cases the selection must be repeated because it can return `null` in some cases.

The probability `p = 0.5` by default in the current implementation.

**Expansion**

In the expansion phase we first decide what kind of move will be done: pawn movement, wall build, spell cast. This is done randomly and each kind of move has the probability of ⅓ by default. The target of the move is also chosen randomly from the set given by the rules of the game. After the selection of the move, a new node is created and connected as a descendant of the node selected in the selection phase of MCTS.

**Backpropagation**

First we will evaluate the newly generated move according to the score function (9) presented earlier in chapter 3.9.3.

Then we will use Algorithm 1 introduced in chapter 2.2.3 to backpropagate the score through the tree back to the root.

### 3.9.5 Optimizations

Usually when a game tree is being created, the game state is cloned in every node of the tree. In case of Sector 66 this is very inefficient.

First of all the game states are large because of all the plug-ins used in it and secondly the cloning of the game state is time consuming because of the loading of the plug-ins.

The alternative option to cloning the states is using reversible moves. This option is used in the implementation. The disadvantage is that each move must be reversible, which makes implementation of plug-ins a bit harder. When using the reversible moves there is only one copy of the game state used in the Artificial Intelligence and when moving toward leaves of the tree there are the corresponding moves applied and when moving from the leaves to the root of the tree the moves are undone.

When we are adding a new node, we must choose a move done from the current state. To do that, we need to generate a set of all possible moves and then pick one from it. Because there are plug-ins involved in this phase too and also the rules of Quoridor forces us to do some rather complex checking concerning walls. The moves generation is quite time consuming. The calculation of the possible moves is repeated each time a new descendant is added under a node. To reduce the amount of work to do, there is caching implemented of generated moves in each node.

As another optimization continuous tree computation was considered, but when implemented, the overhead generated by this solution was not worth it. Though there is something similar used. When a move is calculated and reported back to the application and made, the Game will let the Artificial Intelligence know that the move has been made. If the artificial intelligence has considered that move during the build of the tree, the corresponding subtree is selected and the data in it is used in the next move calculation. This is being done with every move, even those which are not calculated by the Artificial Intelligence.

# 4 Development Documentation

This chapter documents all modules of all applications and also demonstrates how the applications and modules communicate with each other. More detailed low level documentation can be found in Attachment 1.

Source codes were developed using C# programming language and can be found in Attachment 2.

User documentation is in chapter 5.

## 4.1 Used software

**Microsoft Visual Studio 2013**

Development IDE

**.NET Framework 4.0**

Environment framework

**Windows Communication Foundation**

Framework for network and inter-process communication

**XNA framework 4.0 Refresh**[3]

3D graphics engine based on DirectX

**Neoforce**[4]

Tom Shane's Graphical user interface for XNA

**NLog**[5]

Logging library distributed as NuGet package

**Enterprise Architect 7.1**

UML editor

**Visual SVN Server**

Source control server

[3] https://mxa.codeplex.com/releases
[4] https://neoforce.codeplex.com/ and https://github.com/NeoforceControls/XNA
[5] https://www.nuget.org/packages/NLog

**Atlassian Jira**

Issue tracker

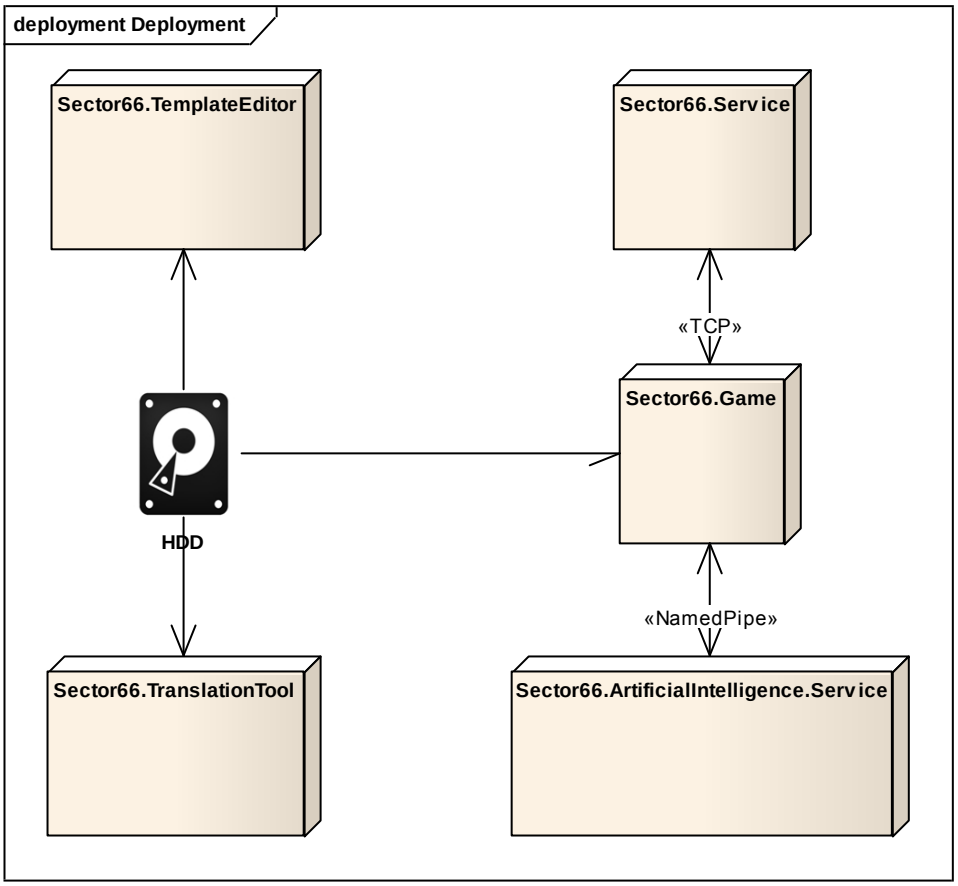
## 4.2 Architecture overview

The project is divided into five stand-alone applications: Sector 66 Game, Sector 66 Service, Sector 66 Template Editor, Sector 66 Artificial Intelligence Service and Sector 66 Translation Tool. Each of the applications and each project will be described later on in this chapter. The applications will be described first and the common libraries later.

The design is as modular and reusable as possible, so the number of the libraries is quite high – there are 12 of them. Also none of the loaders contains any game business logic, which is rather contained in the libraries. The benefit of this design is that the business logic does not rely on the graphical interface at all. As a result developing another graphical interface or any other part of the application, plug-ins especially, is easy. Also it allows a programmer to run the game without the graphical interface at all, which is useful for the purposes of the Artificial Intelligence.

While developing this project, there was a need to create a library which would ease the usage of Windows Communication Foundation (WCF). That library is described in chapter 4.4.12 and analysed in chapter 3.6.

Picture 8 shows how the projects communicate with each other.




*Picture 8. Application communication diagram*

## 4.3 Applications

The results of the build of these projects are executables which serves as loaders of the applications. Despite that they contain a minimum of business logic.

### 4.3.1 Sector66.ArtificialIntelligence.Service

This project contains everything needed for the Artificial Intelligence and move calculation for computer controlled players. It is compiled as a Windows application without a window.

The application is implemented as a stand-alone rather than a part of the game, so it can be run on another computer than the actual game, for example on a very powerful server to give better results.

It communicates with the game using WCF duplex channel. In the default configuration it uses netNamedPipesBinding, but it can be configured by changing the configuration file *Sector66.ArtificialIntelligence.Service.exe.config* to communicate via TCP. To use the TCP channel instead of the named pipe, it is

necessary to change the content of the binding attribute to netTcpBinding of the following line of the configuration file:

```
<endpoint address="" binding="netNamedPipeBinding"
contract="Sector66.ArtificialIntelligence.Contract.IArtificial
IntelligenceService" />
```

It is also necessary to change the base address to meet the netTcpBinding requirements, for example to

`net.tcp://localhost:8734/Sector66ArtificialIntelligence`

instead of

`net.pipe://localhost/Sector66ArtificialIntelligence`

Corresponding changes must be done in the *Sector66.Game.exe.config* file too.

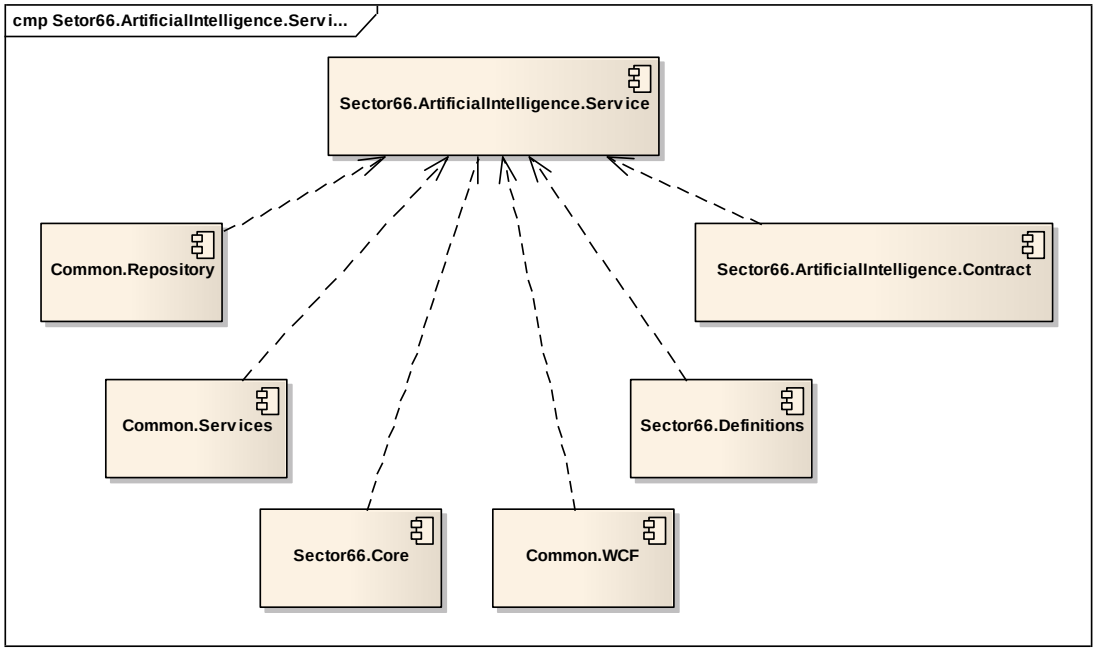
NetNamedPipes is a communication channel used for inter-process communication within one computer and it is duplex. Using one of the duplex channels is necessary to allow the Artificial Intelligence Service to give the results of the calculation back to the main application. It is not possible to return the calculated move immediately as a result of the WCF call because it can potentially take a long time to calculate the move.

The most important class of this project is `GetMoveJob` which implements exploration of the game tree and the selection of the move for the artificial intelligence. To keep the responses of the application as low as possible, the code of this class is executed by another thread, so the main thread of the application would not get stuck during the move calculation.

The tree of the game is maintained in the memory by the instance of class `Node`.

Class `ArtificialIntelligenceWcfService` serves as an entry point of the WCF calls of the application.

Picture 9 shows how this project depends on the rest of them.




*Picture 9. Sector.ArtificialIntelligence.Service dependences*

**Configuration file**

Above there is described how to change the endpoint of the service in the configuration file of the application *Sector66.ArtificialIntelligence.Service.exe.config*.

We can also change parameters which control how the artificial intelligence is calculating the moves. Parameters are in section `applicationSettings` of the configuration file.

In Table 2 there is a description for each parameter, its description and default value.

| **Parameter** | **Default value** | **Description** |
|---|---|---|
| ChanceToAddDescendant | 0,5 | Probability of adding a descendant to currently examined node of the game tree.<br>Values range: 0 to 1 |
| ExplorationConst | 0,1 | Constant which determines the amount of exploration vs. exploitation. Higher value means more exploration<br>Value range: real numbers |

| | | |
|---|---|---|
| CastSpellProbability | 0,3 | Probability of casting a spell when deciding about what kind of move will be done.<br>Value range: 0 to 1 |
| BuilWallProbability | 0,3 | Probability of building a wall when deciding about what kind of move will be done.<br>Value range: 0 to 1 |
| WallValue | 0,5 | Value of a wall which is owned by a player.<br>Value range: real numbers |
| SpellValue | 0,75 | Value of a spell which is owned by a player.<br>Value range: real numbers |
| ScoreConst | 0,5 | Determines the growth of the evaluation function of game nodes.<br>Value range real numbers |
| NodeGenerationCount | 500 | Number of nodes generated when exploring a tree before choosing a move.<br>Value range: positive integers |
| SaveNodes | False | Determines whether the game tree is stored as a XML. If plug-ins altering a player order are used, setting this to true will lead to inconsistent states.<br>Value range: True, False |
| MaxGenerationTime | 30 | Maximum amount of time in seconds used by artificial |

| | | |
|---|---|---|
| | | intelligence to calculate a move.<br>Value range: positive integers |
| CurrentPlayerScoreCoefficient | 0,9 | Weight of score of the player on the move when evaluating a game node.<br>Value range: real numbers |

*Table 2. Sector66.ArtificialIntelligence.Service settings*

### 4.3.2 Sector66.Game

This project contains the high level implementation of the Sector 66 game. It is compiled as a Windows application.

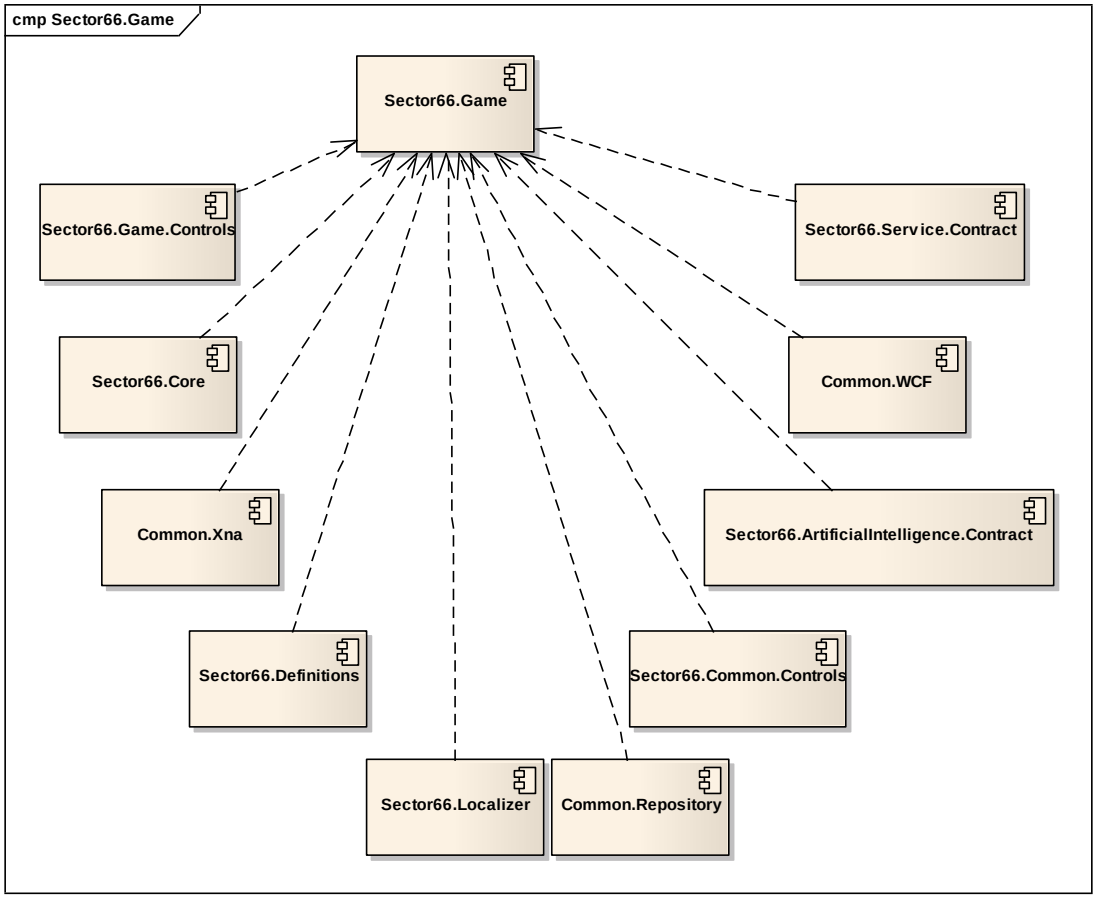
It doesn't contain any business logic nor the controls of the user interface. It just uses some of the libraries and puts their functions together to form the game.

Most of the code is contained in class `Sector66Game`, which basically just creates and initializes the instances of the objects which are controlling the game. It is a base class of the game and it is inherited from `Microsoft.Xna.Framework.Game`.

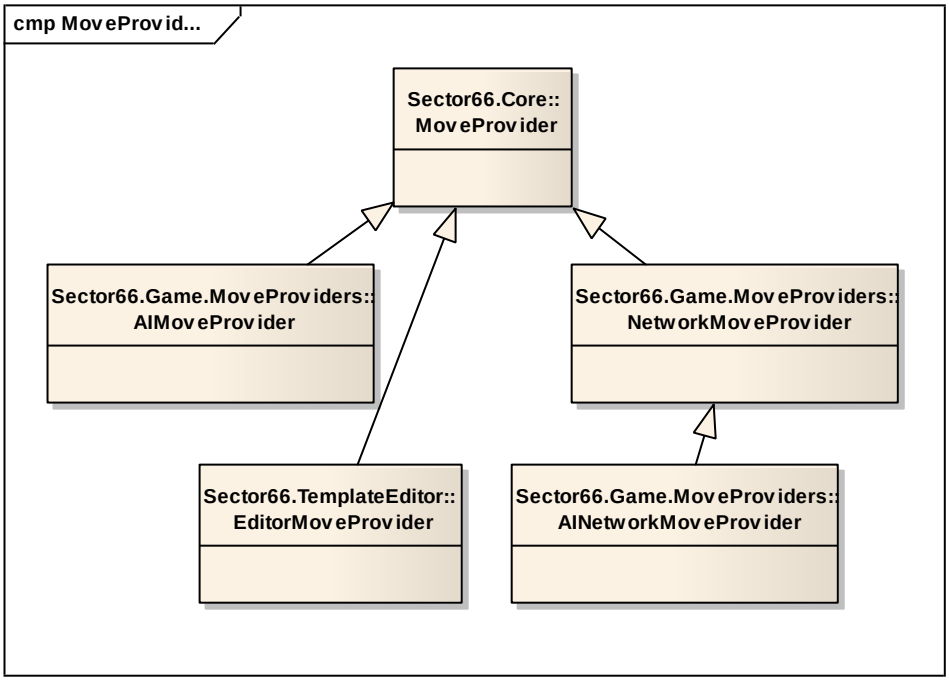
An important part of the project is the namespace `MoveProviders`, which contains classes that provide moves and manage interaction with a user, Sector 66 Service for playing over a LAN and Artificial Intelligence Service.

`AIMoveProvider` is used in local games where artificial intelligence is used. `AINetworkMoveProvider` is used with LAN games with artificial intelligence and `NetworkMoveProvider` is used with LAN games without artificial intelligence. For local games without artificial intelligence the base class of these providers is used. The base class is called `MoveProvider` and is implemented in project `Sector66.Core`.

Picture 10 demonstrates the dependences of this project on the rest and Picture 11 shows how Move providers are inherited.




*Picture 10. Sector66.Game dependences*




*Picture 11. Move Providers heredity*

**Configuration file**

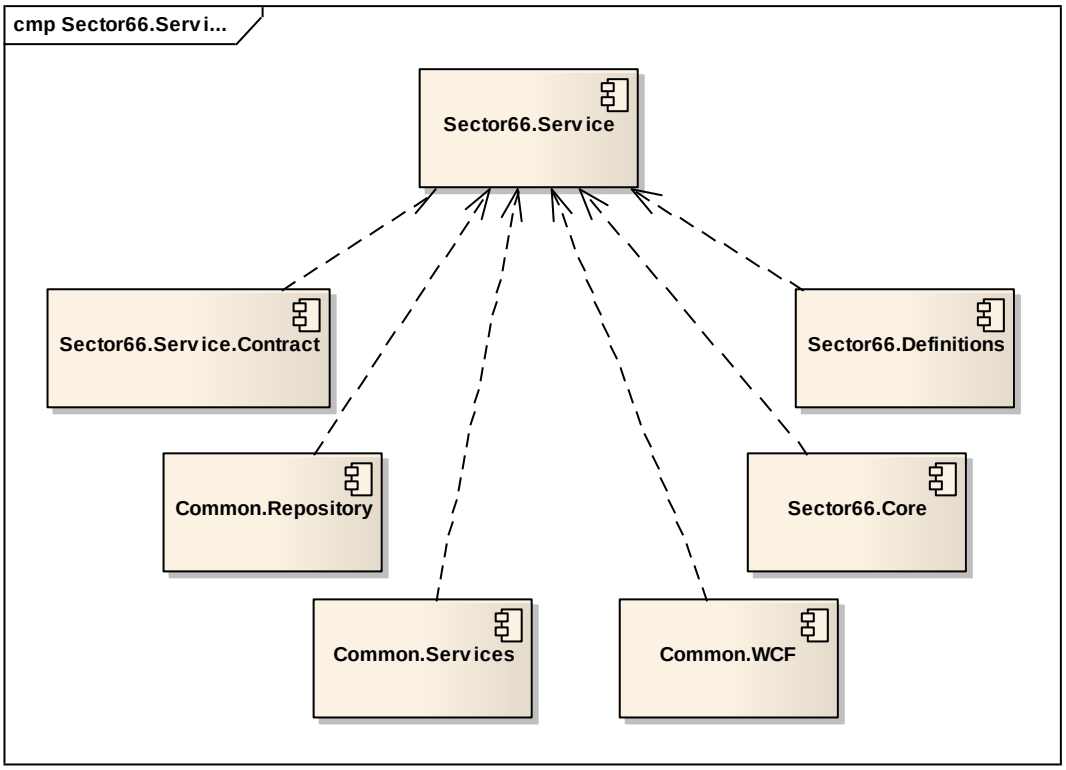
The executable of this project has a configuration file *Sector66.Game.exe.config* which mostly configures WCF communication with *Sector66.Service* and *Sector66.ArtificialIntelligence.Service*. Most importantly, the endpoints of these services are specified there, which means that if we want to configure the game to communicate with a service for hosting LAN games, we need to specify the URL where the service is running in this configuration file. This is being done in section `<client>.` By default it is set to communicate with a service at the same computer:

```
<endpoint address="net.tcp://localhost:8733/Sector66Service"
bindingConfiguration="bindingConfiguration"
binding="netTcpBinding"
contract="Sector66.Service.Contract.ISector66Service" />
```

### 4.3.3 Sector66.Service

*Sector66.Service* project contains logic for managing games played over LAN and handles the distribution of messages among all players. If a player makes an action, it is sent to the service and from the service to all the players including the one who made the move. After reception of the action it is realised at all players, so the received action is handled the same in all cases.

Information about each game is stored in an instance of class the `Game`. All the instances are stored in `GameContainer` which handles the access to the games. Class `Sector66WcfService` is an entry point of WCF calls and also handles sending data to all clients. All WCF projects, including this one, depend strongly on the *Common.WCF* project, which will be described in chapter 4.4.12. All the dependences for this project are shown in Picture 12.




*Picture 12. Sector66.Service dependences*

This project is compiled as a Windows Service, but it can also be run as a regular executable.

### Configuration file

The configuration file of this project is named *Sector66.Service.exe.config*. There is only one thing which might be changed in it and that is the endpoint on which the service runs. It is located in section `services`. If this endpoint is changed the endpoint in *Sector66.Game.exe.config* must be changed accordingly.

There also are two game timeouts in section `applicationSettings` of the configuration file, which are described in Table 3. Both settings are of type `TimeSpan`.

| Parameter | Default value | Description |
|---|---|---|
| KeepWonGames | 00:20:00 | Specifies how long the game is kept in memory after it is won. |
| KeepGames | 01:00:00 | Specifies how long the game is kept in memory when idle. |

*Table 3. Sector66.Service settings.*

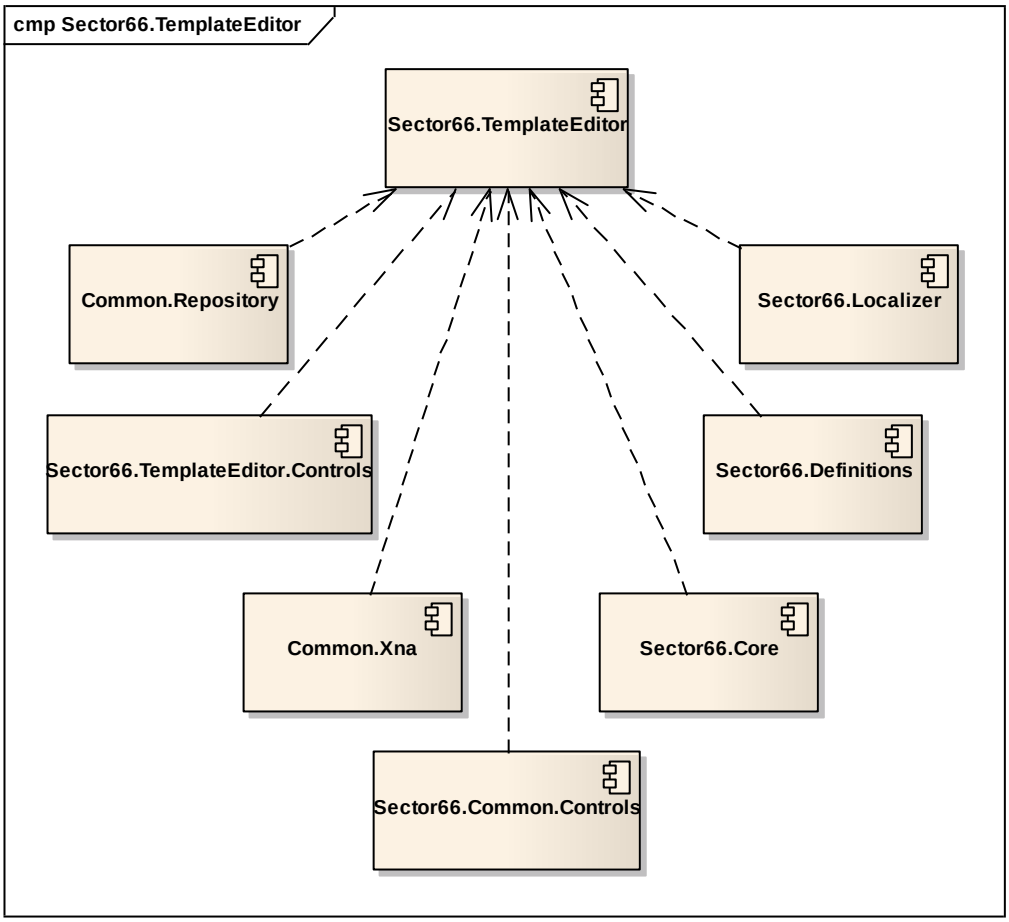
### 4.3.4 Sector66.TemplateEditor

This project is similar to *Sector66.Game*. Both of them use a lot of common projects because they display similar information. *Sector66.Game* gives a user the ability to play the game. *Sector66.TemplateEditor* gives a user an opportunity to create and edit templates of the game. The templates can define how the game will be played, what game mechanisms are allowed and how the initial game board will look like.

Just like *Sector66.Game* it does not contain any business logic and it just initializes instances of some objects which manage the game. Most of the described is implemented in `Sector66TemplateEditor` class.

For the purpose of this project a new move provider is introduced. It is implemented in `EditorMoveProvider` class. Apart from projecting player's "moves", which are the actions that define the look of the game board, it stores this moves, so they can be saved to the game template when the template is being saved.

The project is compiled as a Windows application and its dependences are shown in Picture 13.

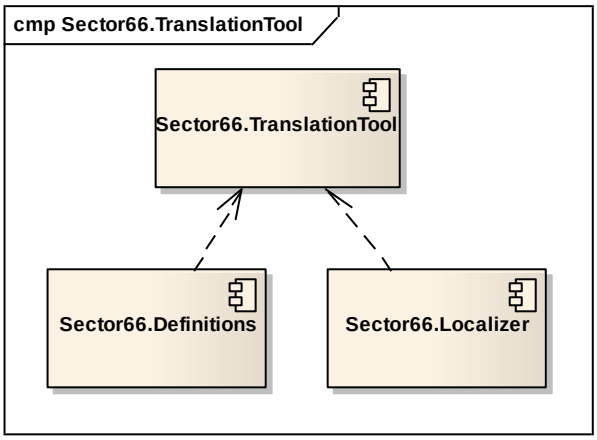



*Picture 13. Sector66.TemplateEditor dependences*

### 4.3.5 Sector66.TranslationTool

This project is compiled as a Windows application. It serves as an easy to use translation tool for creating and editing localizations of Sector 66 Game and Template Editor. It is a very simple application with just one grid and an ability to save and load the localizations. The code handling these functions is implemented in `FormTranslation` class.

Picture 14 shows the dependences of the project.




*Picture 14. Sector66.TranslationTool dependences*

## 4.4 Libraries

Libraries contain the business logic of the applications and also the things which are better to keep separately, such as the graphical user interface. They also contain the code which is common to multiple projects of the applications.

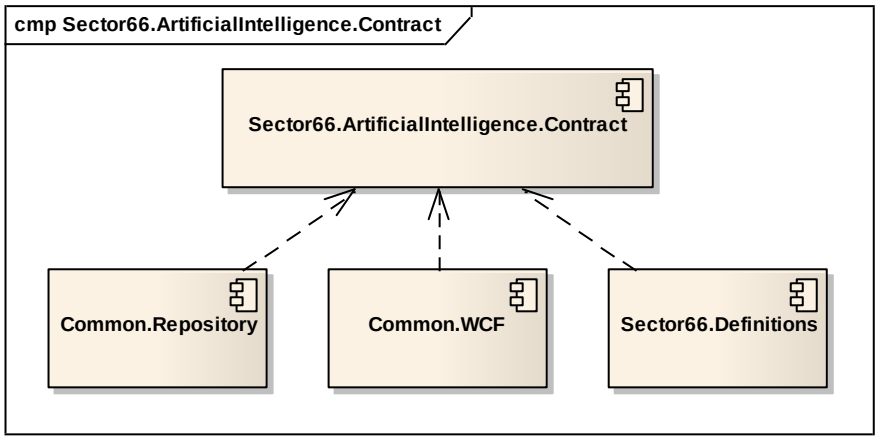
All of the libraries except *Sector66.GameContent* are compiled as Class Libraries.

### 4.4.1 Sector66.ArtificialIntelligence.Contract

All WCF services must have a contract defined. For *Secto66.Service* this purpose is held by *Sector66.ArtificialIntelligence.Contract*.

This project defines two interfaces: `IArtificialIntelligenceService` and `IArtificialIntelligenceServiceCallback`. `IArtificialIntelligenceService` defines the contract used by the client when reaching the service and `IArtificialIntelligenceServiceCallback` for the other direction of the communication. Apart from the interfaces there is just a thin callback proxy class `ClientCallbackProxy`, which implements `IArtificialIntelligenceServiceCallback` and eases the usage of the callback part of the communication for the client.

This project has just two dependences as shown in Picture 15.




*Picture 15. Sector66.ArtificialIntelligence.Contract dependences*

### 4.4.2 Sector66.Common.Controls

This project contains the base classes of controls of the graphical user interface and controls that are common for projects *Sector66.Game* and *Sector66.TemplateEditor* which are the only two applications that use XNA graphical interface. Beside that it contains some support classes and helpers.

There are two base classes defined in the project. `NeoforceBaseComponent2D` is a link between the Neoforce controls and XNA. All 2D components use the common interface `IComponent2D` which is defined in project *Common.Xna* and gives the components the ability to draw themselves and update themselves.

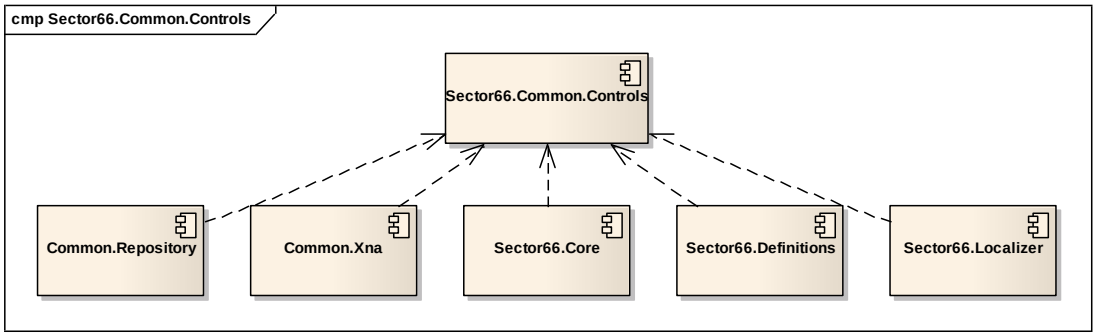
`NeoforceBaseControl` is the base class from which all the user-defined controls are derived from. It adds only one ability: to resize the component to the minimal size in which all the components placed on this one are visible without scrolling.

There are three common controls defined in this project.

`BackgroundComponent2D` implements the background of the Game and the Template Editor and `SettingsMenuControl` implements the settings menu for the applications. `GameComponent3D` is the most important graphical user interface component in the project. It defines the look and implements the functions of the Game including pawns, walls and fields of the board of the game. It basically draws the game state so a user can see it and interact with it. Through this class the user realise their moves, which are then projected back to the game state through the business logic of the game.

Among the helpers there is `TextureCache` class. Textures of the projects are provided to the XNA controls as instances of `Bitmap` class because the game objects must not depend on XNA, so it is needed to convert these bitmaps into XNA `Texture2D` class instances which represent the textures for the purposes of XNA. This conversion is consuming resources quite a lot and slows down the loading of the game components. `TextureCache` is solving this problem. It creates one instance of a texture for each type of a game component and these instances are then reused by all the components which use the same texture. So each texture is converted just once. It saves both time and resources.

The dependences of the project are shown in Picture 16.




*Picture 16. Sector66.Common.Controls dependences*

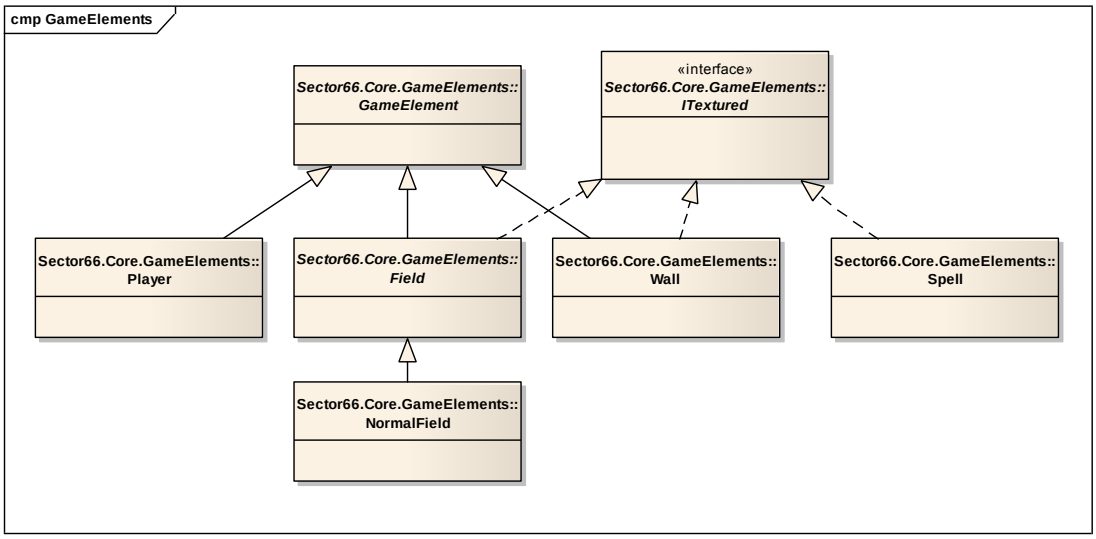
### 4.4.3 Sector66.Core

In a way this is the most important project of all. This is where the game rules, game state and game elements (fields, players and walls) are implemented.

The state of the game is implemented in class `GameState`. An instance of this class holds the placement of the players, states of walls and types and states of the fields of the game board.

`GameRules` is another important class. Implementation of this one tells what game states are acceptable and what moves can be done from the given game state.

All game elements are derived from `GameElement` class, which defines some common things which are useful for plug-ins and spell casting. Classes `Field`, `Pawn` and `Player` are derived from it. Class `Spell`, which is a base class for spell plug-ins is not derived from this class mostly because a spell cannot be casted on another spell. The relationship of the game elements classes is shown in Picture 17.



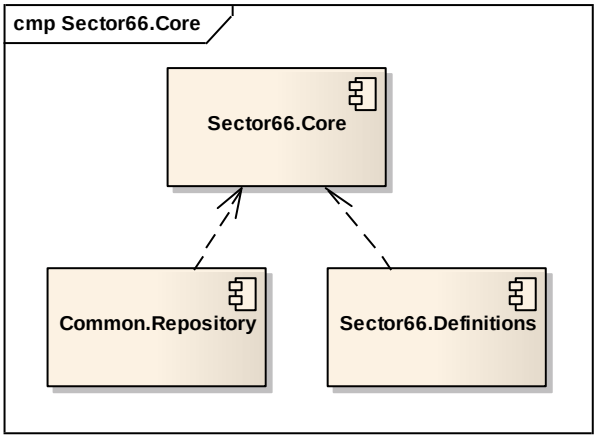

*Picture 17. Game elements relationship*

Another important class is `GameElementsLoader`. This class is responsible for loading plug-ins of spells and field to the game. It must locate the assembly where the plug-in is implemented and load it into memory. It can also read and write the plug-ins' information and assemblies from a drive.

For the evaluation of the game states it is crucial to know the distance of each pawn to the closest target field. This calculation is implemented in `GameStateHelper`. This process is repeated for the purposes of the Artificial Intelligence a lot so it is important that this calculation is as fast as possible. The process of the calculation is described in chapter 3.9.

Also the `MoveProvider` class is implemented in this project. `MoveProvider` is the base class for all the move providers described sooner in chapter 4.3.

The dependences of this project are shown in Picture 18.




*Picture 18. Sector66.Core dependences*

### 4.4.4 Sector66.Definitions

This project contains basic definitions used by nearly all other projects. Among these definitions there are classes like `Move`, which defines an action performed by a player – move of a pawn, spell cast or wall building. It also contains some objects for communication between the game and both of the services like `PluginInfo` and `PluginContainer`. Both of these classes are used to exchange information about plug-ins.

Also there is implemented the class `GameTemplate`. In instances of this class the information about game templates are stored.

Also there is a class `SettingsValues` which makes common settings of the game available.

This project depends just on *Common.Repository*.

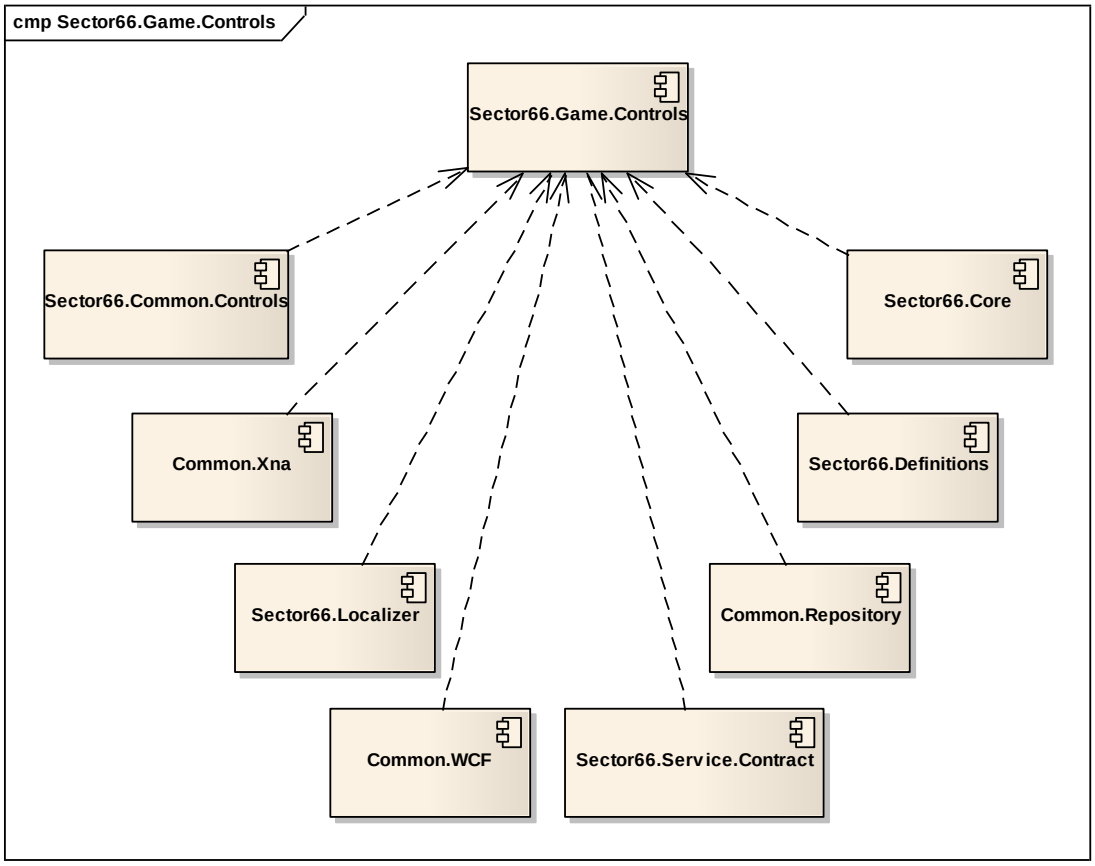
### 4.4.5 Sector66.Game.Controls

Project *Sector66.Game.Controls* contains graphical user interface controls used just by *Sector66.Game*. The most important control `GameComponent3D` which displays the state of the game is implemented in another project because it is used by multiple projects. So for this project we are left only with controls which are used for setting up the game `GameSettingsMenuComponent2D`, the game toolbar `GameToolbarComponent2D` and the game menu `GameMenuComponent2D`. These components are straightforward. There is a lot of code though because the declaration of all the controls which are needed to set up the game takes a lot of writing.

The game toolbar is used to control the game while playing the game. It displays available spells and walls.

The game menu allows the player to exit the game back to the home screen.

Dependences of this project are shown in Picture 19.




*Picture 19. Sector66.Game.Controls*

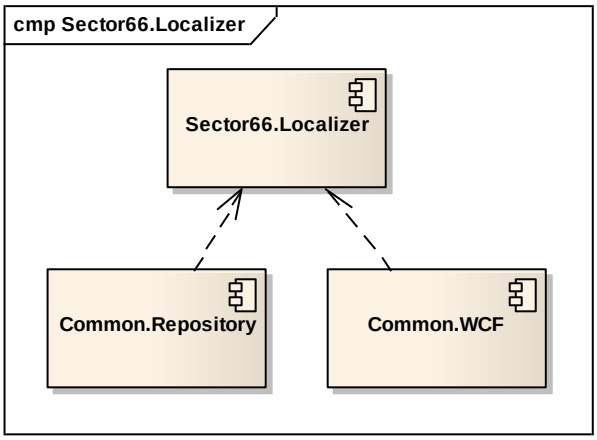
### 4.4.6 Sector66.GameContent

This is a XNA content project where the XNA textures, effect file and mesh models are stored. It does not contain any code. The result of the compilation of this project is files with the extension `xnb`. These files contain XNA compiled data.

### 4.4.7 Sector66.Localizer

This project defines the structure of XML localization files and implements functions for getting and saving data from and to the localization files. The structure of the localization XML files is defined in the classes `Sector66Translation`, which represents the root node of the localization XML, `Translation` and `Expression`. Functions for accessing the localization data is implemented in class `Translator`. There also is the default English localization of the game in this project and extension methods for translating possible error messages provided by the WCF services.

Picture 20 shows the dependences of the project.



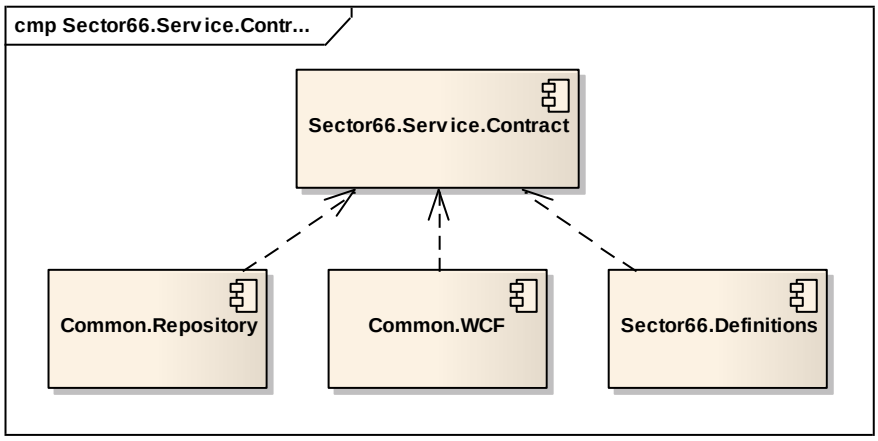

*Picture 20. Sector66.Localizer dependences*

### 4.4.8 Sector66.Service.Contract

This project serves a similar purpose as *Sector66.ArtificialIntelligence.Service.Contrant* but this time it is a contract for *Sector66.Service*. Interface `ISector66Service` defines WCF contract methods used by clients when reaching the service and `ISector66ServiceCallback` defines the contract methods for the other direction of the communication.

There also is `ClientCallbackProxy` class which is used by the client to ease the receiving of messages sent from a service to a client. There also are some definitions of data contract members.

Dependences are shown in Picture 21.



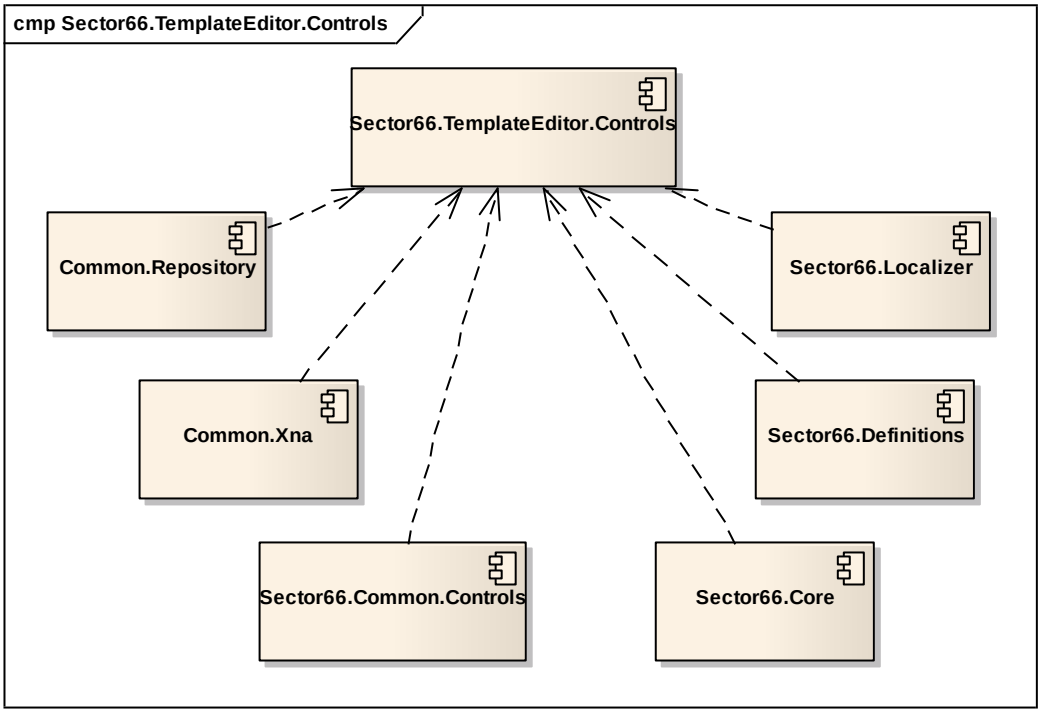

*Picture 21. Sector66.Service.Contract dependences*

### 4.4.9 Sector66.TemplateEditor.Controls

*Sector66.TemplateEditor.Controls* project is similar to *Sector66.Game.Controls*. It implements graphical user interface controls used in *Sector66.TampleteEditor*.

The main class here is `EditorToolbarComponent2D` which is the only high level control of *Sector66.TemplateEditor*.

Dependences for this project are shown in Picture 22.




*Picture 22. Sector66.TemplateEditor.Controls dependences.*

### 4.4.10 Common.Repository

This project contains a repository of some useful tools and data types. Mostly helpers are implemented here – for serialization, for password random generation and for splitting text. There also are generic collection types meant for thread concurrent environments. `ConcurrentList` class uses `Monitor` as synchronization primitive and `ConcurrentRWList` uses `ReaderWriterLockSlim`. These collections are not used in the final version of the thesis.

There also is an implementation of a generic priority queue in class `PriorityQueue.` Implement of this collection is based on the heap data structure.

This project does not depend on any other.

### 4.4.11 Common.Services

This very small project contains just one interface `IConsoleService` used for hosting Windows service classes as a console application, and also a helper which will run them as such: `ServicesHelper`.

The project does not depend on any other.

### 4.4.12 Common.WCF

The content of this project was implemented in a need to have an easy to use library for calling WCF services.

Without this library there would have to be a proxy defined for each service. Also handling possible errors properly and taking care of disposing used channels after each call and reconnecting after possible disconnect is not comfortable. It is also not very easy to let the client know if something wrong happened on the service.

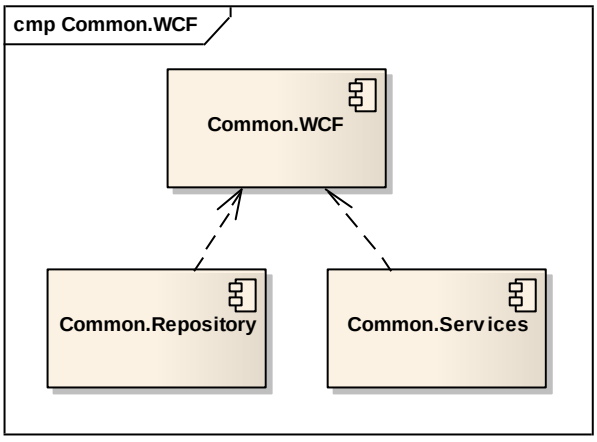
All these things are handled in this project.

Most important parts of this project are client proxies. There are two different kinds meant for direct usage. One for services without callback, which is implemented in `WcfClientProxy` and the other, which is more complex, serves for services with callback – `WcfDuplexProxyManager`.

The rest of the proxies are not meant for the end user, these proxies are used by the two introduced earlier.

There also are mechanisms for easing hosting of the services – `WcfServiceHost` and `WcfHostProxy`. The first is used for direct hosting of a WCF service and the other one is used to host a WCF service as a Windows service.

Dependences of this project are shown in Picture 23.




*Picture 23. Common.WCF dependences*

When we are communicating with a service using simplex channel the call looks like this:

```
ServiceResult<T> result = WcfClientProxy<IContract>.Do(a =>
a.Method(paramaters))
```

*Algorithm 6. WcfClientProxy usage*

In the code abouve `T` is the type of the data of the result of `Method` and `IContract` is the interface representing a contract of the called WCF service.

When using a duplex channel the usage is a bit more complex. First we must initialize `WcfDuplexProxyManager` like this:

```
ClientCallbackProxy callback = new ClientCallbackProxy();
WcfDuplexProxyManager<IContract> clientProxy = new
WcfDuplexProxyManager<IContract>(new
InstanceContext(callback));
```

*Algorithm 7. WcfDuplexProxyManager usage*

In the code above `ClientCallbackProxy` is a proxy of the callback of the WCF duplex channel used to deliver data from the service back to a client and `IContract` is a contract of a WCF service. Calls of the service are then done in the same way as in the case of one-way communication. The initialized `clientProxy` should be disposed when it is no longer needed.

Regardless the communication channel each call returns a `ServiceResult`. If the called service method returns some data of type `T`, the service result will be of type `ServiceResult<T>`. In each service result there is property `Success`, which indicates whether the call ended successfully or not. If it was not successful there is also an error code provided in the property `InnerError`.

The methods of the WCF service should be called using `OperationProxy`, which handles exceptions. If an exception occurs the client will receive a `ServiceResult` filled accordingly. If we want to pass some error code to the client, we should raise `ServiceException` with desired code; otherwise the client will receive just a general error code.

### 4.4.13 Common.Xna

This project contains a set of helpers and base classes for easy usage of XNA framework. There is interface for 2D and 3D components - `IComponent2D` and `IComponent3D`. And also some common 2D and 3D objects.

There is a convertor of `System.Drawing.Color` and `Microsoft.Xna.Framework.Color`. XNA and the rest of .NET framework are using different colour representation. In XNA the colour is represented as a premultiplied vector of floats but in `System.Drawing` it is represented as four bytes (R, G, B, alpha). The conversion between these two representations takes some time because of the necessary multiplication/division. Images are usually quite large, so the conversion takes a noticeable amount of time.

There is also implemented `Painter` class which is responsible for painting all the XNA scenes and objects in the game.

This project does not depend on any other.

## 4.5 Plug-ins

In this chapter we will describe how to develop plug-ins for Sector 66 and also show how those which are already implemented work.

Plug-ins development will be described using Visual Studio 2013 and C#.

### 4.5.1 Plug-in development

To develop a plug-in it is first necessary to create a Visual Studio Class Library project using .NET framework 4.0 and C#. Using C# is not necessary. It is possible to use any other .NET programming language.

If we have the source codes of Sector 66 we must add to the solution in Visual Studio at least projects *Common.Repository* and *Sector66.Definitions*. In plug-in class library we must then reference both of these projects.

If we don't have the source code of Sector 66 it is possible to reference assemblies *Sector66.Core.dll* and *Sector66.Definitions.dll* directly from the installation directory of the game.

All plug-ins must have a parameterless constructor.

Compiled plug-ins are stored in the folders *FieldPlugins* and *SpellPlugins* in the installation path of the game according to the type of the plug-in.

When talking about dimensions width is the dimension along the X axis, depth is the dimension along the Z axis and height is the dimension along the Y axis.

Already developed plug-ins *Sector66.Plugins.Fields.ForbiddenField*, *Sector66.Plugins.Spells.StunSpell* and *Sector66.Plugins.Spells.TeleportSpell* may serve as a hint how to develop a new one.

All the plug-ins should be compiled in *AnyCPU* mode. In each plug-in assembly there should be one plug-in. It is not necessary, but it leads to a better arrangement and plug-ins distribution across the players. If it is clear that some plug-ins must be used together every time, then it would be a good reason to break this rule.

**Field plug-ins**

The field plug-in class must be inherited from class `Sector66.Core.GameElements.Field` and must implement its abstract method `GetTypeId`. The method returns `Guid` which identifies the plug-in in the game. This identifier must be unique for the plug-in. It is unlikely that two randomly generated `Guids` will be the same, so using random is the best practice.

The best way to do it is to decorate the class of the plug-in with `System.Runtime.InteropServices.GuidAttribute` like this:

```
[Guid("random guid")]
public class FieldPlugin : Field
```

*Algorithm 8. Field plug-in class declaration*

In the code about `random guid` is a string generated by Visual Studio Create Guid tool, which is located in Tools menu. The example of the `random guid` is `"8B509F67-93B3-4C81-9EC0-66481278A5FF"`.

Then the content of `GetTypeId` method may look like this:

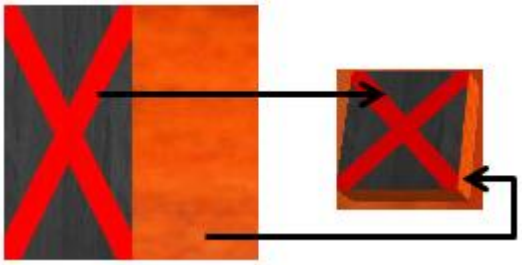
```
return typeof(FieldPlugin).GUID;
```

*Algorithm 9. Recomanded content of GetTypeId method*

In the .NET every type has a unique `Guid`, but by specifying it by using `System.Runtime.InteropServices.GuidAttribute` we are forcing .NET to give the type the same `Guid` every time no matter the changes we might do later or anything else.

In the constructor of the plug-in class we must fill in the properties `Description`, `Name` and `Texture`. `Name` and `Description` are displayed in the application when

using the plug-in to inform users about the plug-in and `Texture` is used as a texture of the field in the game. Due to limitations of the XNA Framework both dimensions must be aligned to the power of 2, but not necessarily the same power of 2. Default mapping of the texture on the field is shown in Picture 24. Mapping can be changed by overriding method `GetVertices`.



*Picture 24. Field texture mapping*

After doing this preparation steps we can now override methods of the `Field` class we want to alter.

By overriding method `CanStepOn` we can change the logic which tells the game if a player can step on the field. When overriding this method we must take care not to alter the game state. Actions in this method will not be undone anywhere. The method is called every time when the game is deciding if a player can step on the field, so it may be called multiple times in a round.

By overriding the method `GetPawnPosition` we can specify where a pawn of a player will stand on the field. The position is relative to the bottom left corner of the field and the centre of the bottom of the pawn. Default implementation will return the centre of the upper side of the field. If you override method `GetVertices`, it may be necessary to override method `GetPawnPosition` too. Otherwise it will probably not be necessary.

`GetSpellPosition` method is similar to `GetPawnPosition` but it does not specify the position of a player, but the position of a spell on the field. The position is relative to the bottom left corner of the field and the bottom left corner of the spell.

Default implementation will return the top left corner of the field plus a little epsilon to be sure that a field will not collide with a spell. It is also not necessary to override this method if `GetVertices` is not overridden.

By overriding method `GetVertices` we can specify the shape, normal vectors and texture mapping of the field. When the shape of the field is changed, it will be

necessary to override methods `GetSpellPosition` and `GetPawnPosition` too. When overriding this method several invariants must be kept.

The width and depth of the field must be equal to `Field.SIZE`. The height can be changed. Default height of a field is `Field.HEIGHT`. Between the fields there is a gap of size `FIELD.GAP_SIZE`.

`GetVertices` method must return an array of vertices. These vertices grouped by three must form triangles (so the count of the vertices must be dividable by 3), which when drawn will be culled[6] counter clockwise, so the ordering of the points within the triangles matters.

Finally we can override method `OnStepOn`. In this method we can program the action which will happen when a player steps on the field. The method is called when a player is moving to the field. Everything we do in this method must be undone by method `UndoOnStepOn`. To undo the actions and for the purpose of the implementation of the plug-in we might need to store some information when doing an action in `OnStepOn` method. For this purpose the `Field` and `Move` has property `Values`, where we can store anything we want. It is a dictionary of pairs (`Guid`, `object`) so each entry of `Values` property is identified by a unique `Guid` which we must provide when storing or accessing any object into it. If only one object is stored, it is recommended to use the result of `GetTypeId` method.

The same instance of `MoveBase` class will be handed over to method `UndoOnStepOn` when undoing the move.

After compiling the plug-in we will place it in the *FieldPlugins* folder in the installation folder of the game. After starting the game plug-in will be loaded from it automatically.

**Spell plug-ins**

When developing a spell plug-ins we prepare a project in the same way as we would do in case of a field plug-in.

[6] Culling is a technique used in 3D graphics to draw only the faces which are facing the camera. Those which are not facing it will not be drawn. Counter clockwise culling means that only faces which have vertices in counter clockwise order will be drawn.

Spell plug-in must be inherited from class `Sector66.Core.GameElements.Spell`. Abstract method `GetTypeId` must be implemented in the same way as described above in the case of a field plug-in.

In the constructor of the spell plug-in class we must fill in properties `Description`, `Name` and `Texture`. These properties have the same purpose as in the case of field plug-in. The mapping of the texture is simple in this case. By default spell is a square where the texture is mapped as it is in `Texture` property. Again the mapping can be overridden in method `GetVertices`.

Now we can override methods of `Spell` class to achieve desired behaviour.

There are three overloads of method `CanCastOn` according to the type of the game object (Field, Player and Wall). Default implementation of these methods return `false` indicating that the spell cannot be casted on any game object. According to what game object we want to cast the spell on we choose the correct overload of the method and then we will override it. The correct overload is recognizable according to the second parameter. These methods work similar to the `CanStepOn` method of `Field` class. These methods must not change the state of the game because the actions done here are not undone anywhere.

There are also three overloads of method `CastOn` again according to the game object and again the overload is recognizable by the second parameter. In these methods we specify what the spell does. When we override the `CastOn` method, we must also override the corresponding `UndoCastOn` method to undo the effect of the spell we are developing. All the action must be reversible. Again we can use `Values` property on parameter `move` as we could when implementing a field plug-in. `Spell` does not have the `Value` property because it makes no sense to add data on runtime. The spell cannot be casted on another spell, so there is no game mechanism which would allow altering a spell on runtime.

Finally there is method `GetVertices` which we can override. It works similar to the method of the same name of `Field` class. Default shape of the spell is a square with the size of `Field.SIZE - 2 × Field.SPELLRELATIVESIZE` the spell has 0 height. By overriding this method we can easily create a spell with 3D icon.

### 4.5.2 Sector66.Plugins.Fields.ForbiddenField

This field plug-in represents a spell on which a player cannot step on. To achieve this behaviour method `CanStepOn` is overridden to return false every time.

### 4.5.3 Sector66.Plugins.Spells.StunSpell

This spell plug-in stuns a player, who is a target of the spell, for one round, so the player will not be able to move for the next time. This is achieved by overriding `CanCastOn`, `CastOn` and `UndoCastOn` methods overloads for the situation when a player is the target of the spell.

In `CanCastOn` method we specify that the spell can be casted on another player than is the one who is casting the spell and also the target of the spell cannot be already stunned.

In `CastOn` method we add an action to the target player `CanMoveActions` and we also insert a marker into the target player's `Values` collection to know, that the player is stunned. The newly added can-move-action returns `false` if the marker is present in a player's `Values` collection, which will prevent the player from making a move of any kind.

`UndoCastOn` is implemented to undo the addition of the marker into the `Values` collection.

### 4.5.4 Sector66.Plugins.Spells.TeleportSpell

This spell plug-in teleports the player who is casting the spell to another neighbouring game field. The advantage of this spell is that the player can jump over a wall.

To do that `CanCastOn`, `CastOn` and `UndoCastOn` methods overloads for `Field` are overridden.

In `CanCastOn` method the distance of the targeted field is checked. Also we must check if a player can step on the field and if it is not occupied.

In `CastOn` method we mark the casting player's current position to `Values` collection on move parameter to be able to undo the move. And of course we set up

the player's position to match the target field position. Finally we must mark the player to be redrawn and do step on actions on the target field.

In `UndoCastOn` method we undo all the actions described in `CastOn` method.

# 5 User Documentation

This chapter describes how Sector 66 works from the point of view of a user.

The following description is for the game installed with the installer in Attachment 3 and English localization.

Configuration files of each executable are described in chapter 4.3.

## 5.1 System Requirements

Processer: At least Intel i3 or higher

RAM: 2GB or more

Resolution: 1366 × 768 or higher

Inputs: Keyboard and mouse

XNA Rich mode compatible graphical card (nearly any graphical card will do)

Hard disk space: 10MB

Windows XP or newer with .NET framework 4 (tested on Windows 7)

## 5.2 Installation

There are two ways how to install the game.

There is an installer prepared in Attachment 3 which will install everything the game requires including XNA framework redistributables. Optionally Windows service for hosting LAN games *Sector 66 Service* will be installed and turned on too. The installer must be run with administrator privileges to have the permission to install a Windows service. The installer will add shortcuts to Start menu in folder *Sector 66* to all game components.

Alternatively it is possible to install just XNA redistributables from Attachment 4 and then unpack a zip file in Attachment 5 which contains all the data which would be in the installation folder after the installation as described above. If this option is used then user is responsible for adding all the shortcuts by himself/herself if needed.

Installing the Windows service for hosting LAN games can be done by this command in the Windows command line where `path` is the full path to the directory where the game files are:

```
sc create “Sector66.Service” binPath=
“path\Sector66.Service.exe”
```

To remove the installed Windows service we run this command:

```
sc delete “Sector66.Service”
```

All the components of the application must be run from a location where the game can write.

## 5.3 Sector 66

Sector 66 is the main part of the installation – it is the game itself. It can be run either by using the shortcut *Sector 66* in Start menu or by *Sector66.Game.exe* from the installation folder.

The window of the application is divided by tabs to three pages: Game on Local Computer, Game on LAN and Settings.

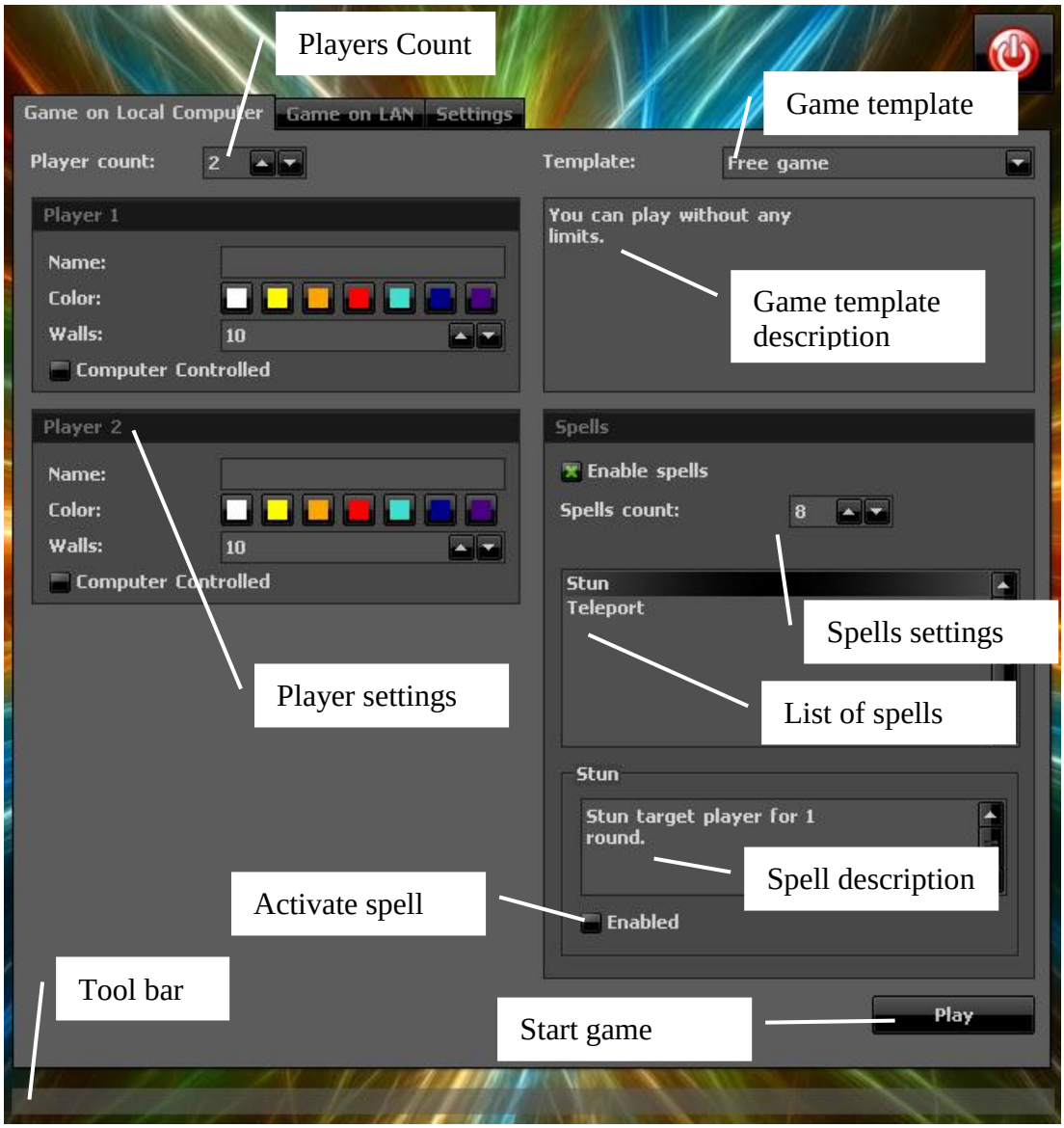
Over these tab pages there is an exit button and under them there is a status bar which will show important notes and error states.

Each of the tab pages is described in the following text.

### 5.3.1 Game on Local Computer

Controls on tab page *Game on Local Computer* serves as inputs to specify the settings for a game played on local computer. Some of the players may be operated by artificial intelligence.

Picture 25 shows how the described tab page looks.




*Picture 25. Game on Local Computer tab page*

In top left hand corner we can specify how many players will be in the current game. There can be from two to four players according to the selected game template.

The template, which will define a basic character of the game, can be chosen in the top right hand corner. After selecting one, a description of the game template will be written in the designated text area. Some templates can disable certain settings of the game. For example a template can force some spells to be used in the game. The templates which are loaded into the menu must be stored in the *Templates* folder of the installation folder of the game. When creating a template, this is where it will be saved automatically.

Under the players count selection control there is one player settings panel for each player where we must specify their names and colours. We can also specify how

many walls each player will have. Maximum and minimum number of walls is limited by the selected template and players count.

The last thing we can change is whether a player is in control or the Artificial Intelligence is.

On the right side of the screen we can change how the spells will be used in the game. The inputted number of spells will be generated at the start of the game. We can choose which spells will be used in the game, or we can decide to disable the spells at all.

The game will start after filling in all data and clicking *Play*.

### 5.3.2 Game on LAN

This chapter describes how to start a game played over the LAN. The service which is used to host the game is specified in the configuration file of the executable. This configuration is described in chapter 4.3.2.

Described controls are located on *Game on LAN* tab page of the game menu. We can either create a new game or connect to an existing one.

**Starting a new game**

When creating a game, one must specify the game name, game template and maximum number of players. Game password and description of the game can be left blank. If we fill in the password, only those who know it can connect to the game.

To create a new game we will click on the button located in the bottom right corner. After successful creation of the game, the control where we can specify players and spells settings will be shown. The controls are similar to the ones used to start a local game. The only two differences are that the game can be started after each player confirms the settings by clicking on the green tick located in the bottom right corner of the player settings control. The other difference is that there is a control to add a new player in the top of the screen.

The game will be started after setting up all the necessary inputs, confirming the player settings and clicking on the *Play* button.

**Connecting to an existing game**

There is also another tab page for connecting to an existing game.

There is a list of games on the server on the left hand side. Above this list there is a button which refreshes that list.

After selecting a game in the described list, description and some other information of the game will be displayed on the right hand side of the screen.

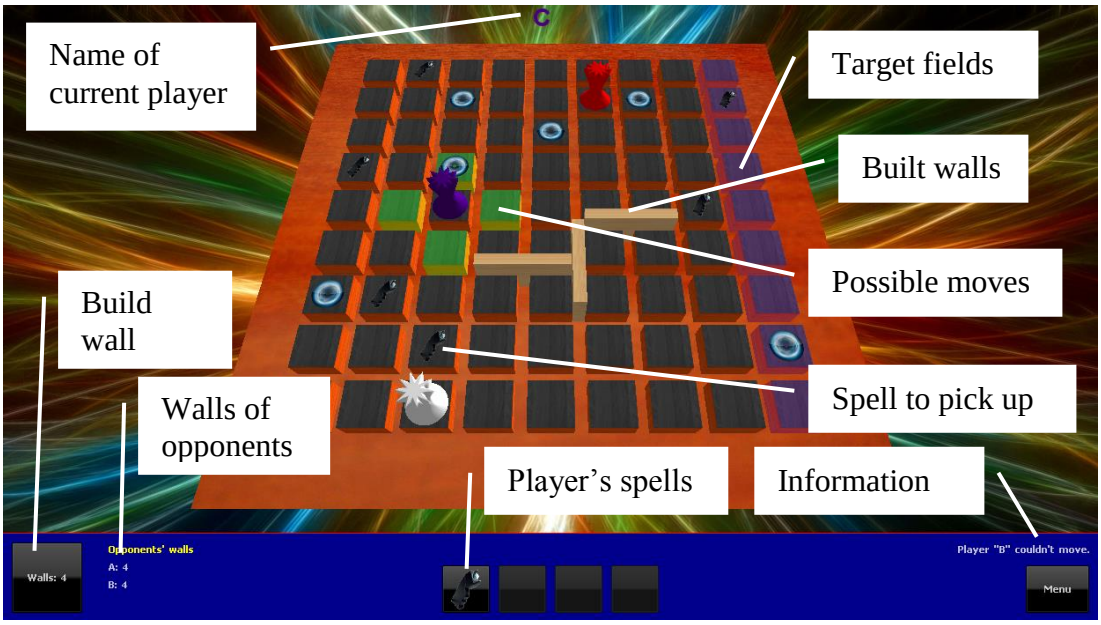
It is mandatory to fill in the name of the player who is connecting to the game. According to settings of the game it might be needed to input a password of the game.

After filling in all the necessary information and clicking on the connection button, another screen will be displayed to setup players.

This screen is similar to the one described in chapter 5.3.1 for local games but in this case only the colour of local players can be changed. All the other things are set up by the one who created the game. To start the game all the players must confirm their settings by clicking on the green tick in the bottom right corner of the player settings control. The game will be started by the player who created the game.

**Playing a game**

After starting a game (no matter if a network game or a game on local computer) a window showing the board of the game will be displayed. The look of the window is shown in Picture 26.




*Picture 26. Game window*

There is the name of the player who is currently on the move in the top of the window. Fields where the player can make a move are highlighted and also the target fields are highlighted with the colour of the player. When a player reaches a target field the game will end and the one who stands on the corresponding target field first will win the game.

In the bottom of the window there is a tool bar. In the tool bar there is a button to build a wall, information about the number of walls of the opponents and buttons with spells which are available to the current player. On the right there is a button which will show a menu of the game.

By clicking on the wall button the mode of the move is changed. By moving a mouse over the board of the game one can select a wall to build. The currently picked wall is highlighted and by clicking it will be selected to build.

Spells are working in a similar way. When a spell is selected, the game elements on which the spell can be casted are highlighted. A target of the spell is selected by clicking on it.

To cancel or change the mode of the move we can click on another spell or wall button or click on the same one again.

After finishing the move, the next player will be selected automatically.

All the buttons have tooltips with more information about them. These tooltips are displayed when the mouse is hovered over them.

### 5.3.3 Settings

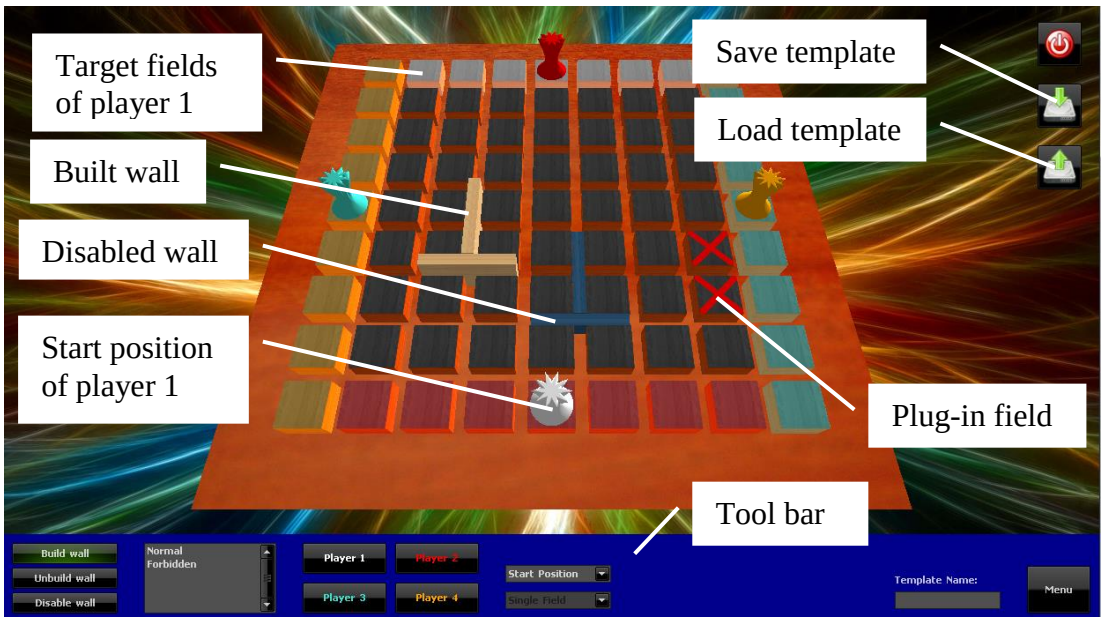
In this tab page localization can be set up. The change of the localization will be fully effective after restarting the application.

## 5.4 Sector 66 Template Editor

Sector 66 Template Editor is a tool to edit game templates. It can be started by using *Sector 66 TemplateEditor* shortcut in start menu or by *Sector66.TemplateEditor.exe* from the installation folder of the game.

The templates are used to state basic rules of the game. We can set up spells, target and start fields of each player, we can limit players count in the game and we can also change fields of the game and build or disable some walls.

The window of the editor looks similar to the game window. It is shown in Picture 27.




*Picture 27. Template Editor window*

Most of the window is taken by the board of the game where the positions of the players, their target positions and other game elements positions are shown.

In the bottom of the window there is a tool bar where we can change the modes of the placement of the elements. On the left there are wall modes, then there are field

plug-ins and then player position modes. On the right side there is an edit box for entering the name of the template and a button which opens the menu of the editor.

In the top left corner there are buttons to exit the editor, save a template and load template.

Wall modes allow building, disabling or unbuilding a wall.

Field plug-ins mode allows replacing a field of the game with another one.

When changing a player's positions we first must select a player by clicking one of buttons *Player 1* to *Player 4*. Then we must select the type of the position. There are two options for this. We can either change the position where a player starts or the position where a player's target positions are. As a target position we can choose column, row or a single field.

The menu of the editor is divided to three tab pages. In the first one we can set the count of the players and wall count limits for each situation given by the number of players present in the game. We also must fill in the description of the template, which is together with the name of the template mandatory.

Second tab page allows us to set up spells. We can specify how many spells can be generated in the game, if spells will be allowed at all and we can also force some spells to be used.

The last tab page contains localization settings. As in the case of the game, if we change localization it will be fully loaded after restarting the application.

After saving the template it will be stored to the *Templates* folder in the installation directory of the game.

## 5.5 Sector 66 Service

Sector 66 Service is an application which is hosting LAN games. It can be run either as a Console application or as a Windows service. But it can't be run in two instances on one computer without changing the TCP port in the configuration file of one of the instances of the application. If we start the second instance on the same port it crashes and writes to log that it couldn't open a port which is already taken.

Configuration of the service is described in chapter 4.3.3.

The service can be run either from a windows services console, which can be opened by typing `services.msc` to Windows run dialog or by Start menu shortcut *Sector 66 Service* or directly by *Sector66.Service.exe* from installation directory of the game. If the service is installed as a Windows service by the supplied installer, it is started automatically and set to auto start, so it will start automatically after each boot of Windows.

## 5.6 Sector 66 Artificial Intelligence Service

This application is the artificial intelligence of the game. We can run it manually by Artificial Intelligence Service shortcut in Start menu or directly by *Sector66.ArtificialIntelligence.Service.exe* but it is not necessary. This application will be started and exited automatically by the game if it is needed. When it is running the icon can be found in the system tray. When clicking on it with right mouse button we can open a context menu of this service and exit it from there. If the application is exited while playing a game with artificial intelligence it will not be possible to finish the game. Exiting the application safely is possible only when the game is not running, but in this case the service should be exited automatically.

This application is localized by satellite assembly with the localization. The localization is chosen automatically according to the localization settings of Windows. There are two languages implemented: English and Czech.

## 5.7 Sector 66 Translation Tool

Translation tool is an application for creating and editing the localization of Sector 66 game. It can be run by using Start menu shortcut *Sector 66 TranslationTool* or directly by *Sector66.TranslationTool.exe* from the installation folder of the game.

This application is localized in the same way as Artificial Intelligence Service. It uses satellite assembly with localization which is picked according to the localization settings of Windows. There are two languages implemented: English and Czech.

Most of the window is occupied by a grid with the translations. There are three columns: the key, reference translation and translation. Only the last column is editable. The translation can be loaded and saved from menu *Translation*. Reference translation can be loaded from menu *Reference Translation*.

Some localization text has a string `{0}` inside. That is a mark which will be replaced by some other text. Usually it is a name of a player or something similar. When this string is in a reference translation, it should also be in the translation. Otherwise the text displayed then in the game might not make sense, but the application will not crash when it is not present.

Before saving the localization the header, which is located in the top of the window must be filled in. It is recommended to name the localization files by the name of the language, which should also be filled in the same way in the header.

The localization files of Sector 66 must be saved in folder *Translations* in the installation folder of the game; otherwise the game will not load them.

# 6 Conclusion

The goal of the thesis was to design and implement Artificial Intelligence for Quoridor/Sector 66 Game which would be usable with two to four players. According to the assignment of the thesis the Artificial Intelligence should also handle plug-ins, which are bringing something completely unknown and fully customizable to the Game.

To do that it was necessary to redesign the original game of Sector 66 proposed in the bachelor thesis "Sector 66 – A Modular Board Game" [1], which was used as a base of this thesis.

The Artificial Intelligence proposed in this thesis is not excellent but when the difficulty described in chapter 3.9 is taken into count the outcome seems good. When there are two players in the Game the Artificial Intelligence might be quite a challenge for an average player of Quoridor.

When there are more players in the game, it is hard to design an Artificial Intelligence in general. Alliances and short time truce may be formed between some players. The alliances are shifting the balance of the game a lot. It is also hard to decide which player to block and whom to support in what phase of the game. Though, the proposed Artificial Intelligence is of some use in this case too.

The game of Quoridor is still far from being solved. Compared to Chess and other well-known games, Quoridor is still not analysed well. If there was more theoretical knowledge about the game, the Artificial Intelligence might perform much better. The Artificial Intelligence designed and implemented in this thesis does not use any strategy, as it often is with the game of Chess and thus it can be further improved after some theoretical analysis of Quoridor.

When plug-ins are used the calculation of moves by the Artificial Intelligence may take significantly more time in comparison to the situation where no plug-ins are used. This is due to their dynamic nature. There was a decision made rather to give more freedom to the developers in chapter 3.8 of the plug-ins than to limit them and thus there is a mechanism implemented which can alter the rules of the game in a way. But injecting this dynamic code into the core of the game may lead to a noticeable slowdown of moves calculation.

During the work on this thesis there was an evaluation function (9) for the states of the game proposed in chapter 3.9.3, which approximates the value of game states well. So well, that it is better not to use the simulation phase of Monte Carlo Tree Search and instead it is better to use this evaluation function. Maybe if there would be multiple simulations done in the simulation phase of Monte Carlo Tree Search, the results would be more accurate. On the other hand it would slow down the computation because the simulation can take significant amount of resources, as described in chapter 3.9.3.

The contribution of this work is that it showed that Monte Carlo Tree Search can be used for Quoridor and in general for multi-player games even if there are unknown elements, such as plug-ins involved.

## 6.1 Future works

It would be possible to design some experiments in which the Artificial Intelligence would play the game against itself but with a different set of parameters. The results of the games would then be statistically evaluated and as a result of the evaluation the best set of the plug-ins might be picked. The parameters have a significant impact on the quality of the results of the Artificial Intelligence.

If possible it would be nice to develop the game further and to form a community around it.

# 7 Bibliography

1. **Brenner, Matyáš.** Bachelor thesis: Sector 66 - A Modular Board Game. 2012.

2. **Turing, Alan.** Computing Machinery and Intelligence. Mind. October 1950, Vol. 59, 236. Available at http://www.loebner.net/Prizef/TuringArticle.html.

3. MSTE - Office for Mathematics, Science, and Technology Education. Buffon's Needle. [Online] College of Education @ University of Illinois. [Cited: 24 July 2015.] http://mste.illinois.edu/activity/buffon/.

4. **Anderson, Herbert L.** Metropolis, Monte Carlo and the MANIAC. Los Alamos Science. 1986, 14, pp. 96–108. Available at http://library.lanl.gov/cgi-bin/getfile?00326886.pdf.

5. Random Samples. Science. 4 February 2000, 287, p. 799. available online: http://oldweb.cecm.sfu.ca/~jborwein/algorithms.html.

6. Game complexity. Wikipedia. [Online] [Cited: 26 July 2015.] https://en.wikipedia.org/wiki/Game_complexity.

7. **Browne, Cameron and et.al.** A Survey of Monte Carlo Tree Search Methods. Computational Intelligence and AI in Games, IEEE Transactions on. 3 February 2012, Vol. 4, 1, pp. 1 - 43.

8. **Kocsis, Levente and Szepesvári, Csaba.** Bandit based Monte-Carlo Planning. Berlin, Germany : Machine Learning: ECML 2006, 17th European Conference on Machine Learning, 2006.

9. **Auer, Peter, Cesa-Bianchi, Nicolò and Fischer, Paul.** Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning. 2002, 47, pp. 235–256. available at
http://moodle.technion.ac.il/pluginfile.php/192340/mod_resource/content/0/UCB.pdf
.

10. **Russell, Stuart J. and Norvig, Peter.** Artificial Intelligence: A Modern Approach. 2nd. 2003. ISBN 0-13-790395-2.

# 8 List of Tables

# 9 List of Abbreviations

| | |
|---|---|
| BFS | Breadth-first search |
| HTTP | Hypertext transfer protocol |
| IDE | Integrated Development Environment |
| LAN | Local area network |
| MCTS | Monte Carlo Tree Search |
| NAT | Network Address Translation |
| SVN | Subversion |
| TCP | Transmission Control Protocol |
| UCB | Upper Confidence Bound |
| UCT | Upper Confidence Bounds applied for Trees |
| WCF | Windows Communication Foundation |

# 10 Attachments

All the attachments are located on attached CD.

**Attachment 1**

Attachment 1 contains documentation generated from the comments of the implementation of this thesis. It is located in folder *Documentation*. It is run by file index.html.

**Attachment 2**

Attachment 2 contains source codes of the diploma thesis. These codes are available in folder *Sources*.

**Attachment 3**

Attachment 3 is an installer of Sector 66. It is located in folder *Install*.

**Attachment 4**

Attachment 4 is Microsoft XNA 4.0 Refresh redistributable. It is available in folder *XNA*. It can be also downloaded from https://www.microsoft.com/en-us/download/details.aspx?id=27598.

**Attachment 5**

Attachment 5 is a zip file containing the data which would appear in the installation folder after using the installer in Attachment 3. This zip file is located in folder Sector66Zip.