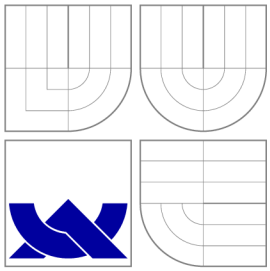
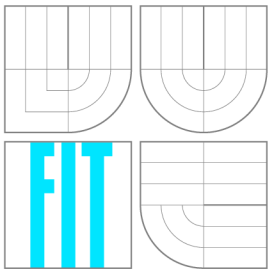


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ARTIFICIAL INTELLIGENCE AND USER INTERFACE FOR BOARD GAME OF THE SETTLERS OF CATAN

UMĚLÁ INTELIGENCE A ROZHRANÍ PRO HRU OSADNÍCI Z KATANU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ROSTISLAV HUSA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL KOŠÍK
SUPERVISOR

BRNO 2015

Abstrakt

Tato bakalářská práce se zaměřuje na tvorbu uživatelského rozhraní, datové reprezentace probíhající hry a návrh a implementaci umělé inteligence protivníka pro deskovou hru Osadníci z Katanu v její základní verzi. Motivací je zde jednak skutečnost, že ačkoliv se tato hra již dočkala několika počítačových implementací, žádná z nich nenabízí umělou inteligenci na úrovni, která by pro zkušeného hráče byla výzvou. Za druhé pak to, že analýza rozhodovacího procesu hráče v průběhu hry je netriviální problematikou, jejíž rozbor z pohledu umělé inteligence slibuje přínosné poznatky. Tvorba uživatelského rozhraní a datové reprezentace hry není hlavním zaměřením této práce, je však nezbytným krokem k tomu, aby bylo možné vytvořenou umělou inteligenci patřičně otestovat a odhalit její případné nedostatky.

V první části je prezentován stručný úvod do problematiky uplatnění umělé inteligence na poli stolních a deskových her, ať už jako studijních příkladů k ověření algoritmů se širším uplatněním, nebo přímo s cílem překonání lidského hráče. Také je zde stručně zmíněna historie hry a její klíčové charakteristiky, především pak ty vztahující se právě k pohledu na problematiku z hlediska umělé inteligence. Závěr první části pak shrnuje plánované kroky a cíle této práce.

V následující části je pak představen přehled jednotlivých oblastí umělé inteligence, jejichž dosavadních poznatku může být uplatněno při analýze problematiky, na kterou se zaměřuje tato práce. Dále jsou pak také zmíněny možné přístupy k návrhu inteligence hráče (bota) jako takové spolu se stručným přehledem jejich obvyklého nasazení, zhodnocením jejich vlastností a shrnutím očekávatelných výhod a nevýhod.

Třetí část se zaměřuje na podrobnější analýzu hry osadníci z Katanu jako takové. Představuje jednotlivé herní mechanismy, součásti herní plochy a jejich význam, možnosti interakce hráčů mezi sebou a dosažení cíle hry. Následuje rozbor jednotlivých fází herního kola s představením rozhodnutí, která hráč v jeho průběhu činí, zhodnocení jejich dopadu na hru a jejich důležitosti pro úspěšnou herní strategii. Zvláštní důraz je pak kladen na fázi obchodu, která je kooperativním prvkem hry a vyžaduje tedy nejen nalezení optimálního postupu jednoho z hráču, ale nalezení kompromisu mezi hráči. V závěru této části jsou pak shrnuty obvyklé herní strategie a případné přechody mezi nimi v půběhu hry.

Další kapitola je věnována představení návrhu výsledné aplikace. Je přiblíženo plánované rozdělení funkcionality do jednotlivých segmentů implementace - uživatelského rozhraní, datového modelu probíhající hry a modulu umělé inteligence bota. Dále jsou podrobněji rozebrány detaily každého z nich a jejich očekávatelná úskalí. Následuje rozbor možných přístupů ke konkrétním způsobům implementace inteligence hráče (bota) v pořadí priority realizace. Zmíněna je také forma zápisu logu z probíhající hry, umožňující monitorování chování aplikace jako takové i jednotlivých rozhodovacích kroků umělé inteligence, což je nezbytné pro ladění a případné opravy programu.

Na to navazuje podrobný popis skutečné implemetace shrnující detaily, které zůstaly v předchozí sekci nevyjasněny, nebo se v implemetaci oproti návrhu z nějaké příčiny změnily. Do větších podrobností je zde také rozebrána funkce uživatelského rozhraní a práce s ním.

Následující sekce je věnována zhodnocení výsledků. Jsou stručně představeny testy použité k prověření toho, nakolik aplikace splnila vkládaná očekávání a v co nejpřehlednější formě shrnuty a sumarizovány jejich výsledky. Prověřeno je srovnání jednotlivých implementací umělé inteligence proti sobě a především srovnání ve hře proti živému hráči.

Závěrečná sekce pak zhodnocuje výsledky práce jako takové a zamýšlí se nad možnostmi navázání na učiněná zjištění a dalšího využití aplikace.

Abstract

Subject of this thesis is creation of graphical user interface, internal data representation of going game progress and design and implementation of player artificial intelligence bot for a game of The Settlers of Catan in the basic version of the game. One part of the motivation behind the idea is the fact that while the game already has several computer implementations, none of them can so far be a challenging opponent for an experienced player. Another reason is that analysis of player decision making process presents a nontrivial problematic that can bring valuable knowledge when studies from artificial intelligence point of view. Creation of graphical user interface and internal representation of ongoing game aren't focus of this project, they however are necessary step to allow for proper verification of bot functionality and revealing possible issues that need to be addressed.

First section presents overview of application of artificial intelligence in the field of board and tabletop games, both as case studies for a verification of algorithms with wider application and with the purpose of challenging and overcoming ability of human player. This section also lists a brief history of the game and highlights its characteristics, focusing on those relevant to potential bot design. End of this section the summarises goals and planned steps of this thesis.

Next section brings more focus on individual areas of artifical intelligence theory, highlighting those that can be used for the analysis of problematic at hand. The suitable options of specific approach to the bot design are also mentioned here, giving brief review of their usual primary use and their respective strengths and weaknesses.

Third section further explains the Settlers of Catan game itself. Introduces individual game mechanics, game board components and their function as well as the means of player interaction and point scoring. This is followed by more detailed description of individual game steps, overviewing the decisions players are making during each of them and their impact on successful gameplay. Further emphasis is given on the trading, as it represents the cooperative aspect of the game and thus doesn't require one player's optimal choice but the ability to find consensus between players. Lastly this section mentions commonly used game strategies and possible flow of the gameplay between them.

Following section focuses on design of the planned application, intended distribution of functionality between individual components - graphical user interface, data representation of the game and framework of the player bot. Followed by explanation of their respective details and expected issues that need to be resolved. Next up is overview of the possible ways of bot design in order of implementation priority. Importance of log record necessary for proper monitoring of both application functionality and bot decision making is also mentioned here.

As a direct follow up, next section goes into detail of actual application implementation, explaining aspects that were not clarified in the previous section or that changed against the original design. This section also describes specifics of user interface and its use.

Next section evaluates results of the whole project. This opens with introduction of tests perpared to measuere application performance, followed by listing of their outcomes and summarisation of the results. Individual implementations of the arificial intelligence are compared against each other and more importanly against a live player.

Last section draws conclusion from the outcomes of the project, presenting possible use in the future and prospects of a follow up work and project extension.

Klíčová slova

Umělá inteligence, Osadníci z Katanu, Hry s neurčitostí, Kooperativní hry, Bot, Java, Grafické uživatelské rozhraní.

Keywords

Artificial Intelligence, Settlers of Catan, Games of Chance, Cooperative Games, Bot, Java, Graphical User Interface.

Citace

Rostislav Husa: Umělá inteligence a rozhraní pro hru Osadníci z Katanu, bakalářská práce, Brno, FIT VUT v Brně, 2015

Umělá inteligence a rozhraní pro hru Osadníci z Katanu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Košíka

.....
Rostislav Husa
May 20, 2015

Poděkování

Chtěl bych tímto poděkovat panu Ing. Michalu Košíkovi za vedení a odbornou pomoc při tvorbě této práce.

Contents

Chapter 1

Introduction

Board games are one of the areas that artificial intelligence has been challenging since the day it was established as a field of computer science. Starting with relatively easily algorithmized deterministic ones such as Gomoku (Five in a Row) and Draughts (Checkers), through navigating massive state-space complexity of Chess and Go, to games with elements of chance, games with imperfect information, games based on negotiation between players incorporating level of trust and emotional stance, or any combination of the above. Proper choice of algorithm and right heuristics have proven growing success across the whole field. One example for all being Deep Blue versus Garry Kasparov [3]

Despite that, there's still much left to learn and explore in this area, be it refinement of existing solutions for optimal compromise between quality of decision and time necessary to reach it or introduction of new methodologies allowing for different approach.

The field of board games also simplifies application of artificial intelligence solutions somewhat on merit of being systems that already come with strictly defined rules and level of abstraction isolating things down to the key problematic, thus bypassing necessity of most preparatory work - careful study what can and cannot be left out in abstraction - that most other artificial intelligence solutions demand before their application. This makes it ideal test bed for comparison of various types of approach, both for academic study offering better understanding of their principles, differences and similarities, as well as for benchmarked measurements of speed, accuracy, resource requirements and effectivity for competitive and commercial use.

The purpose is not just solving the problem game represents or proving a relating theory, but often also providing human player with practice opponent for learning the game mechanics or acting as a fill-in for multiplayer games. Sometimes findings from board game projects can even become basis of solution for more complex real life problematics.

First part of this thesis introduces board game Settlers of Catan, why are the game's mechanics worth attention of artificial intelligence research and goal of the thesis. Following part reviews the areas of artificial intelligence that are relating to the game's mechanics and can find application in designing a game bot. Third part goes more into details of the game mechanics and highlights points of interest. Following two sections are given to design of an application consisting of a center point managing game board and connecting together game bots and user interface for human player. Then, results of the application run and bot decision making are presented and in the final chapter, conclusion is drawn based on those.

1.1 Settlers of Catan

Settlers of Catan (SoC) is a modern multiplayer board game first introduced in 1995 that quickly gained popularity all across the globe. The key characteristic of modern board games (also known as designer board games) is being designed from a scratch by single person or group of individuals with consistent aproach to both rules and the game's theme, often reflecting specific historical events or branch of human activity. As opossed to traditional board games such as Chess or Draughts, that have long history, but details of their origin are unknown, their theme is rather vague, and that often come in many regional sub-types and variantions. Modern board games are, as name suggests, relatively recent idea, first pineered by Monopoly and Diplomacy in the middle of 20th century, that started as a rather niche hobby, but eventually gained much wider following, in recent years often rivaling popularity of well-established classic board games.

Modern board games can be further divided into so called American style games and Euro style games. The division doesn't so much reflect place of the origin (even if that often is the case as well) but rather distinctive characteristics of the game. American style games are characterised by bigger influence of luck, generally faster pace, and player conflict being rather direct, and if they are multiplayer games, player elimination often takes place at any point throughout the game. With Euro style games the element of luck has much lesser importance or is not present at all, player conflict is usually indirect and takes form of competition over resources or collection of points. Player elimination does not happen until game conclusion and game mechanics often incorporate some sort of negative feedback (the more ahead of the others player is the more difficult it becomes for them to make further progress) giving other players better chance to catch up. There's also typical differences regarding game themes, artwork, material of components and such, but those are not important for our view on the problematic. The important point is that despite incorporation of the element of chance (in form of dice), SoC belongs disctinctively to the latter category [1].

Exact SoC game mechanics will be further discussed later but it's already obvious that makes SoC ideal candidate for further study of game mechanics in attempt to come up with ideal strategy for each game. The problematic is made even more challenging by the fact that while rules remain the same, exact setup of game board and therefore optimal strategy is different with each game.

1.2 Goals

This thesis sets to meet several goals, first of them is explaining the necessary background for understanding artificial intelligence principles and mechanics that could be used for designing a game bot, followed by analysing rules of the game using this knowledge and designing how bots for playing it could be implemented.

Game platform with user interface and at least one of the bot types will be implemented and tested against human player and eventually each other. The application should be designed with modularity in mind allowing for easy extensions and creation of additional bot types.

Finally, results of bot performance will be analysed, matched against expectations rising from theoretical research and studied for possibilities of improvement or follow-up work.

Chapter 2

Theory of Artificial Intelligence

2.1 Game theory

Game playing is one of the specific kinds of problems that can be approached with artificial intelligence. We're assuming existence of two or more players that both fully understand and strictly follow game rules, in other words, will play to the best of their capability, but will not cheat, and that each strive to win. In its basic form there is two players taking turns that both have full knowledge of the game state and make decisions based on evaluation of available options. Said evaluation is usually done by function, or heuristic, that translates next game state that would follow after each decision into a numerical value, or several such values that are used for weighted sum. The decision is then made simply comparing these values (sums) for a maximum. [9] It would, however, be quite short-sighted to make decisions based only on the present game state alone and a player might easily lead themself into a losing path that's headed by momentarily gain. A simplistic example of this could be chess player valuing game state only by sum of their own and opponent's pieces falling for a gambit. More sophisticated way to approach this is to evaluate game several turns ahead looking at both player's and their opponent's options, considering that while the player will always try to take most advantageous option (one with the highest associated value), their opponent will try to take the one that's most disadventageous for the player (one with the lowest value). This can be noted in form of an and-or tree with acting player's node being or composition, they win as long as at least one of the child nodes leads to victory, and opponent's node being and composition, player wins only if all of the nodes lead to victory.

The process of going through the tree can be implemented by recurrent MinMax algorithm exploring branching of possible game states to a desired depth. This can be either until winning move is found, which can be done for games with reasonably small state space such as tic-tac-toe [1] , or limited by available processing resources - time or memory for game state expansions. MinMax algorithm can be further optimised by leaving out unnecessary exploration of some nodes. This is done by noting and carrying over the best (worst) evaluation from already explored branching and when it's confirmed that currently explored branch cannot lead to better (worse) outcome, the exploration is terminated and moves on to the next branch, if any. This optimised algorithm is called Alpha-beta pruning. This, or other forms of pruning, is especially helpful for games with large and quickly expanding state-space such as chess.

[1] In reality if both players choose best play in a game of Tic-tac-toe, the game will always end in a draw. But it's good example of a game with reasonably small state space.

2.2 Games of chance

Games of chance are non-deterministic games where outcome to some degree depends on element of chance. That is usually present in form of dice, cards or some form of lottery. Games where the role of chance is dominant, such as roulette or bingo, are not interesting for artificial intelligence research as for the scientific purpose the players are making wild guess rather than informed prediction. On the other hand, games where the element of chance goes hand in hand with players' skill and where the quality of players' decision needs to accomodate for the randomness are well suited for artificial intelligence analysis.

As in above case of strictly deterministic games, the basic scenario would be zero-sum two player game where both players have full knowledge of game state and consequences of individual decisions can be evaluated. The difference is in each players' decision being preceded (or followed) by a random event. Game state expansions noted as a tree structure therefore require new type of nodes - chance nodes - that are evaluated by weighted average of the follow-up node evaluations. This part works exactly the same no matter if the following action belongs to the active player or their opponent. And again, on their active decision making the acting player strives to pick option with the best evaluation and expects opponent to pick the one that gives acting player the worst evaluation.

Exploration of node expansions can be implemented by modified version of MinMax algorithm called ExpectMinMax. Further improvement of its efficiency by pruning unnecessary expansions is more difficult than in case of simple MinMax but even more important because chance nodes further contribute to rapid growth of state space.

Special attention should be paid to evaluation function. First of all because existence of nodes with extremely high or low evaluation could skew decision in their path even if they are very unlikely. For example if player gaining advantage would evaluate at +1 and gaining disadventage at -1 while game win and loss would be +500, respectively -500. Choice leading to 1 chance to win and 99 chance to gain disadventage would evaluate as +4.01, while one with 100 chance to gain advantage would only add up to +1. The prior choice would be preferred, which, in most cases, isn't desired behavior.

The second point being that well designed evaluation function allows for effective pruning. Some of the nodes don't need to be evaluated fully before getting pruned, it's enough to have their best or worst estimate for comparison with best (worst) evaluation of already evaluated nodes. One of the possible ways to approach this is Gamma-pruning that builds on Alpha-beta pruning. Overall efficiency will, however, depend not only on quality of evaluation funcion and pruning but also on several other factors such as order or branching or. So far there is no known universally best approach and design of decision mechanics for each game should be approached with specifics of its problematic in mind. [5]

2.3 Cooperative games

Cooperative games have been introduced relatively recently. One of the reasons to that is the fact they require whole new level of complexity over traditional games played by two competing players or two opposing teams. Cooperative games instead introduce several player factions that can negotiate and make alliances with each other. Pioneer of this type of games would be Diplomacy introduced in 1950's and with theme of power struggle in Europe during 20th century. Game has no element of chance so the success depends only on player ability alone.

Obviously this type of games proposed interesting challenge for artificial intelligence research. Success in cooperative game can depend on succesful dealing with other players in a way that's beneficial for both sides of the deal, because that's what makes it acceptable for the other party, and scaling these advantages to get player ahead in the game.

In case of diplomacy, another important aspect is element of trust between individual parties, because game works with lot of nonpublic information and alliances can be broken at any point. It's necessary to be able to estimate whether the existing deals are reliable or should be expected not to last.

One thing that can help with that and recognition of player standing with each other is understanding of player emotions. In this case within context of game actions, player should not only consider how would action they're making affect them but also how are the other players going to perceive it and feel about it. Emotion Annotation and Representation Language (EARL) gives description to several dozens of them, which is way more than necessary for such project, where degrees of satisfaction / dissatisfaction will suffice. [2]

These aspects have to be used in coordination with one another balancing logical and emotional sides of decision making, as neither of those in their extreme form lead to optimal decisions. As to what is the exact proportion of each for the optimal results depends on problematic in question and needs to be thoroughly tested. [7]

In a resolution of problematic where element of cooperation has only partial role it could bebeneficial to deploy cooperative emotional system as a component part of decision making procedure, eg. as an agent in agent system.

2.4 Expert system

The history of expert systems as a distinguished field of artificial intelligence dates back to 70s. The basic idea came from realising that quality of data is often more important than complexity of mechanics processing it. Expert systems were intended to work with knowledge base provided by specialists in a given area of expertise, experts, and emulate their counsel.

First expectations were unrealistically high, aiming to create universal system that can work over any kind of knowledge base and provide quality advice for any desired expertise simply by changing provided knowledge base. Later computer scientists settled for more realistic aim to create systems focused on one specific field of expertise, often inteded to be used as embeded part of more complex application.

There is no formal definition of expert system, but generally accepted understanding is a computer program emulating decision making process of an expert solving complex problematic using efficiently encoded explicitly expressed expert knowledge acquired from specialist with the aim to emulate problem-solving ability comparable to that of a specialist in given problematic. The goal is quality of decision made, not exact emulation of decision making process. [4]

The important point is strict separation of knowledge base and decision making mechanism. The main advantage is that both knowledge base and decision making logic can be easily modified, as there's no inherent dependancy on one another save for defined format of knowledge base records. This allows for dynamic growth of knowledge base and ensures high level of reusability, expert system logic can easily swap knowledge base and vice versa. Similar modification in application where knowledge and logic are inherently interconnected would take much more effort or might not be possible at all. Another obvious plus is possibility of independant development for each component.

The information expert systems use has to be expressed explicitly in form of knowledge base. That means there is no ambiguity how to interpret it. There can, however, be a degee of uncertainity in the information itself, furter discussed below. This transparency of knowledge base makes it not only easy to modify (extend) but also easy to read, so it can also be used as a study material for newcomers into the field of expertise. Knowledge base is also idependant on expert building it and can easily change ownership over the years of its existence and collect contributions from many specialists, up to merge with another knowledge base in the same field of expertise.

Content of knowledge base should usually cover wide scale of options ranging from well known textbook cases covering common situations to specific cases dealing with most peculiar scenarios. The knowledge can be both well defined methodologies backed by theory and proofs or personal heuristic proven to work by specialist's practice with but with no theoretical backing. Reality shows that this personal know-how is what makes the difference between the distinguished specialist and average practicioner in the given field of expertise.

This setup provides expert system with general knowledge in the given field. To get advice for specific scenario a dialogue is initialised between the expert system and its user. The user can be less qualified specialist or even a layman. In the dialogue expert system asks questions to collect enough information about the problematic to make suggestion. The questions don't have fixed order and are generated dynamically depending on specific subject and previous answers to quickly collect most relevant information.

Expert system has to be able to deal with uncertainity. Both as part of its knowledge base, for example when heuristic recorded as part of the knowledge base is known to work

for nine out of ten cases or symptoms are usually pointing to specific problem but not being sole reliable indicator of its presence, and in the information acquired from user dialogue. That's because user might not be able to answer some of the question accurately or at all.

Expert system should be able to provide advice even when part of the information is missing, buidling complext decision structure instead of interpreting facts as simple true/false conditionals. It is also entirely possible to come up with several equally likely answers instead for single decision. Either way expert system should be able to, if requested, back given advice up by detailed explanation of reasoning that lead to the conclusion as well as to handle any follow-up questions to adjust or further specify the problematic.

Expert systems can be further divided into three cathegories - diagnostic, planning and hybrid. Diagnostic systems are used to match one of the possible hypotheses with information input. The decision making process is therefore finding the best match and queries asked in user dialogue are chosen by their informational yield. Planning systems are used for scenarios with know initial and goal states and the desired outcome is optimal sequence of actions (applied operators) to make transition between the two states. Solving this task by traditional means would lead to massive state space expansion, especially if there are many possible operators and large number of steps between initial and goal state. Expert knowledge is used to reduce this expansion by focusing only on promising branches speeding up the process significantly. Hybrid systems combine aspects of diagnostic and planning, one of examples could be learning assistant using diagnostic to assess student's knowledge and planning to suggest best study schedule. [4]

2.5 Agent systems

The idea behind distributed artificial intelligence is distributed problem solving. As frequently observed in nature and human society, collective of individuals cooperating together can solve much more complext tasks than single individual ever could. In computer implementation, these individuals are called agents.

To successfully apply this methodology in artificial intelligence there's several issues that need to be addressed - competency of individual agents, form of communication and coordination between the agents and structure of their colective. The possible structures are all agents being equal, agent hierarchy, and existence of single decision maker and scheduler coordinating the other agents.

Distributed artificial intelligence can be further divided into two main cathegories - distributed problem solving and multi-agent systems. Distributed problem solving focuses on splitting solved problem into several sub-goals that can be assigned to modules, solved separately and then composed into final solution. The agents don't have to be fully independant and some parts of the distributed solution can be closer-knit together. The emphasis is put on solving the initial problem, so if one type of agent proves to make bigger contribution towards the overall solution it can be advance at the expense of the other agents. With multi-agent systems the empasis is put on independance of agents, their interacion is more dynamic and flexible. The enviroment should be designed to encourage cooperation towards the common goal between any type of agents. Even those with selfish priorities. This approach is suitable for widely defined, more generic problems. [4]

Either way, agent systems offer several advantages similar to human teamwork. The solution is found faster, even compared to simple parallely split processing, because each agent can focus on part of the whole problematic and only needs to have data relevant to that specific part instead of having to iterate over all of the data. Communication

between individual agents is also simplified and steamlined thanks to defined distribution of responsibilities, allowing for specialised agents to process their specific task separately and only forward their conclusion. There's also improve in reliability and flexibility of the system as the agents can be easily added, replaced, and possibly substituted for one another as the specific problem demands. When designing agent system, we need to aswer four basic questions [4]:

1. How is solved problem defined and described and how it will be distributed between individual agents? This is dynamic process as specific sub-problems may come up gradually through the processing. This also helps to define how individual pieces of the solutions will be put together to compose the origial problem solution.

2. How agents communicate? This doesn't cover only syntax of their messaging but also when the information are sent and who are they forwarded to.

3. Is there a mechanism to detec and resolve conflicts between agants?

4. Can agent access and work with information about actions and plans of the other agents?

In other words, distributed artificial intelligence is about designing and testing possible ways to answer there questions. Proper decomposition of the original problem can make agent interaction and communication efficient with no need for external coordination and speed up solution significantly. Two possible ways to approach problem decomposition are top-down, starting from the view of problematic as whole, and bottom-up, starting from view of individual agents.

There are three levels of agents' behavior. Most basic one is reactive agent that only responds to outside impulses. Reactive agent logic builds image of outside world in its internal representation and determines response to individual impulses depending on the impules itself and agent's current assumptions. Part of the response can be forwarding received signal or sending signal of their own to another agents. Next level is intentional agent that can set on individual goals and internally considers possibilities to reach them. Intentional agents can communicate their goals and present state of their curent belief, their internal image of the outside world, with other agents in attempt to organise cooperation with each other. The third level is social agent. Social agents work with behavioral models of the other agents making complex plans of cooperation with them.

Multi agent systems are most frequently deployed in several scenarions. One of them is to replace previously existing monolithical applications when they no longer allow for demanded modifications or extensions. Another is to incorporate proven solution as a part of another application, either to ease intergation or to keep proprietary knowledge of exact decision making process secret while allowing for its use a „black box“. Multi agent approach is also useful when designing application as a collection of modules with emphasis on interchagability and reuse of code. Next area where multi agent systems find use is modelling and simulation. Behavior of animal herds, insect collonies or even crowds of people can be easily described by multi agent system with single agent representing each individual.

Chapter 3

Analysis of game mechanics

As mentioned in the introduction, Settlers of Catan is multiplayer game with elements of chance, incomplete information and possibility of player cooperation. It is expected that the game will be played by 3 or 4 players. While it's technically possible to play by just 2, doing so will significantly reduce element of strategy in the game and will therefore not be subject of this thesis. Game is symetric, in the sense that all players have same starting conditions, same goal and same options to take on their turn.

Game has three zones, first of them is game board shared between players and fully visible at all times. The other is grip of resource cards that are individual for each player and hidden from the other players, although it is usually possible to deduce some of resource cards held by player. Finally there's development cards also individual for each player, those remain hidden until used.

Pieces placed on game board cannot be removed and remain on their position until end of the game, with exception of Thief marker which can relocate over the board. Spend resource cards are discarded and used development cards left revealed in player's possession.

Players take turns in fixed order and compete in collecting points, first player to collect 10 or more points wins the game. Players can only gain points on their own turn and it is therefore impossible for two or more players to reach 10 points simultaneously, there will always be a single winner.

Main way to collect points is construction of settlements and cities on the game board. Each settlement awards player 1 point and each city 2. It is worth noting that initial board setup already awards each player 2 points as each player begins the game controlling two settlements. Some of development cards discussed bellow can award player 1 additional point. Finally there's two additional conditions and first player to meet either of them gets corresponding award. Controlling each award is worth 2 additional points. One award is given to player controlling most revealed Knight development cards (at least 3) the other to player controlling uninterrupted road with most segments (at least 5). Obviously eiter award could be lost when another player suprasses the player currently holding the award. When there's a tie award remains with the player who met the condition first.

To summarise what above means in terms of available information - points earned from settlements and cities are public and cannot be lost, points earned from awards are public but can be lost, and points earned from development cards remain secret until they add up to 10 with player's other points. It is prudent to assume that when player holds some unused development cards, some of them might be point scoring.

3.1 Game board

Game board consists of 19 hexagonal tiles, 18 of which can provide resources, having assigned resource type and a number to be rolled for yield, the last tile is blank. The tiles are put together into roughly circular game board. Exact position of each tile and its number can either follow predetermined patter or be generated randomly each game. Settlement and City markers are placed at the edge of a tile, thus neighbourning with up to 3 tiles, roads are place along the tile's sides. Tiles forming the game board are wrapped by board frame and some of the frame segments are marked as harbours. Those are considered to be associated with the two tile edges (possible settlement/town locations) nearest to the marker. Amount of markers each player can place on the gaming board is limited. The maximum values are 5 settlements, 4 cities and 15 segments of the road. Once expended, no additional marker of the same type can be placed. There are 5 types of resources represented by resource cards. Those are also limited, specifically to 19 pieces per resource. Spent resource cards are returned to the resource pool, but if at any time all the resource cards of give type are held by players, no more of that resource can be acquired. Game incorporates mechanic preventing consitent hoarding of the resource card with the goal of denying them to the other players, futher explained below. These limits can affect game strategy, as it is, for example, impossible to win solely by construction of roads and settlements.

3.2 Turn order

This section briefly describes individual steps of the gameplay and corresponding player actions and decision making taking place during each of them.

3.2.1 Deploy

First step of the game after the game board has been assembled is deployment of initial settlements and roads. At this point all the information about the board composition is already available. Players randomly choose their order and each places single settlement marker and road segment adjacent to that marker. Thr only limitation at this point is that settlements have to be at least two tile sides (road segmets' distance) away from each other. Then, players place their second initial settlement in reverse order along with another segment of road adjacent to either settlement or existing segment of the road. The order reversal at this part is in place to balance the range of options each players have in settlement deployment. After the second deployment round each player draws their initial resouce cards, one each of the resources yielded by tiles neighbourning their second settlement, and first game turn starts.

3.2.2 Resource gathering

Regular game turn begins with active player rolling a dice pair. If their sum is other than 7 tiles with that number are yielding. That means that for each settlement neighbourning the tile the settlement's owner gets one resource card of the tile's resource and two resource cards for each town marker meeting the conditions (assuming there's sufficient resource cards left in the bank). On the roll of 7 there's no yield and thief marker is activated instead. Note that not only the active player, but all the eligible players can get the resource cards. Active player is, however, the only one that can spend and trade resource cards int the following step.

3.2.3 Resource spending

Player can spend resources either placing new markers on the game board or drawing action cards at appropriate resource cost. Cards can be drawn as long as there's some in the deck, but plancing markers has to meet further requirements.

- Road segment - has to be placed adjacent to player's existing road segment through tile edge that's either unoccupied or occupied by player's own marker.
- Settlement - has to be placed on tile edge touched by player's road segment and at least two segments (regardles whether occupied or unoccupied) away from other settlement and town markers on the board (belonging to any player). It doesn't matter if the edge is also touched by another player's road segment as long as the above requirements are met.
- Town - town marker can only be placed replacing player's existing settlement marker. Settlement marker is returned to the pool and can be used again to establish new settlement.

3.2.4 Action cards use

There's five types of action cards. They're drawn at random from a shuffled deck and are not reused over the course of the game. Exact counts of each type are listed in appendix. Player keeps the drawn card secret until played, which can be at any point of player's own turn after dice roll and resource gathering has been resolved.

1. Knight - player moves thief marker and scores the knight card towards Largest Army archievement.
2. Victory point card - player scores 1 additional victory point.
3. Progress - Roads - player builds 2 road segments at no resource cost (if able).
4. Progress - Monopol - player names a resource type and the other player hand him over all resource cards of that type.
5. Progress - Invention - player draws two resource cards of his choice (can be the same).

3.2.5 Thief use

Thief is the only marker that moves across the board. At the beginning of the game thief occupies Desert tile where it has no effect. Thief marker moves whenever player rolls 7 on the dice or plays Knight action card. If 7 was rolled, thief move is preceded by discard phase where each player holding over 7 resource card must choose and discard half of them (rounded down).

When thief marker moves to a tile, that tile becomes blocked and will not yield any resources until the thief marker moves somewhere else. Additionally player moving the thief picks one of the players owning settlement or town marker at the edge of affected tile (if any) and takes one resource card from them at random.

3.2.6 Trade

Another part of game strategy is resource cards trade. Trade can only be initiated by active player, who can either propose a trade to the other players or trade with game bank. When trading with other players the proposal is entirely up to the player, but most commonly will be in a form of one for one offer. Non-active players then have a chance to accept the offer (ideally in the same order as the game progression, but realistically on first come first served basis). Player can theoretically propose any number of offers. Trade with game bank is done at exchange rate of 4 resource cards of the same type for any one resource card. This rate can be further improved by controlling settlement or town neighbouring harbour marker. Harbour of specific resource type improves exchange rate of that one resource to 2:1 and harbour of unspecified type rate of any resource to 3:1.

3.3 Common game strategies

The succesful gameplay can be approached from several angles depending on the specific board setup and choices made by other players. Most common strategies are [8]

- The Ore-Grain Strategy - focusing on upgrading initial settlements to towns as soon as possible doubling resource yield and either trading away for the other resource or folloing into Card Builder strategy.
- The Wood-Brick Strategy - focusing on fast construction of roads occupying good spots for construcion of new settlements. Follow up should focus either on getting access to the other resources or forwarding into Bank Trade strategy
- The Card Builder Strategy - focusing on drawing Action cards aiming for Largest Army archievement, additional victory points and perhaps Longest Road archievement. With less emphasis on construction of additional settlements.
- The Balance Strategy - focusing on access to all resources allowing for flexible trade options and relatively simple construction of additional settlements.
- The Rare Resource Monopoly Strategy - only viable on very specific board setup, focusing on isolating access to the resource with lowest yield probability and using it as a negotiation lever in trade with other players (usually in form of 1:2 or even 1:3 deals)
- Bank Trade Strategy - focusing on one resource with a good yield probability and corresponding harbour acquiring the other necessary resources via trade with other players and 2:1 trade with bank.

Chapter 4

Application design

This section explains the ideas behind planned application structure, segmentation of problematic into logical parts, responsibility of individual packages and expected highlights that will be requiring extra attention. Analysis of possible approach for individual bot types is also presented with a brief description of each and its expected upsides and downsides. Only some of the planned bots may be actually implemented depending on complexity of the application as a whole.

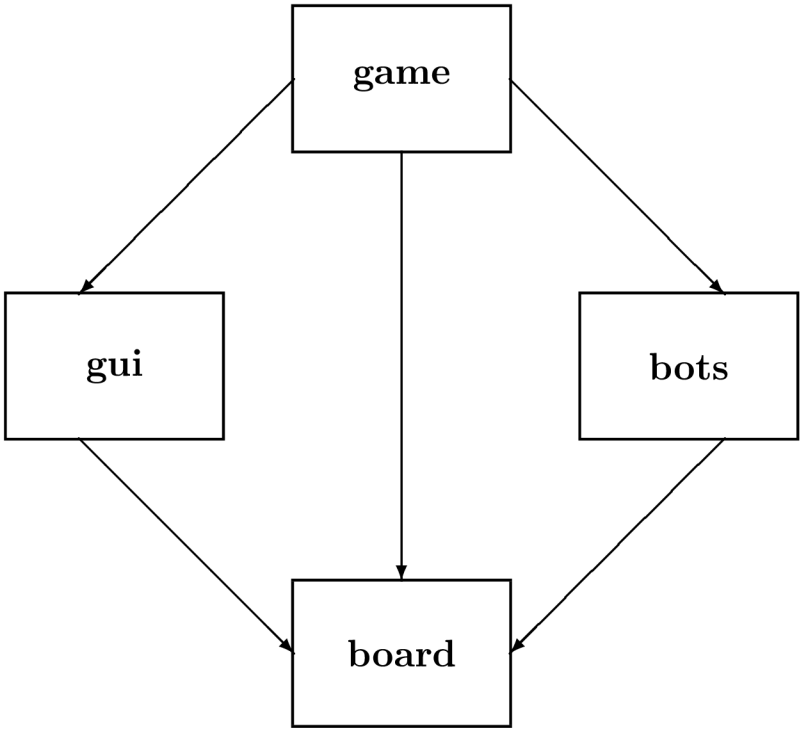


Figure 4.1: Designed package dependance

4.1 Game framework

First step in establishing groundwork of the application design is deciding exact extent of expected functionality. In this case that would be running game locally with up to 4 bots or a live player and up to 3 bots. Game playing over network or with multiple live players is not expected, thus is should be sufficient to have the application as a single piece of software. The program run will be driven centrally by a designated class that will be responsible for setup of game board followed by creation of bots and possibly with player UI. then it will be scheduling turn order between participants and processing their decisions and actions. This class will also be responsible for recording a log of game progress. while this solution doesn't meet the idea of „single responsibility per class" it should be acceptable here, as this will be the only class with access to the complete knowledge of the game situation and any other implementation of log recording would have to work through it anyway.

4.2 Game board

With hexagonal tiles and game mechanics using not just tiles themself but also their edges and vertices as board segments, maintaining both consistency of board state and easy navigation for bots and live player alike becomes obvious concern. Out of several possible approaches this project will be opting for pointy topped hexagon placement using adjusted even-r offset coordinates (the adjustment being head start of 2nd row versus 0th and 5th) [10]

Furthermore, during the game run there will be constant need for neighbourning elements access, ownership verification and distance measurement. As such it's for the best to create tiles, tile edges and tile vertices each as separate objects and have them interlinked with knowledge of their surrouding. Game board class will then provide wrap of sorts mediating access to this structure. Individual elements of the structure will not need any identification of their own as they will never be accessed directly, always either through neighbour or through the wrapper structure. Board object will also be responsible for accounting of resource cards and action cards after a consideration that they're also part of the board state even if they're not present on the board itself. Above in combination with the concept of central control means that neither bot nor the player interface will need to keep their own records of the board state as they can always retrieve needed information from the board object. This will eliminate risk of any data incosistency.

4.3 Player interface

Live player will interact with the game via Swing GUI. Key element of the GUI will be of graphical representation of the board state that will also serve as an input for board element selection. Given that after initial deployment board elements, that is - not player and thief markers, don't change position, they can be precomputed at the initiation of the player window and stored as the window's private variables for faster updates. Conveniently, functionality of Swing class Polygon also provides a way to check if point in a window area lies within the polygon's boundaries, which can be used for aforementioned selection. The canvas will respond to a mouse click as a whole, comparing mouse coordinates against aforementioned polygons, allowing for more steamlined response than would be possible with setting multiple sensory segments. Another part of player GUI will be status bar

monitoring public information not obvious from the board alone, such as number of resource cards and action cards held by each player. Finally there will be control elements, most likely in form of buttons, for initiation of player actions and another information bar with players private information - own resource cards and action cards. This interface will also allow user to spectate game run with bot players alone. In such case player control elements, private information bar and interactive function of game board visualisation will be disabled.

4.4 Bots

As explained above, game will be driven centrally by a designated class and thus bots will be designed as passive / responsive. When bot player's order comes, it will be prompted for a decision what action the player wants to take. Some of the information necessary to make such decision will be stored in the object representing the bot, this will be the case of information that doesn't change frequently and/or is proprietary to the bot alone and should be kept secret. Th rest of the information will be handed to the bot as a part of decision prompt call, which will be the case of information that changes frequently, is public or is provided by another player (bot or not) and couldn't be known until just before the call was made. The idea of having multiple bot types is ideal opportunity to use OOP principle of inheritance. Game driving class will assume all the bots are of generic parental bot type with clearly defined prompt call interfaces and specific bot types will be inherited from this class guaranteeing compatibility, this approach is also known as Liskov Substitution Principle [6] Hopefully, it will also be possible to steamline the design with player interface prompts allowing for very uniform operation of game driving class as it will not need to make any distinction between bot and live player GUI when interaction with their respective objects.

4.4.1 Evaluation bot

First bot type will be based on the idea of evaluation functions collection that will be able to examine and evaluate the game state at any given prompt and formulate the decision. The job will be made easier by the fact that the prompt call will already indicate what type of response is expected and thus what evaluation should be made. On the other hand, this means having to come up with quite a range of evaluation functions covering all the situations and game steps. Hopefully, some of those can be answered by reuse of the same or similar algorithms reducing the amount of necessary production code by recursive and/or parametrised calls. Expected evaluations will be including but not limited to:

- Selecting spot for a settlement marker placement - this evaluation will have to go through most of the candidates, discounting only those already occupied or within distance of 1 from already occupied. Given that even empty board range of possibilities is still well within double digits, even a linear time complexity shouldn't cause too much workload.
- Finding shortest road path connecting to a given spot - good opportunity to deploy Breadth-first search over a list of road segment sequences. Backtracking wouldn't be suitable because of possible path loops that can be easily treated in BFS but would cause more problems otherwise

- Finding best spot for a thief marker placement - similar problematic as in the case of settlement markers placement. In this case the range of considered options is even lower, so again linear complexity is fine.
- Decision to make token placement or an action card purchase - generally speaking, because of the thief mechanic preventing resource hoarding, resource cards should always be spent if at all possible, so the decision in this part is more about finding the best location for a marker placement using evaluation mentioned above.
- Action card use - action cards use is a little more complex because they can be safely kept in reserve. That should always be the case of some to make the most of their effect while the others can be spend immediately. This cannot be covered by any eastablished algorithm and will have to be decided on case to case basis.
- Trade making - this really consists of two steps, the first is decision what the trade tries to archieve, which is generally speaking getting together appropriate resource types to complete a purchase of card or marker. The following step will be proposing a deal that gets the player closer to that, expending resources not needed at the moment. Trade making will be partially done out of order as the bot has to respond to proposals made by the other players.

4.4.2 Agent bot

Next logical approach to the problematic is designing a bot that will decompose decision making and evaluation into corresponding sub-problems which can be resolved independantly and their solutions then put together into a composite decision. This allows for more flexibility in bot design as individual decision making parts can be exchanged without affecting the rest of the components. This bot type can also better utilise internal storage of the data, with each component (agent) keeping their own relavant parts. It can also possibly save some evaluation runs by having agent store last evaluation inputs and corresponding results, returning the same result on the same input and only running new evaluation on a different one. This wouldn't very well be possible on aforementioned bot type where isolating the problematic to a small enough fraction that such repeated calls can realistically happen would take considerably more effort than it would save. The problematic decomposition into agent responsibility can be following:

- Central agent - responsible for receiving prompt calls, passing them to corresponing agents and formulating decision from the information received back. This agent will also be cutting ties in case of conflict between the other agents.
- Build agent - responsible for evaluation of marker placement spots, monitoring marker placement of the other players and evidence of available and demanded resources for best.
- Trade agent - responsible for trade with other players and game bank both on players own turn and in response to trade offers from the others, cooperating with token agent on setting up trade priorities.
- Score agent - agent proposing best opportunities to score points - placement of more awarded markers vs striving for one of the archievements or victory point cards. This agent will also monitor points of the other players, both revealed and presumed and advise trade agent on threat level each of the other player poses.

- Thief agent - agent responsible for thief location evaluation used when thief marker movement takes place. Also measuring thief threat level - importance of blocked resource and impact on the other players.
- Card agent - overviewing use of action cards for their optimal effectivity.

4.4.3 Expert bot

Another possible approach to the bot design is as an expert system. This would obviously consist of two parts - the decision making process and the knowledge base. While the game is quite popular worldwide and many players have their preferred strategy, this knowledge has never been, as far as I've managed to research at the time of preparing this thesis, properly formalised. Considering individual steps of the gameplay, it becomes obvious that while some of the decision making knowledge can be formalised relativelty easily, such as preference of resources depending on board composition and resources already available, the other, such as appropriate use of action cards, would make such task far more challenging. Either way, this would first require clarification of a format in which the information will be stored followed by consultation with large enough group of skilled players. Even then, preparation of a decision making process that can effectively utilise this knowledge base would not be easy and while some extent of inspiration can be drawn from existing game playing expert systems, it would have to be designed from the grounds up just for the purpose. User working with the expert system will be represented by game core prompt consulting the bot for the next player action.

Another possibility is to use expert system approach as a component of another solution (eg. aforementioned Agent bot), utilising expert system functionality on for those sections of the problematic where it's most suitable, dealing with other parts of the decision making via another method.

4.5 Logs

While the application interface will allow visualisation of ongoing game, both with player participation and played by bots only, this alone would be insufficient for a proper analysis of game run and bot performance. Therefore it's necessary to be able to record a log of game progress all the way from board setup to game conclusion.

As to what exactly will be recorded is up to further consideration, log should most definitely provide enough information for a person reading it to be able to reconstruct the game run, even if doing so would be somewhat tedious. Given the fact that game uses hexagonal board where edges and vertexes matter, recording the board state will not he anywhere as easy as in case of games with square board. Ideal solution seems to be setting up coordinates system for tiles and player markers placement and recording action taken on the board via those.

As for the bot decision making, possibility of running sepatare log for each bot has been considered, but interpretation of those without the context of game as whole would be difficult, if at all possible, so the bot decision making will be included in main game log. Besides the action taken it should also record justification why the bot made that action, so result of evaluation function or similar indicator will be included along with the decision.

Next concern is recording of information not directly obvious from the game board state. Specifically, that means resource cards and action cards. Resource cards change

hands quite frequently, so the log should provide not only information of their movement but also summaries at certain points of a game (eg. after one round of player turns) for easier review. Action card traffic is comparatively more sparse, so just noting thier draws and plays with respective variable, if any, should suffice.

Point scoring should be noted as well, but while most important for the actual game play, it's not really so importat for the log keeping, as it can be easily deduced from seeing game board state at any point of the game. Trading between players will also be recorded in somewhat simplified state, partially because it will be happening a lot, each player can make any number of trade offers on their turn, they have to be considered by the other players and most of them get turned down. Second reason is that goal of trade doesn't have to be explicitly stated, it will be obvious form player's next purchase or just sought resource alone.

Logs will be recorded in plain text, use of a code format to save space shouldn't be necassary. Even a log of long 4 player game is eastimated to be well under 100kB, which is easily acceptable. Option of game replay from a log will not be implemented, because doing so would give no more information than can be deduced from existing log alone.

Chapter 5

Implementation

This section goes into exact steps taken in implmentation of the aplication, highlighting differences from the original design and explaining in further detail decisions made on code level. Individual subsections representing packages are introduced in chronological order of their implementation.

5.1 Board

As mentioned in previous section, first step toward implementation of the project had to be representation of the game board, as it is the centerpiece that every other segment of the application will interact with. Tile edges and vertices (road and town / settlement placements) can be generated upon board initiation as they all start blank and only gain further properties as the game goes on.

Meanwhile, generation of tiles is more complex, while there is predefined setup, mentioned in game rules as introductory game, most of the times player will want to have their board randomised. This randomised tile generation is made in a way that emulates the procedure done by players as described in game rules (only simplification being use of Fisher-Yates shuffle that can randomize corresponing arrays in single run) placing tiles spirally around the game board from any given edge, this functionality has been separated into its own class.

All three kinds of sub-elements are then stored in corresponding two dimensional arrays of variable 2nd dimension length. In following of the design idea, these classes can interlink and reference their neighbouring elements. Given complexity of this interlinking, doing so right on the creation would be difficult if at all possible, this is therefore done in separate run over the arrays already filled with corresponding objects. After that any element of the board can navigate its way to any other element without having to interact with wrapper class at all. Access to the elements via wrapper class is however still important because of the interaction with other game segments where navigation via coordinates still has its use.

Representation of harbours on the board frame has been excluded from having their own objects, they are noted as attribute of corresponding tile vertices and as an array of their respective types around the board frame used for display only. This separation of information doesn't matter, as the coherency can be verified on board generation (where one representation is derived from the other) and neither changes as the game progresses so there isn't any risk of inconsistency.

Another information mantained in this section are player resource and action cards.

These could have been separated into their own class as well, but because of the close interaction of resource pool and game board itself it will be a lot easier to verify consistency between the two within single class.

At this point it became obvious that quite extensive testing of this package will be necessary before implementation of any other elements that build on it to avoid missing any underlying issues. But at the same time without any classes that work with the board representaion or can properly visualise it, such testing would be complicated. Therefore it was decided to add functionality that records board game state in a plaintext, both tiles with markers and resource pool. This was used for debugging via standard output printing and also later used for recording of game log instead of originally intended recording via coordinates only. While that aproximately doubles size of the logs it makes them a lot more readable.

Some additional functionality was later added to the wrapper class for more convenient work with it, such as retrieving list of tile vertices occupied by specific player and executing actions corresponding to action card play. Those ass little functional complexity and make work with the class a lot more convenient.

This package also includes class representing a pair of dice used during the gameplay. Value of the roll might need to be read several times over fromvarying segments of application so this is prefered over storage in a variable.

While board and dice pair classes will always have instance of a single object in this application it was decided against making them singletons, because further extensions of the project might want to run more games paralelly.

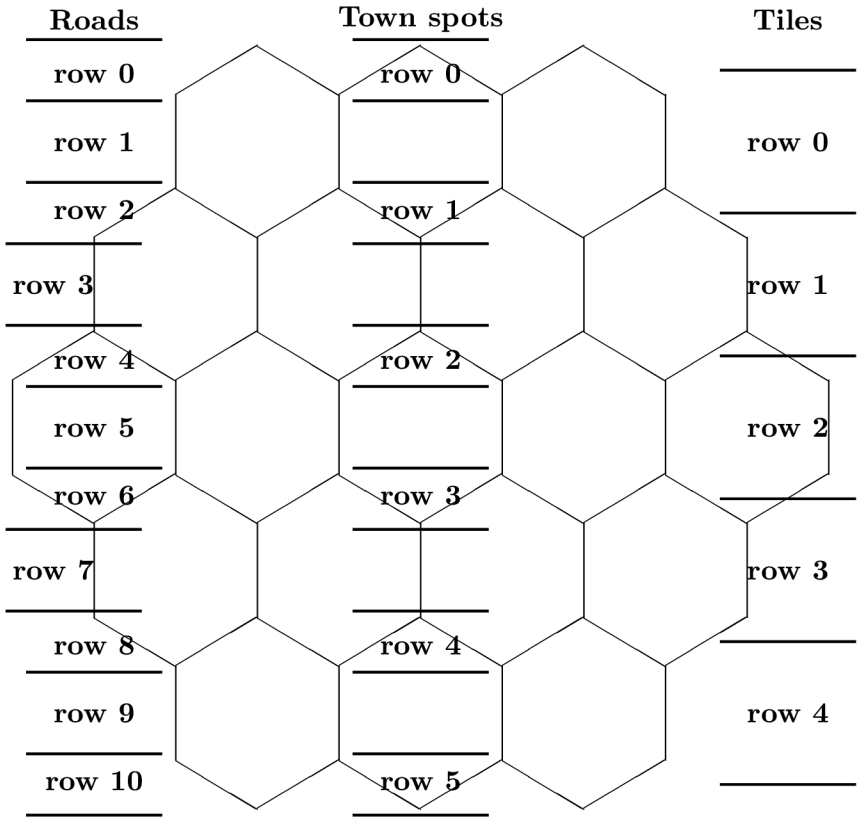


Figure 5.1: Row indexing of roads, town sposts and tiles.

5.2 Game

This class was implemented as a game centerpoint keeping tabs on game board, participating players and interaction between them, representing game run as a single thread run calling subsequent functionality as necessary. Initiation of the game object will create corresponding game board, expected number of bots and optionally player user interface, then schedule their rescpective order and process their choices.

At first, bot and player decisions were replaced by completly randomised or pre-set choices to verify that connection between this class and underlying game board works correctly, including attempts to execute disallowed moves to check if the design can handle refusing them correctly without exception calls.

Other than keeping the game flow and recording game log, this class is basically mediator between other components of the application. It also does monitor player scores, which could be done by board alone but from the application logic makes more sense here to know when the game run concludes. That is including the two possible archievements.

Another responsibility of this class is rolling dice and processing resource yield, essentialy taking over first step of player turn. While in real life game this is handled by players, the process is entirely set and doesn't involve any degree of choice on the player side and can therefore be automatised like this, reducing amount of code on player side.

5.3 Bot

First implementation of bot was done in accordance with planned design, creating series of evaluation functions that can respond to game prompts. First goal of the bot is spending available resources on whatever can be purchased this turn, taking into account best possible placement for a new markers. This is followed by decision whether to use some of the available action cards and finally trade with other players in preparation to have prepared the right resources for the next turn.

While the amount of trade offers made is theoretically unlimited, it was decided that opportunity for a trademaking by a bot will be reduced to a single offer per round, because the logs indicated that if the first deal isn't accepted right anway, chance of success of any subsequent one is very low. While going through all the possibilities wouldn't be issue for a bot-to-bot game, having to deal with that many prompts as a live player would slow game down a lot. Otherwise trade functionality works as designed using similar set of rules for both making and accepting deal from another player. Given the passive nature of the bots, this necessiates non-active bots to be prompted to update their trade preference whenever deal is to be proposed to them.

Pathfinding of road connections was incorporated into the board itself as an additional feature, because it's rather simple and there isn't any reason why various bot types should ever not have clear access to optimal path.

Evaluation of settlement and town placement works on the idea of balanced gameplay strategy where all resources have similar value to a player and tile quality is indicated chiefly by yield probability. Of course past the deploy step the distance to reach the next placement point is also taken into consideration, preventing the bot from expending resource on attempts to reach tiles too far away to be viable.

Thief placement evaluation works on the similar idea, except the intent is to cause most inconvenience to the other players, blocking tile with best yield for the most opponents. While player will shy away from blocking tile they're using themselves (indicated by negative thief effect evaluation), it can theoretically come to it rarely.

Action cards acquisition is prioritised second to token placement but over hoarding the resources to the next round. At this point attempts to predict drawn card type are not made as doing so with only fraction of knowledge necessary would be very complex and still unreliable.

5.4 User interface

Graphical user interface (GUI) was designed using Java Swing libraries. First version was designed to only monitor progress of game driven by central elements and using bot players, basically an alternative to monitoring game flow via log. Apparently a delay (driving thread sleep sequence) had to be introduced to slow display down to the level where it can be reasonably perceived by human observer (0.1 to 1 seconds between steps). This worked as designed without any problems.

Following step allowing for active player interaction however introduced some unexpected difficulties requiring further reseach and partial redesign of the application workflow. The issue in question is nature of Event Dispatching Thread (EDT) not allowing update of displayed elements from another (subsequently launched) threads. In practice this would mean that EDT launching game thread doesn't allow the game thread to update graphical interface until the control is passed back to EDT. Even a bot game launched by GUI in-

stead of the central class would put displaying on hold until the game run concludes, so no updates are shown, only the final game state. Passing the control from GUI actively used by player is arguably even worse, as the player isn't guaranteed to have up to date view of the board because the update might be in postpone.

Forcing the update from another thread is not possible as it would disrupt thread safety. Several solutions have been considered, most radical of them being complete redesign of the application separating GUI strictly from the game core and setting up communication between the two via established protocol, this would however require redesign of several other elements making most of the progress done so far void. So while this proves that the original idea of designing application as a single piece might not have been optimal decision, this project will push through with it to focus on confirmation of the other premises.

The GUI / EDT issue was instead solved by rewriting game run from a thread representation to repeated calls of function making single game step per call, or fraction of the step in case of player interaction, updating GUI after each if any change in game state was made. EDT will launch in short runs via SwingUtilities.invokeLater() giving the necessary space for update to execute. Control will be passed back and forth between game class and player window. This doesn't allow for originally intended hierachical order (window elements need to be able to call game class and the other way around) and would cause cross-reference between the packages and therefore the game class has been moved to the same package as the window elements. Exact step in the game workflow and expected calls are marked by status codes on either side of control passing as a first number of numerical array passed as call argument (the others being further specifications of the choice, such as resources being traded etc.) and corresponding return values.

GUI itself consists of launch window where parameters of a game can be decided, main player window with game board display canvases and buttons for a player interaction and several prompts for further choices invocated via buttons (player activity) or game flow (player response to prompt from another player). These subsequent prompts can also interact with game object directly, launching only functionality in their competence (eg. prompt for a parametrised play of action card can only make call of that one card) but still needing accees to it and therefore being located in the same package. Stepping into the calls out of order is prevented by disabling GUI elements that are not in order at given point.

These adjustments brought up one more issue, which is that when bot prompts other players for a response for its trade offer, this prompt should be answered in the order of gameplay. Unfortuntelly, in case of live player this means having to interrupt bot logic execution, waiting for an answer from the player. Following after that at the exact point of interruption would be quite difficult requiring complete rewrite of corresponding part of bot and game core logic. It was therefore compromised that live player will be prompted last after the bot players (if none of them agreed to the deal) which allows for easier application following no matter if the deal was made or not. This theoretically puts player at a slight disadventage against bot players but it shouldn't be much of an issue as actual gameplay trading between live players works on first come first served basis, which is not entirely fair either. To further simplify player's access to trademaking, deals asking for resources player does not have are turned down automatically without coming up as a prompt.

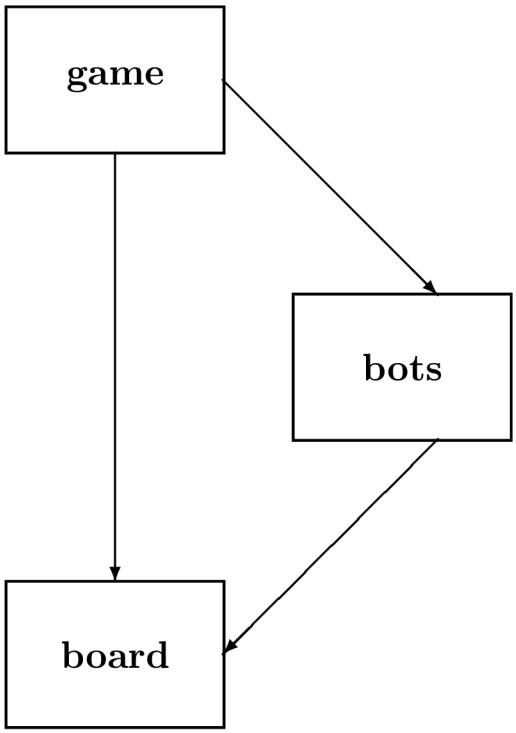


Figure 5.2: implemented package dependance

5.5 Bot versioning

Because of the scale of modifications and adjustments made during the application implementation, coming up with second bot type from the ground up in time for a proper testing became unrealistic. So instead of that project focused on further refinement of existing bot creating second evaluation bot version with dynamic allocation of resource and corresponding harbour priorities. While relatively small change in the amount of code it is expected that this will improve bots performance noticeably.

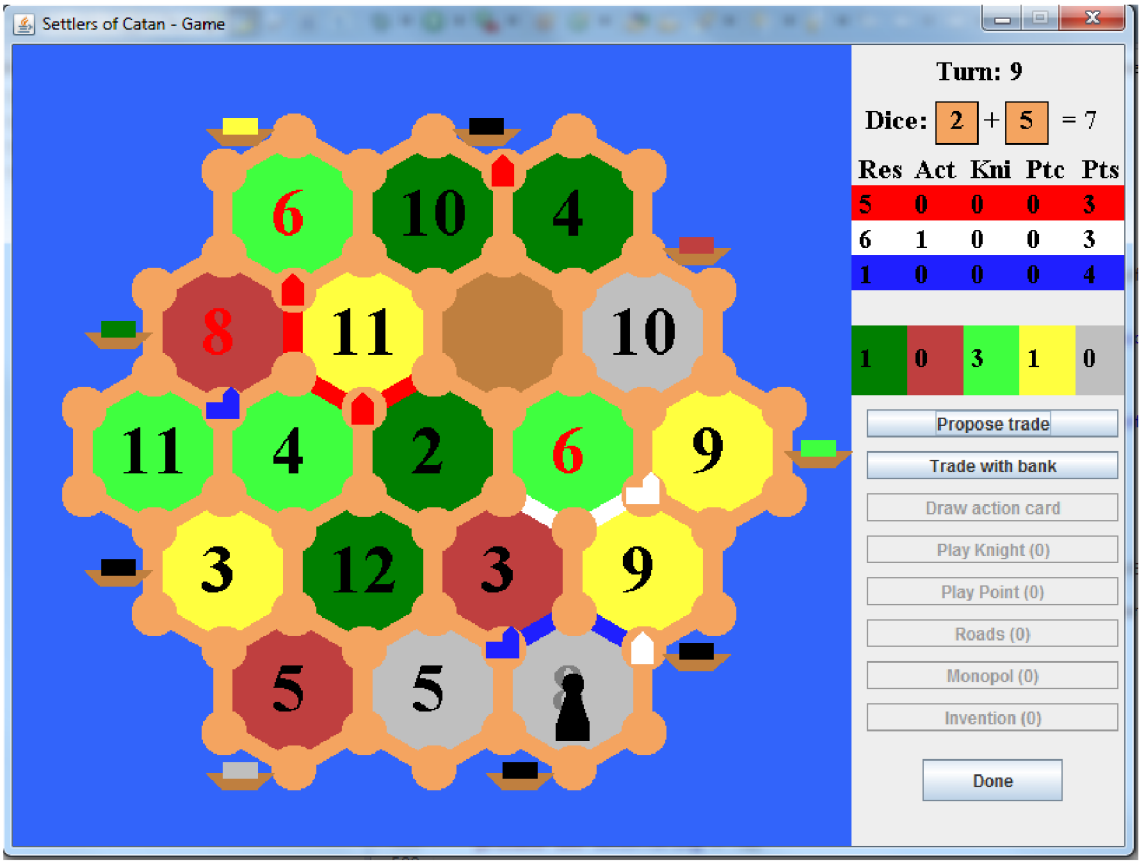


Figure 5.3: Application GUI - gameplay

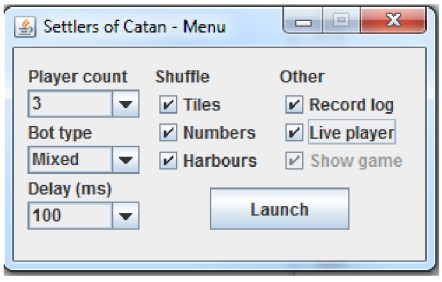


Figure 5.4: Application GUI - initial menu

Chapter 6

Results

This section measures quality and performance of final application, pitting bots against each other and against human opponents of varying skill. First measure of success is obviously win ratio and the point difference behind winning player, main goal of the bot is collecting as many points as possible. [1]

Another measurable quality is how long game takes, this will mostly make sense for a play between one type of bots where the speed of group finishing game as a whole should indicate how well they are adapted to the problematic. That is to say, measuring length of gameplay turns. The testing should also verify bot performance depending on randomness of board setup, bots should perform equally well on randomised and non=randomised board. Game between human players on this game version usually takes between 20 and 40 turns.

Application real time run is not a concer of there tests, sufficient to say tat even on weaker machine (single core CPU, 2GB RAM) game run finishes under one second.

Because the game incorporates element of chance, it's necessary to run each type of test several times to eliminate the influence of luck and measusure actual performance.

[1] When winning player finishes the game with more than 10 points, eg. by getting 2 points archievement at 9 points, the runned up is counted against the original target of 10 points.

6.1 Game length

Game length depending on bot complexity and and randomness.

run number	two bots		three bots		four bots	
	turns	lead	turns	lead	turns	lead
run 1	45	1	27	3	15	3
run 2	32	4	28	4	29	5
run 3	50	3	32	1	25	2
run 4	37	5	23	3	19	3
run 5	42	2	36	1	32	1
run 6	35	3	27	3	28	2
run 7	55	2	32	4	19	2
run 8	26	5	33	3	23	1
run 9	45	2	21	2	17	3
run 10	34	3	41	3	24	2
average	40.1	3	30	2.7	23.1	2.4

Table 6.1: Dummy bots preset board

run number	two bots		three bots		four bots	
	turns	lead	turns	lead	turns	lead
run 1	63	4	39	2	23	5
run 2	48	5	29	2	22	2
run 3	54	3	32	2	22	2
run 4	63	2	32	1	25	1
run 5	58	6	41	2	32	2
run 6	61	4	35	1	17	1
run 7	81	4	21	3	22	3
run 8	52	1	18	3	23	2
run 9	47	6	25	2	36	2
run 10	59	5	34	2	24	4
average	58.6	4	30.6	2	24.6	2.4

Table 6.2: Dummy bots random board

run number	two bots		three bots		four bots	
	turns	lead	turns	lead	turns	lead
run 1	22	6	18	5	27	2
run 2	37	3	29	1	25	2
run 3	31	1	24	1	24	1
run 4	30	2	26	3	26	2
run 5	34	1	36	2	32	3
run 6	25	3	32	2	19	2
run 7	34	2	46	3	23	1
run 8	36	6	22	1	29	2
run 9	43	6	24	2	31	2
run 10	45	4	31	1	22	3
average	33.7	3.4	28.8	2.1	25.8	2

Table 6.3: Smart bots preset board

run number	two bots		three bots		four bots	
	turns	lead	turns	lead	turns	lead
run 1	29	3	28	4	23	2
run 2	30	4	35	2	31	3
run 3	45	3	23	3	45	3
run 4	24	2	46	1	36	1
run 5	42	1	26	2	19	2
run 6	21	4	26	3	25	3
run 7	32	3	44	2	22	1
run 8	43	2	36	4	21	2
run 9	46	4	21	1	25	3
run 10	32	2	19	2	26	2
average	34.4	2.8	30.4	2.6	27.3	2.2

Table 6.4: Smart bots random board

Results of this test suite show two things. First is that with just two players, Dummy bots have difficulties finishing the game. This is understandable because game is intended to be played in at least three people. Second point is that otherwise performance of bots is about the same across all the options with expected shortening of duration of game with more players, this is because of more trading possibilities and more frequent resource yield rolls.

6.2 Bot types

This section compares the two bot types against each other scaling their success in gameplay. In this case only randomised board will be used, previous section confirmed that the board type doesn't impact bot perfomance and randomised board test

run number	**Dummy bots vs Smart bots**			
	1 vs 1	**2 vs 1**	**1 vs 2**	**2 vs 2**
run 1	Dummy won	Smart won	Smart won	Smart won
run 2	Dummy won	Smart won	Dummy won	Smart won
run 3	Smart won	Dummy won	Smart won	Smart won
run 4	Dummy won	Dummy won	Smart won	Smart won
run 5	Dummy won	Smart won	Smart won	Dummy won
run 6	Dummy won	Smart won	Smart won	Smart won
run 7	Smart won	Dummy won	Smart won	Smart won
run 8	Dummy won	Smart won	Dummy won	Smart won
run 9	Dummy won	Smart won	Smart won	Smart won
run 10	Dummy won	Dummy won	Smart won	Smart won
run 11	Smart won	Dummy won	Smart won	Smart won
run 12	Dummy won	Dummy won	Smart won	Dummy won
run 13	Dummy won	Smart won	Smart won	Smart won
run 14	Dummy won	Smart won	Dummy won	Dummy won
run 15	Smart won	Smart won	Dummy won	Smart won
run 16	Dummy won	Smart won	Smart won	Dummy won
run 17	Dummy won	Dummy won	Dummy won	Dummy won
run 18	Dummy won	Smart won	Dummy won	Smart won
run 19	Smart won	Dummy won	Smart won	Dummy won
run 20	Dummy won	Smart won	Dummy won	Smart won
victory rate	15:5	8:12	7:13	6:14

Table 6.5: Dummy vs Smart random board

Results indicate that while dummy bot does better in two player game, presumably because there is very little room for more complex strategy or cooperation, smart bots begins to outperform them in three and four player games even when there is just one of them.

6.3 Bots versus player

This section compares bot performance against live player. To get better range of comparison, several volunteers were asked for assistance. Each player was set up against 3 bots of either type in five games.

Volunteers:

1. Male student, inexperienced player
2. Female student, inexperienced player
3. Male IT worker, experienced player
4. Male retiree, average player

run number	**player vs Dummy bots**		**player vs Smart bots**	
	game length	**player position**	**game length**	**player position**
run 1	36	2nd	29	3rd
run 2	25	3rd	19	2nd
run 3	32	1st	24	4th
run 4	29	1st	22	3rd
run 5	41	2nd	35	1st
average	32.6	1.8	25.8	2.6

Table 6.6: Live player 1

run number	**player vs Dummy bots**		**player vs Smart bots**	
	game length	**player position**	**game length**	**player position**
run 1	42	2nd	35	2nd
run 2	39	1st	27	2nd
run 3	51	3rd	29	1st
run 4	33	2nd	32	4th
run 5	24	3rd	19	3rd
average	37.8	2.2	28.4	2.4

Table 6.7: Live player 2

run number	player vs Dummy bots		player vs Smart bots	
	game length	player position	game length	player position
run 1	15	1st	19	3rd
run 2	22	1st	32	1st
run 3	23	2nd	27	1st
run 4	32	1st	25	1st
run 5	29	1nd	21	2nd
average	22.4	1.2	24.8	1.6

Table 6.8: Live player 3

run number	player vs Dummy bots		player vs Smart bots	
	game length	player position	game length	player position
run 1	26	2nd	32	2nd
run 2	42	3rd	35	2nd
run 3	21	1st	15	4th
run 4	32	2nd	41	3rd
run 5	46	2nd	24	1st
average	33.4	2.0	29.4	2.4

Table 6.9: Live player 4

Tests indicate that even the weaker version of the bot is a worthy opponent for inexperienced and average player. Experienced player can overcome it without much difficult but will usually tie with smart bot. Generally smart bot performs better than dummy bot and average game length with live player is comparable to that with bots only.

Chapter 7

Conclusion

This thesis introduced the game of the Settlers of Catan and its mechanics, summarised artificial intelligence theory relevant to player bot design and based on these information proposed several possibilities of specific bot implementation.

Putting these ideas in use necessitated creation of game framework and graphical user interface allowing for interaction between player and bots. This was done using Java (version SE8) with use of Swing toolkit.

One of the proposed bot architectures was implemented in two versions and their functionality was tested against each other and with live players. Tests confirmed most of the expectations - bots pose challenge for a casual player and can stand against experienced one. Majority of scenarios indicated superiority of bot with dynamic priorities, only expection being one on one games with static priority bot, where more straightforward approach had advantage.

7.1 Follow up

Findings of this thesis open up plenty of possibilities for expansions and following work. Graphical user interface of existing application can be further refined giving dynamic three dimensional view of game board and possibility of game playing over network can be added.

More importantly, besides approaches mentioned in this thesis there's still opportunity for application of more artificial intelligence methodologies such as machine learning or fuzzy logic allowing bots to analyse behavior of the other players and adjust their own strategy.

Naturally it is also possible to extend scope of the application beyond the core version of the Settlers of Catan and introduce some of many existing game expansions with their new mechanics.

Chapter 8

Appendix

Log sample - 4 player game with mixed type of bots, summary of previous turn plus complete run of observed turn

```
Board and resources after Turn 15:
          ~*~  (  )      (  ) ~O~  (  )
              0    0    0    2    0    0
          (  )      (  )      (  )      (  )
           0   L@6   0   B@9   2   L@8   0   ~B~
          (1S)      (1T)      (2S)      (2S)
         1    1    1    1    2    2    2    0
     (  )      (  )      (  )      (  )      (  )
 ~L~  0   O@4   0   G@6   0   B@5   0         0
     (  )      (4T)      (3S)      (3S)      (  )
    0    4    4    0    0    3    3    0    0    0
(  )      (  )      (  )      (  )      (  )      (  )
 0   G@3   4   O@12  0   L@9   0   W@2   0   O@3   0   ~*~
(  )      (4S)      (  )      (  )      (  )      (  )
    0    0    3    0    0    0    0    0    0    0
     (  )      (  )      (4T)      (1S)      (  )
 ~G~  0   G@5   3   W@10  4   BT8   0   W@11  0
     (  )      (3S)      (  )      (  )      (  )
         0    0    3    0    4    2    2    0
          (  )      (  )      (2S)      (2S)
           0   L@10  3   G@4   0   W@11  0   ~*~
          (  )      (  )      (  )      (  )
              0    0    0    0    0    0
          ~*~  (  )      (  ) ~W~  (  )

Owner    | Lumber | Brick  | Wool   | Grain  | Ore    |
Player 1 |    2   |    1   |    0   |    3   |    0   |
Player 2 |    2   |    0   |    0   |    2   |    4   |
Player 3 |    1   |    1   |    0   |    3   |    2   |
Player 4 |    1   |    1   |    0   |    1   |    0   |
Bank     |   13   |   16   |   19   |   10   |   13   |
```

```
Player 1. (Smart Bot) Turn 16
Player 1. (Smart Bot) rolled 2 + 3 = 5
Player 1. (Smart Bot) gained: nothing
Player 2. (Dummy Bot) gained: 1 Brick
Player 3. (Dummy Bot) gained: 2 Brick 1 Grain
Player 4. (Smart Bot) gained: 1 Grain
Player 1. (Smart Bot) build new Road at [3,0] and now control longest road,
earning 2 points.
Player 1. (Smart Bot) wants: 1 Brick offers: 1 Grain
Player 2. (Dummy Bot) refuses
Player 3. (Dummy Bot) refuses
Player 4. (Smart Bot) refuses
Player 2. (Dummy Bot) Turn 16
Player 2. (Dummy Bot) rolled 1 + 3 = 4
Player 2. (Dummy Bot) gained: 1 Grain
Player 3. (Dummy Bot) gained: nothing
Player 4. (Smart Bot) gained: 2 Ore
Player 1. (Smart Bot) gained: 1 Ore
Player 2. (Dummy Bot) upgraded Settlement at [1,5] into Town, earning 1 point.
Player 2. (Dummy Bot) wants: 1 Wool offers: 1 Ore
Player 3. (Dummy Bot) refuses
Player 4. (Smart Bot) refuses
Player 1. (Smart Bot) refuses
Player 3. (Dummy Bot) Turn 16
Player 3. (Dummy Bot) rolled 6 + 5 = 11
Player 3. (Dummy Bot) gained: nothing
Player 4. (Smart Bot) gained: nothing
Player 1. (Smart Bot) gained: 1 Wool
Player 2. (Dummy Bot) gained: 3 Wool
Player 3. (Dummy Bot) wants: 1 Ore offers: 1 Grain
Player 4. (Smart Bot) refuses
Player 1. (Smart Bot) refuses
Player 2. (Dummy Bot) refuses
Player 4. (Smart Bot) Turn 16
Player 4. (Smart Bot) rolled 1 + 2 = 3
Player 4. (Smart Bot) gained: 1 Grain
Player 1. (Smart Bot) gained: nothing
Player 2. (Dummy Bot) gained: nothing
Player 3. (Dummy Bot) gained: nothing
Player 4. (Smart Bot) wants: 1 Wool offers: 1 Grain
Player 1. (Smart Bot) refuses
Player 2. (Dummy Bot) refuses
Player 3. (Dummy Bot) refuses
```

Board and resources after Turn 16:

```
          ~*~  (  )      (  ) ~O~  (  )
              0    0    0    2    0    0
          (  )      (  )      (  )      (  )
           0   L@6   0   B@9   2   L@8   0   ~B~
          (1S)      (1T)      (2T)      (2S)
         1    1    1    1    2    2    2    0
     (  )      (  )      (  )      (  )      (  )
 ~L~  1   O@4   0   G@6   0   B@5   0         0
     (  )      (4T)      (3S)      (3S)      (  )
    0    4    4    0    0    3    3    0    0    0
(  )      (  )      (  )      (  )      (  )      (  )
 0   G@3   4   O@12  0   L@9   0   W@2   0   O@3   0   ~*~
(  )      (4S)      (  )      (  )      (  )      (  )
    0    0    3    0    0    0    0    0    0    0
     (  )      (  )      (4T)      (1S)      (  )
 ~G~  0   G@5   3   W@10  4   BT8   0   W@11  0
     (  )      (3S)      (  )      (  )      (  )
         0    0    3    0    4    2    2    0
          (  )      (  )      (2S)      (2S)
           0   L@10  3   G@4   0   W@11  0   ~*~
          (  )      (  )      (  )      (  )
              0    0    0    0    0    0
          ~*~  (  )      (  ) ~W~  (  )
```

Owner	Lumber	Brick	Wool	Grain	Ore
Player 1	1	0	1	3	1
Player 2	2	1	3	1	1
Player 3	1	3	0	4	2
Player 4	1	1	0	3	2
Bank	14	14	15	8	13

Bibliography

[1] Board Game Geeks community. Eurogame entry. `http://www.boardgamegeek.com/wiki/page/eurogame`, January 2015. seen 20. Jan 2015.

[2] Association for the Advancement of Affective Computing. Emotion annotation and representation language. `http://emotion-research.net/projects/humaine/earl`, April 2015. seen 23. Apr 2015.

[3] Feng-hsiung Hsu. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002. ISBN 0-691-09065-3.

[4] V. Mařík, O. Štěpánková, J. Lažanský, and col. *Umělá inteligence (2)*. Academia, Praha, 1997. ISBN 80-200-0504-8.

[5] Ervin Melkó and Benedek Nagy. Optimal strategy in games with chance nodes. *Acta Cybernetica*, 18(2):171–192, 2007.

[6] Reinhold Ploesch. *Contracts, Scenarios and Prototypes: An Integrated Approach to High Quality Software*. Springer, 2004 edition, 2004.

[7] S. Polberg, M. Paprzycki, and M. Ganzha. Developing intelligent bots for the diplomacy game. *Computer Science and Information Systems*, pages 589–596, 2011.

[8] Haseeb Saleem and Rashdan Raees Natiq. A multi-agent player for settlers of catan. Master thesis, Blekinge Institute of Technology, 2008.

[9] E. Triantaphyllou. *Multi-Criteria Decision Making: A Comparative Study*. Dordrecht, The Netherlands: Kluwer Academic Publishers (now Springer), 2000. ISBN 0-7923-6607-7.

[10] Red Blob Games tutorials. Hexagonal grids entry. `http://www.redblobgames.com/grids/hexagons/`, April 2015. seen 12. Apr 2015.