Collaborative Strategies for Distributed Problem Solving

A Treasure Hunt Game with Intelligent Communicating Agents

Bachelorarbeit

im Studiengang Angewandte Informatik
der Fakultät Wirtschaftsinformatik
und Angewandte Informatik
der Otto-Friedrich-Universität Bamberg

Verfasser: Tina Kämmerer

Gutachter: Professor Dr. Ute Schmid
# Table of contents

1. Introduction  

2. A Treasure Hunt Scenario  

3. Intelligent Agents and Multi Agent Systems  
   3.1. Agents  
      3.1.1. Attributes of Intelligent Agents  
      3.1.2. Types of agents  
      3.1.3. The Role of Environment  
   3.2. Collaborative Distributed Problem Solving  
      3.2.1. Multiagent Systems  
      3.2.2. Coordination, Cooperation and Collaboration  
      3.2.3. Types of Interaction  
      3.2.4. Communication  
      3.2.5. Cooperative Problem Solving Process  
         3.2.5.1. Task Sharing and Result Sharing  
         3.2.5.2. Planning and Synchronisation  
   3.3. Decision Making in Multiagent Systems  
      3.3.1. Voting  
      3.3.2. Auctions  
         3.3.2.1. English auctions  
         3.3.2.2. Japanese auctions  
         3.3.2.3. Dutch auctions  
         3.3.2.4. First-price Sealed-bid auctions  
         3.3.2.5. Vickrey auctions  
         3.3.2.6. Multiunit auctions  

4. Problem Solving and other Fields of Research  
   4.1. Cognitive Psychology and Problem Solving  
   4.2. Organisational Psychology and Group Dynamics  
      4.2.1. Theoretical basics  
      4.2.2. Problem solving and decision-making in groups  
      4.2.3. Advantages and disadvantages of groups  
      4.2.4. Communication Structure  

5. Treasure Hunt Game  
   5.1. Game Setup  
   5.2. World  
   5.3. Information Display  
   5.4. Actors  
      5.4.1. Agents
## Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.2. Objects</td>
<td>34</td>
</tr>
<tr>
<td><strong>6. Implementation and Evaluation</strong></td>
<td>35</td>
</tr>
<tr>
<td>6.1.1. LibGDX</td>
<td>36</td>
</tr>
<tr>
<td>6.1.2. Textures and Sprites</td>
<td>37</td>
</tr>
<tr>
<td>6.1.3. Tiled</td>
<td>37</td>
</tr>
<tr>
<td>6.1.4. Additional Tools and Sources</td>
<td>38</td>
</tr>
<tr>
<td>6.2. Game Design</td>
<td>39</td>
</tr>
<tr>
<td>6.3. Turn-based Strategy Games</td>
<td>40</td>
</tr>
<tr>
<td>6.4. Implementation and Code</td>
<td>40</td>
</tr>
<tr>
<td>6.4.1. Player-controlled Gameplay</td>
<td>42</td>
</tr>
<tr>
<td>6.4.2. Autonomous Gameplay</td>
<td>43</td>
</tr>
<tr>
<td>6.4.3. Further Improvements</td>
<td>44</td>
</tr>
<tr>
<td>6.5. SuperAgentHuman versus MultiAgentHuman versus SuperAgentAutonomous versus MultiAgentAutonomous</td>
<td>46</td>
</tr>
<tr>
<td><strong>7. Conclusion</strong></td>
<td>50</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>51</td>
</tr>
<tr>
<td><strong>A. Java-Quellcodes</strong></td>
<td>54</td>
</tr>
<tr>
<td>A.1. Animation</td>
<td>54</td>
</tr>
<tr>
<td><strong>B. Features of LibGDX</strong></td>
<td>56</td>
</tr>
<tr>
<td>B.1. Graphics</td>
<td>56</td>
</tr>
<tr>
<td>B.2. Utilities</td>
<td>57</td>
</tr>
<tr>
<td>B.3. Tools</td>
<td>57</td>
</tr>
<tr>
<td>B.4. Audio</td>
<td>57</td>
</tr>
<tr>
<td>B.5. Input Handling</td>
<td>57</td>
</tr>
<tr>
<td>B.6. Math &amp; Physics</td>
<td>58</td>
</tr>
<tr>
<td>B.7. File I/O &amp; Storage</td>
<td>58</td>
</tr>
<tr>
<td><strong>C. Sounds</strong></td>
<td>59</td>
</tr>
<tr>
<td><strong>D. Contents of DVD</strong></td>
<td>60</td>
</tr>
</tbody>
</table>
# List of Figures

3.1. Communication between agents and environment, see [RN12, Page 61] . . . 5  
3.2. Simple reflex agent, see [RN12, Page 77] . . . . . . . . . . . . . . . . . . . 6  
3.3. Model-based reflex agent, see [RN12, Page 79] . . . . . . . . . . . . . . . . 7  
3.4. Goal-based agent, see [RN12, Page 80] . . . . . . . . . . . . . . . . . . . . 7  
3.5. Utility-based agent, see [RN12, Page 82] . . . . . . . . . . . . . . . . . . . . 8  
3.6. Learning agent, see [RN12, Page 83] . . . . . . . . . . . . . . . . . . . . . 8  

4.1. Formal communication structures . . . . . . . . . . . . . . . . . . . . . . . 27  

5.1. Screenshot of the MenuScreen . . . . . . . . . . . . . . . . . . . . . . . . . 29  
5.2. Screenshot the GameScreen . . . . . . . . . . . . . . . . . . . . . . . . . . 30  
5.3. Agents of the Treasure Hunt Game . . . . . . . . . . . . . . . . . . . . . . 32  
5.4. Objects of the Treasure Hunt Game . . . . . . . . . . . . . . . . . . . . . . 34  

6.1. Screenshot of Tiled . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37  
6.2. Tilesets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38  
6.3. Example of a sprite sheet . . . . . . . . . . . . . . . . . . . . . . . . . . . . 42  
6.4. Adding information to tiles . . . . . . . . . . . . . . . . . . . . . . . . . . . 45  
6.5. Evaluation of the game types . . . . . . . . . . . . . . . . . . . . . . . . . . 47
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.</td>
<td>Skills of non-specialised agents</td>
<td>32</td>
</tr>
<tr>
<td>5.2.</td>
<td>Skills of Explorer</td>
<td>32</td>
</tr>
<tr>
<td>5.3.</td>
<td>Skills of Explorer</td>
<td>33</td>
</tr>
<tr>
<td>5.4.</td>
<td>Skills of Knight</td>
<td>33</td>
</tr>
<tr>
<td>5.5.</td>
<td>Skills of Explorer</td>
<td>33</td>
</tr>
<tr>
<td>5.6.</td>
<td>Skills of SuperAgent</td>
<td>34</td>
</tr>
<tr>
<td>6.1.</td>
<td>System Configuration</td>
<td>40</td>
</tr>
<tr>
<td>6.2.</td>
<td>Evaluation of the game types</td>
<td>47</td>
</tr>
<tr>
<td>6.3.</td>
<td>Analysis of the results</td>
<td>48</td>
</tr>
</tbody>
</table>
Chapter 1. Introduction

1. Introduction

Nowadays, multiagent systems can be encountered on a daily basis: Webcrawlers analyse the WWW to provide up-to-date information, other multiagent systems are responsible for logistics and production planning. They are deployed in aerial and rail transportation as well as space exploration, and we also encounter them in computer games or the RoboCup.

Multiagent systems exist even in nature: Fish swarms, bird flocks, ant colonies or wolf packs, to name but a few, are also multiagent systems.

Independently of the area of application, all multiagent systems face the same challenges: They have to find collaborative strategies for coordinating their agents and they have to find solutions for problems in order to fulfil their intended objective.

After shortly introducing the Treasure Hunt game accompanying this bachelor thesis, an overview over the topic intelligent agents will be provided. Following this, concepts and methods of collaborative distributed problem solving and multiagent systems will be explained, focusing on types of interaction, communication between agents and the process of decision making.

The next chapter addresses the topic of collaborative problem solving in other fields of research, especially the cognitive and organisational psychology.

Ultimately, the Treasure Hunt game will be discussed further, focusing on the used software, implementation and, most importantly, the evaluation of the game in relation to the performance of a multiagent system versus a expert agent.
2. A Treasure Hunt Scenario

Accompanying the bachelor thesis, a short strategy game was programmed that features either a team of agents or a single agent that hunt for a treasure. The Treasure Hunt game plot follows a previously defined sequence that must be executed mostly step by step to reach the goal of the game: acquiring the treasure that is hidden in the TreasureChest. The game plot will now be presented briefly to enable the understanding of later references on the game. Roughly divided, there are ten separate sections of the plot:

1. At the start of the game, most parts of the playing field are obscured and the agents have to explore the map to find important locations and objects to interact with.

2. The next step is to cut down the four discovered trees on the map to collect WoodenBoards which are required for the following action.

3. The Knight can now build a bridge across the river, thus uncovering new territory that was not accessible before as well as a cave.

4. To make an exploration of the cave possible, an agent has to find the CavePlan that is hidden in a bush.

5. Now the cave can be explored by the Explorer which subsequently finds a SummoningStone within.

6. Using this SummoningStone, the Wizard can now summon the FireImp at the SummoningShrine.

7. The FireImp is capable to retrieve the Sword which lies within a lava lake.

8. In the next step, the Knight can kill the Dragon by using the Sword. The Dragon now drops a map of KeyFragments.

9. Agents dig out the six KeyFragments which are as a result combined to create a key.

10. By using the key, the TreasureChest can be opened and the player wins the game by retrieving the treasure.

It is possible for some of these steps to blend into each other, for instance while some agents are occupied with producing wooden boards, another agent that is currently unemployed can progress in the plot by collecting the CavePlan which is only required when the bridge is build and the cave is detected.
3. Intelligent Agents and Multi Agent Systems

3.1. Agents

Multiagent systems are constructs of two or more agents that work together to reach a common goal which, under some circumstances, they may be not able to achieve on their own.

Before going into detail of how multiagent systems are defined and how the coordination of a group of agents works, it is important to clarify the term agent.

The precise definition of the term agents is a difficult endeavour, because based on its importance in a number of fields of research, there exists no overall consensus on how exactly the term agent is defined, with the exception that autonomy is a crucial aspect for agents. Nevertheless, there are various approaches to define what constitutes an agent, some of them will be presented in the following.

Russell and Norvig explain agents as following:

"An agent is just something that acts (…). Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals." [RN10, Page 4]

Wooldridge describes an agent as

"a computer system that is capable of independent action on behalf of its user or owner. In other words, an agent can figure out for itself what it needs to do in order to satisfy its design objectives, rather than having to be told explicitly what to do at any given moment." [Woo09, Page 5]

and define it as

"a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives." [Woo09, Page 21]

3.1.1. Attributes of Intelligent Agents

According to [WJ95, Page 4] and [Pad05, Page 114], agents have the following attributes:

- **Social ability**
  Agents must have the ability to communicate with other agents and humans via some kind of language and based on a communication structure. In human-computer
interaction it is particularly important to represent information intuitively and comprehensibly to facilitate human understanding. For agent-to-agent communication required in multiagent systems, it is often particularly important to transmit complete information in a rather timely fashion to enable other agents to quickly respond to a changing environment.

- **Reactivity and adaptability**
  Depending on their observation of the environment, agents have to react to and adapt to occurring changes like movements of other agents or changes in their environment.

- **Pro-activity and goal-directedness**
  It is important that the agent follows a goal-directed behaviour, trying to fulfil their task and simultaneously including changes in the environment that have an influence on this process. In multiagent systems it is also important to balance conflicting goal between different agents that each try to achieve their objective.

- **Autonomy**
  Agents are not dependent on human instructions, they decide on and control their actions based on their environment and internal state and coordinate them with other agents if they are a part of a multiagent system.

### 3.1.2. Types of agents

In principle, all agents consist of sensors that enable them to recognise their environment, and actuators, through which they take influence on their surroundings [RN12, Page 60], see figure 3.1.

![Figure 3.1.: Communication between agents and environment, see [RN12, Page 61]](image_url)

Agents observe their environment through sensors, afterwards analyse the information gained through this process and deciding on actions that they perform through the actuators. Depending on the type of agents, the sensors and actuators are different. A software agent receives input in the form of user input or data files and their actions consist of displaying, writing or transmitting information.

A robot agent however has an entirely different set of sensors and actuators. It can perceive its environment through a variety of sensors: sensors for recognising touch, colour, temperature, infrared radiation, brightness, movement or the chemical environment. Possible actuators are motors for movements or constructs that enable them to interact with humans through gestures, collect samples or move obstacles.
According to [RN12, Page 75 et seq.], five separate classes of agents can be distinguished based on their way of processing input from the environment:

- **Simple reflex agents**
  Simple reflex agents calculate their next actions based on current sensor information, previously executed actions or states prior to the actual state are not included in the decision-making process. Although advantageous in many situations, simple reflex agents are not particularly intelligent as they can choose suboptimal actions for a problem and are often caught in infinite loops. This can be prevented by enabling the agent to randomly select actions when caught in an infinite loop. While principally only working in fully observable environments, this feature makes it also possible to function in partially unobservable environments.

  Simple reflex agents act according to simple if-else rules, exemplified by the Treasure Hunt game:

  ```
  If (Tree is discovered) Then
    (Plan path to Tree and cut it down)
  
  If (Bush is discovered) Then
    (Plan path to Bush and retrieve CavePlan)
  
  If (Sword is in inventar) Then
    (Plan path to Dragon and slay it)
  ```

  ![](image)

  **Figure 3.2.: Simple reflex agent, see [RN12, Page 77]**

- **Model-based reflex agents**
  The model-based reflex agent is a improved simple reflex agent, relying not only on sensor data but also additional information, e.g. the development of the environment and information about how own actions influence this environment, an thus able to function is partially unobservable environments. This agents actions are calculated based on sensor data and information about the current state of itself and its environments. Therefore, is has a memory that contains prior actions and states.

  ```
  If (Sword is in inventar) Then
    (Plan path to Dragon and slay it)
  ```

  effect of this action on state: sword is in inventar
Goal-based agents

Goal-based agents expand the model-based reflex agent further by adding a supplementary component to their decision-making process. Before deciding on an action, it also calculates the effect this action has on the process of reaching their intended goal, selecting the one that takes them closer to their objective. The goal-based agent is more efficient compared to the model-based reflex agent or even the simple reflex agent, because it can change its knowledge about situations and modify its behaviour to suit the updated conditions.

Utility-based agents

A utility-based agent assigns utility to actions, e.g. the value this action has for the process compared to other possible actions. Based on this internal utility function, the agent has advantages over the goal-based agent: it can compromise between conflicting goals, deciding for an approach that maximises the expected gain and it can balance the possibility of the success of an action against the importance of reaching a certain goal.
3.1.3. Learning agents
Learning agents are especially suited for unknown environments, as they are able to function in these environments, discover new knowledge about it and integrate this knowledge into its decision-making process. This internal decision-making process is divided into four collaboration components.
The learning element has the goal to make improvements on existing processes and structures whereas the performance element has the task to choose external actions. The critic passes feedback about the effects of actions on to the learning element, which decides on modifications on the performance element to improve the overall performance for the following actions. The problem generator generates new actions that could, while being suboptimal in the short term, result in superior actions in the long term.

3.1.3. The Role of Environment
The class of agents that employed for a task is dependant on their working environment. This working environment can be categorised by different dimensions, cf. [RN12, Page 69 et seq.], illustrated by examples from the Treasure Hunt RPG:

• Fully observable versus partially observable
To be fully observable, a task environment has to fulfil several conditions. Firstly, it has to be completely observable by the agent’s sensors at any given moment. The sensors must be able to identify all aspects that are relevant for the selection of an
action. A partially observable environment is the case if the sensors are not precise enough, if there are interferences or if particular information can not be perceived by the sensors.

It is also possible for an environment to be not observable, which requires very adaptable and flexible agents.

The Treasure Hunt game takes place in a virtual environment rather than a physical environment, and therefore the agents possess no real sensors, they also perceive their environment, and, based on this observation, decide on their actions. The game’s environment is a mixture of a fully observable and a partially observable one. Agents are aware of their surrounding areas that are important for determining their actions, as well as, in case of multiple agents, the positions and discoveries of other agents. Simultaneously, there also exist areas that are unknown to all agents, which have to be discovered or made accessible first.

**Single agents versus multiagent**

At first sight, it might appear to be relatively simple to distinguish between single agent and multiagent environments. To identify exactly whether an environment is a single agent or a multiagent one, one has to define which entities have to be considered as agents. Entities that are characterised by trying to maximise the value of their performance measure, which is influenced by another agents actions, are viewed as agents.

This relation between can manifest in two distinct forms, which can, however, also overlap in some instances. A competitive environment is the case if both agents are trying to maximise their own performance measure while simultaneously trying to minimise the other agent’s performance measure.

One speaks of a cooperative multagent environment, if the action of one agent does not only increase their own performance measure but also affects that of the other agents positively.

Again applied to the game, a competitive environment would be possible by adding a rival team B to the existing team A, which also strives to retrieve the treasure. Team B would then try to find the treasure first, by blocking actions from A or gaining an advantage on them by discovering hints faster.

The existing game environment is a cooperative one, that also has some competitive characteristics. Since the agents work together as a group to achieve their common goal, the environment is cooperative. The action of one agent increases the performance measure of all agents, bringing them a step closer to their goal. The competitive component is demonstrated if two agents have the possibility to perform the same action, e.g. cutting down a tree, at a particular place, thus trying to improving their respective performance measure. Since only one of the agents can be at a specific place at the same time, they are in competition with each other. In this case, the agent that can perform this action the fastest, calculated by factoring in their distance to the place and the amount of turns it takes to accomplish the task, is awarded the task.

**Deterministic versus stochastic**

An environment is deterministic if, with the inclusion of performed actions from agents and the actual state, the next state can be determined. If that is not the
case, it is a stochastic environment. If it is not deterministic or fully observable, an environment is called uncertain.

The implemented RPG has a deterministic environment, because depending on the actual state, the agents actions can be calculated and performed, leading to next state.

- **Episodic v. sequential**
  
  A episodic environment is characterised by the possibility to separate the agents experience into episodes, which have the attribute that they are not dependent on episodes prior to the actual one. An example for a episodic environment are classification tasks.
  
  A environment is called sequential, if previously executed actions have an impact on following decisions.

  The game implements a sequential environment, because the different combination of actions that were performed up to the current state determine the next actions. If, for example, a map that is required for exploring a cave was retrieved in the past turn, the agents can explore it in this turn. Was the map not retrieved because the site of the find was not yet discovered or the agents have not yet reached it, the actions performed in this turn diverge from the above.

- **Static versus dynamic**
  
  Static and dynamic environments are distinguished by whether the environment changes when an agent is deciding on an action. Is this the case, the environment is a dynamic one. Dynamic environments are more difficult to cope with for the agents, because they have to be monitored constantly and a sudden change can affect the implementation and effects of the chosen action. The confronted with a static environment is comparatively simple for the agent. The selection of actions is does not have to include a changing environment and is also not time-sensitive.

  Based on the fact that the Treasure Hunt game is a turn-based game rather than a real-time game, the selection of subsequent actions takes place in static environment, thus facilitating the planning. At the start of each turn, the environment is clearly defined in its status and only changes if each agent has determined its action. In a real-time game, agents would have to observe the environment for changes during their process of finding the next action to compensate for changes which would make the considered action impossible or unfavourable. For further details on the topic turn-based versus real-time games, see section **Turn-based Strategy Games**.

- **Discrete versus continuous**
  
  A discrete environment is one that has a finite number of states, perceptions and actions, whereas a continuous environment is permanently changing its status over time.

  A turn-based game always implements a discrete environment: the progression of time is clearly divided into specific sections, thus not following a continuous flow of time. In each section, the state of the environment as well as the perception and actions of agents are clearly defined. Real-time games have a continuous environment,
although not entirely. While it tries simulate a real and natural flow of time, there is a clear distinction between a virtual simulated game environment and a real one: A game environment has to have discrete traits, fixed time slots to compute and coordinate the movements of all agents and avoid collision between agents whereas that is not the case within a real environment. This difference between turn-based/discrete games and real-time/mainly continuous games is also addressed in section Turn-based Strategy Games.

- **Known versus unknown**
  Known and unknown refer to the knowledge of the agents in respect to the fundamental rules of the environment. Within a known environment, the agent is informed about actions and their effects, while agents in an unknown environment have to learn about the outcomes of their actions.

  Applied to my example, a distinction must be made between the agents and the player: Whereas the agents operate with a known environment, being aware of objects and their purpose, the player is confronted with an unknown environment: They first have to discover all agents and objects, and have to learn about their respective purpose for the game.

To summarise it, the Treasure Hunt RPG is implementing a semi-observable, deterministic, sequential, dynamic, discrete, and, depending on the chosen game style, single agent or multiagent as well as known and unknown environment.

Depending on the environment that the agents are confronted with, there are various possibilities for constructing agents to best suit this environment by adjusting their internal implementation and behaviour, the communication structure and the type of control. This considerations will be further outlined in the following sections.

### 3.2. Collaborative Distributed Problem Solving

Cooperative Distributed Problem Solving is a field of research that focuses on the control and organisation of multiagent systems. In a multiagent system, multiple agents are combined in a group, which can pose a problem: Each agent has their own set of goals and subgoals, follow their own strategy and have their specialised set of skills. Therefore, combining this single agents to a well-functioning multiagent system, there has to be a mechanism to coordinate actions and goals.

The computer scientist Lesser was one of the first researchers that introduced the concept of cooperative distributed problem solving and defined it as following:

"Cooperative Distributed Problem Solving (CDPS) studies how a loosely coupled network of problem solvers can work together to solve problems that are beyond their individual capabilities. Each problem solving node in the network is capable of sophisticated problem solving and can work independently, but the problems faced by the nodes cannot be completed without cooperation. Cooperation is necessary because no single node has sufficient expertise, resources and information to solve a problem, and different nodes might have expertise for solving different parts of the problem." [DLC95, Page 63]
Thus, CDPS is aimed at addressing problems that single agents cannot solve at their own. This would be the case for e.g. physically distributed problems or problems that require skills that the single agent does not possess. Within a CDPS, there are several issues that have to be considered regarding the types of interaction between agents, the communication between agents as well as the decision making process that determines the actions of the agents.

### 3.2.1. Multiagent Systems

Other than CDPS, where the existence of benevolence is assumed, agents within a multiagent system are self-interested and it is possible that they have goals that contradict goal of other agents.

Multiagent systems are systems with more than one agent. These agents communicate and cooperate to reach a common goal that they cannot achieve on their own. To accomplish this, multiagent systems must address several issues: Agents must be capable to coordinate their skills and actions, to exchange information and to determine a common goal, that can also be divided into subtasks, by resolving existing conflicts between the individual goals.

Whether or not agents cooperate can be determined by two conditions [Fer01, Page 97]: Agents cooperate if, by adding an additional agent to the group, the group performance increases or if actions of agents can prevent or dissolve potential or current conflicts.

To evaluate the performance of an existing MAS, two aspects have to be considered [Woo09, Page 153]: The coherence determines the behaviour of the system according according to certain evaluation criteria. That could be the solution quality, resource usage or the performance of the MAS when encountering unexpected errors. Coordinates evaluates the interaction between agents, e.g. the communication and the resolution of conflicts.

### 3.2.2. Coordination, Cooperation and Collaboration

There are several ways of working together that have to be distinguished [SAA+06, Page 172]: Coordination, Cooperation and Collaboration. Interaction between individuals is called **Coordination**, if they are sharing the same space, but are not working together in the sense of working on the same tasks. To prevent interferences, coordination requires a reconcilement of time and place, but not a change to the behaviour of the distinct groups.

**Cooperation** implies that subgroups have to adapt their behaviour to avoid conflicts in time or space, whereas **Collaborations** means the work on a same task or project.

### 3.2.3. Types of Interaction

If multiple agent are in the situation of being situated in a multiagent system, the cooperation of agents can have multiple dimensions. There are different types of interaction that can occur between agents, depending on the goals, resources and skills of each agent, cf. [Fer01, Page 90 et seq.]:

- **Autonomy**
  Scenarios where agents have compatible goals, sufficient resources and skills to reach
their goal are autonomous interactions. Although agents are in physical proximity, they only work alongside one another but do not interact with each other. Thus, this scenarios are not particularly important for the process of distributed problem solving.

• **Simple Collaboration**
Simple collaboration between agents is the case, if there are compatible goals, sufficient resources but the agents lack the means to achieve their goals on their own. Its main distinction from coordinated collaboration is that the agents simply add their skills together but do not have the need for further coordination. This would be the case if, for example, two agents in a multiagent system that specialises in exploration would exchange map information: They collaborate, but there is no need for them to coordinate their actions.

• **Blockade**
If two or more agents have compatible goals and sufficient skills to achieve them, but there is a shortage of available resources, they are in a blockade. In a blockade scenario there are more than one agent that prevent each other from achieving a goal or subgoal, e.g. if they both need to be at the same place to fulfil their action. If in an automated storage systems two agents have to be on the same position because one want to retrieve an item and the other wants to store new items in this place, they would block each other from executing their tasks. Multiagent systems require planners that coordinate actions and therefore prevent scenarios like this that delay the process.

• **Coordinated collaboration**
Coordinated collaboration is required in a scenario where agents have compatible goals but insufficient resources and skills and therefore need to combine their skills to achieve a goal. Illustrated again at the example of a multiagent system used for exploring: Agent 1 is confronted with an obstacle that they can not remove on their own. The area behind the obstacle has to explored to reach the goal. Subsequently, Agent 1 and Agent 2 which is also interested in this area work together to remove the obstacle and to clear the path.

• **Purely individual competition**
Purely individual competition is the case if goals are incompatible but each agent has sufficient resources and skills to achieve them. A good example for this scenario would be the Treasure Hunt game with a SuperAgent if a additional feature was introduced: an opponent that competes with the player for the treasure. The outcome of this competitive situation would depend on the skills of the agents, provided that they each have the same initial situation. The agent with the better or more adjusted skill set would win the competition. It is important that there are sufficient resources, e.g. two swords on the map to kill the two dragons.

• **Purely collective competition**
The purely collective competition differs from the purely individual one in the aspect that a agent does not have the required skill set to achieve its goal. Therefore, agents assemble themselves to different groups which then compete against each other to reach their respective goals. This situation can also be described by the Treasure
Hunt game in a multiagent setting: by not only introducing a single opponent to the game but a set of multiple agents, two rival groups would emerge, because no agent has the skill set to achieve the goal on their own. This rival groups would now stand against each other in their objective to retrieve the only available treasure. Like in the purely individual competition, the resources on the map are in duplicate.

- **Individual and collective resource conflict**
  This scenarios emerge if there are not enough resources for both opponents in a purely individual or a purely collective competition. Thus, the game would degenerate into a simple race to the resources, trying to retrieve them before the opponent or otherwise having no chance to achieve the goal.

### 3.2.4. Communication

A reliable communication structure is of particular importance for distributed multiagent systems. The consideration about when to communicate which information has a great relevance on the functionality and success of multiagent networks. The network topology specifies the process of communication and the susceptibility to errors in form of malfunctions.

Communication between agents is either realised by message passing, e.g. one agent sends information only to certain other agents, either because the information concerns only them, or because they do not want other agents to be aware of this information. If information should be available to all agents, then a shared memory is required where each agent updates their information, which is subsequently accessible to all other agents. There are different possibilities for the design of the network topology, cf. [ZLR07]:

- **Centralised agent network topology**
  All communication runs through one central agent and other agents are only connected through this central element, not directly with each other. This topology does however become problematic if the central agents becomes inoperative, because in this case the whole network crashes and not further communication is possible.

- **Peer-to-Peer agent network topology**
  In a peer-to-peer network, each agent is connected with each other agent via separate communication lines. This is one of the most fail-safe networks, even if one or more connections malfunction, there is a high probability that the remaining connections can compensate it.

- **Broadcasting agent network topology**
  In this topology, all agents communicate through a shared media, broadcasting their message and receiving information from other agents.

- **Closed-loop agent network topology**
  The agents are arranged in a ring structure and each agent has a connection to its two neighbours. This structure has a high susceptibility to errors: By removing one link the structure resembles a linear agent network topology, but if two or more connections malfunction, the network disintegrates into several isolated subnetworks, thus preventing collaborative communications and actions.
• **Linear agent network topology**
  This network is quite inefficient for most applications, because to transmit a message from agent A to agent D, it has to be passed through all intermediate agents. However, it is very useful for processing pipelines, where each agent builds on the results of the preceding agent.

• **Hierarchical agent network topology**
  Within a hierarchical network, agents transmit messages in subgroups, only connected to other groups by upper level agents.

According to [Fer01, Page 334 et seq.] and based on the communication model of linguist Roman Jacobsen, the forms of communication can be divided into six categories according to their function. While some of these forms are more relevant to interpersonal communication, quite a few can also be applied to communication between agents or systems:

• The **Expressive function** describes the status of the sender, e.g. its state, goals and perceptions. By sending this information to other agents in the network and synchronising them, it is possible to create a basis for coordination of actions and exchange of beliefs within a multiagent system. If one agent A in the RPG discovers a new object that another agent B or C in the team can interact with, A can send this discovery to all other agents. When B is occupied performing an action for the next two turns, it can also communicate this status, thus enabling C to take the task.

• The **Conative function** of communication is used to transmit commands from the sender to the receiver and is the most commonly used form within multiagent systems.

• The **Referential function** is contextual and is used to transmit data about the environment and other agents. An agent can communicate its current model of the world to other agents, thus enabling an exchange and comparison of models, finding and resolving discrepancies.

• The **Phatic function** is used for establishing, continuing and pausing communication between agents and systems. A good example for this is the three-way handshake used to establish TCP/IP connection to make sure that the communication channel between two instances is established and working.

• The **Poetic function** is used to emphasise messages, but is so far rarely used by multiagent systems.

• The **Metalingual function** is very important for multiagent systems, enabling the agents to send messages about other messages, therefore creating a foundation for synchronizing syntax, terms and concepts between agents.

### 3.2.5. Cooperative Problem Solving Process

The cooperative problem solving process follows an idealised four-stage model, cf. [WJ99]. The **Recognition Stage** is initiated by an agent’s incentive to cooperate with other agents for reaching its goal. This is the case if this agents either is aware of the fact that
it can only reach its goal with the help of other agents that have the appropriate abilities, or if the agent could reach the goal on its own, but would rather cooperate to accelerate or improve this process.

As soon as this stage is completed, the **Team Formation Stage** starts, where the agents makes contacts with other agents it wants to recruit into its team. This attempts may be unsuccessful, but if the agent prevails a team of agents if formed that have a nominal commitment to their shared goal.

After this step, the **Plan Formation Stage** begins, if the prior stage resulted in a team formation success. All actions that are known to the different team members are introduced to all agents and following this, a collaborative plan will be negotiated by considering the respective objections and proposals.

Assuming that the negotiation succeeded and a common plan was constructed, the **Team Action Stage** follows. In this stage, the jointly developed plan is executed.

### 3.2.5.1. Task Sharing and Result Sharing

To solve a distributed problem collaboratively by splitting the common problem into sub-tasks, there are several steps that have to be implemented, cf. [Woo09, Page 154 et seq.]. Firstly, a problem decomposition has to take place. The problem is recursively split into progressively smaller subproblems until this subproblems have the appropriate scope for being processed by agents.

After finalising this step, the subproblem solution follows, where all subproblems created during the problem decomposition process are identified. With the results of this step, the solution synthesis can take place, where all solutions for the subproblems are combined to a general solution.

Applying that procedure to the example of the Treasure Hunt game, the common goal is to retrieve the treasure. Based on this, there are several sets of subproblems, e.g. retrieving the keyFragments, finding the TreasureChest etc. This sets would produce several subsolutions, e.g. finding the keyFragments and plan the path to them as well as exploring the map to uncover the position of the TreasureChest. This subsolutions will then be integrated into the overall game plan.

Required for this process are **Task Sharing** and **Result Sharing**. Task sharing is applied for allocating subtasks to individual agents according to their set of skills. This process involves four steps described in the following: During the task announcement, the agents will determine whether they meet the requirements for the tasks and thus are eligible for the offered task. When the bid processing and the award processing take place the task is allocated to the winning bidder which then will try to solve the acquired task.

Following the Task Sharing, the Result Sharing takes place, where the solutions for the subproblems, computed by the different agents, are shared between the agents and a general solution will be assembled. This process can be improved by cross-checking the solutions.
3.2.5.2. Planning and Synchronisation

Synchronisation between agents is a central issue in multiagent systems and has several purposes [Sun11, Page 17 et seq.]. On the one hand, synchronising agents is necessary for the safety of a multiagent system. Agents have to be informed about positions and planned actions of other agents as well as status updates about the environment to avoid conflicts or collisions. In the Treasure Hunt Game, information about already occupied fields and positions of obstacles are exchanged after each movement to avoid these complications.

On the other hand, precision is another aspect that is facilitated by synchronisation. A collaboration of multiple agents requires precisely tuned elements to guarantee the successful execution of a task. Precise positions and actions are for example very important for agents participating in the RoboCup: A as small as possible divergence when passing the ball to the other agent is fundamental for a successful pass, an imprecise execution would lead to a misdirected pass and the ball would end up somewhere else on the playing field. For multiagent systems that are deployed in air or rail transport, precise timing between the individual elements is indispensable, otherwise risking collisions and accidents.

Efficiency is also a reason why synchronisation is particularly important. Illustrated by the example of a production system: The better the systems is synchronised, especially with regard to precision of position and timing, the higher is the efficiency of this system. A perfectly synchronised production system has a higher productivity and a lower potential for errors compared to the average.

Less relevant for multiagent systems is the issue of entertainment. It refers for example to the synchronisation between sound and image for a film. Synchronisation also significant for communication, e.g. establishing a client/server connection that allows for a mutual exchange of information.

The coordination of a multiagent system requires an instance that is responsible for planning the actions of each agent. This instance can have three different manifestations, cf. [Woo09, Page 178]:

- **Centralised planning for distributed plans** is the case when a central instance plans the action of each agent and subsequently passes a completed plan to the agent which only has to implement it. The Treasure Hunt game follows this strategy: A centralised planner calculates the actions for all agents which then implement the plan provided by the planner.

- **Distributed Planning** means that multiple agents take over the planning by constructing a general plan where each agent contributes to the process by providing their specialised knowledge on the subject.

- **Distributed planning for distributed plans** differs from distributed planning in the aspect, that multiple agents cooperate to create individual plans instead of a general plan.
3.3. Decision Making in Multiagent Systems

Most multiagent systems will reach a point where there are conflicts between agents, either in their respective goals or resources. For this situations, there are different concepts and means to resolve these discrepancies between agents and to determine an optimal common strategy that enables the agents to achieve their goals.

3.3.1. Voting

In a collaborative system, there are a multiple number of agents that each follow their own strategies and objectives. To make a system of multiple agents possible, there must be a set of rules that determine the group goals, intended subgoals and the actions of each agent. For this, there are different voting procedures used in multiagents systems that balance the interests of all agents to decide on a common goal, cf. [Woo09, Page 255 et seq.], [SLBJ09, Page 241 et seq.] and [Wei13, Page 226 et seq.].

- **Plurality voting** is the most commonly used method to decide on the majority opinion of a group. Each participant compiles a ordered list of the available options, sorted ascending from the less preferred to the most preferred items. By combining all ratings, a global rating is created that represents the general order of options. While it is the simplest way to determine the opinion of the majority, there is a drawback to it: In the case that there are two items that have the same highest rating, there needs to be a tie-breaking rule. It could be a solution to repeat the voting process, counting on the possibility that some participants reconsider their rating. Or to sort the voting according to a second condition, e.g. assigning a reputation to the participants that is calculated into the global rating. A RoboCup example for this voting rule could be the possibility to decide between the next move in the game. There are several options possible and each agent sorts the possibilities according to their opinion. Agents with a defensive position have of course a different order than offensive agents. By combining the voting, a strategy that is preferred by the majority is selected.

- **Cumulative voting** is a voting procedure where each voter has a predefined number of votes that can be assigned to the available options. The number of votes each option can assigned can vary, e.g. the voter has the possibility to assign all their votes to one option, or the number of votes one option can receive from voter is limited to a certain number. This type of voting could be applied to a multiagent system that has the objective to explore unknown environments and to determine resource deposits. Amongst a selection of areas that can be explored next by the group, each agent distributes values to areas that are the most promising according to their respective sensor data. With this strategy, the area that has the highest probability of deposits can be determined.

- **Plurality with elimination** is a recursive voting process: Each voter gives their vote to their preferred N option and the option with the lowest rating will be eliminated for the next voting turn. This step is reiterated until only one option remains that subsequently has won the voting. It can also be a very time-consuming process depending on the number N of available options: It requires N-1 turns until a winner
Chapter 3. Intelligent Agents and Multi Agent Systems

is determined. Notwithstanding the above, this voting process is a very powerful tool compared to most of the other possibilities: Each voter can reassess their favourite option for each turn, if their previous favourite was eliminated or if the conditions that determine their vote changes during the process. Thus, it is suited for precarious, far reaching decisions that are not very time-sensitive.

- **Borda voting** is a voting procedure where each voter first rates all N options according to their personal benefit. The candidate that is ranked on the highest position is awarded N - 1 counts. The allocation of counts follows the general rule that the kth highest candidate is awarded N - k counts.

This selection of voting procedures only represents a small subset of different voting processes used in multiagent systems, nonetheless they provide a brief insight into the topic of how to coordinate multiple agents by combining their strategies and goals to common goal.

### 3.3.2. Auctions

Auctions play a crucial role in multiagent systems because they are the basic mechanism for allocating resources and tasks among selfish agents. In real life, people are encountering auctions on an almost daily basis, for example when making an offer for a product on an online auction house. Auctions between agents follow a similar process: Several agents want to be allocated a certain good and make offers to win the auction. The agent with the best offer is awarded this good.

Formally speaking, auctions are protocols that determine the process of allocating scarce resources to members of a group. This process of dividing resources only makes sense if more than one agent demands the same resource. If this is not the case, the agent is allocated the resource without further auctions.

An auction has several properties, cf. [Woo09, Page 294 et seq.]: They have an auctioneer that offers a particular resource, called good, and a number of agents that are interested in this resource, called bidders. Each good can possess a private and a common value that determine the value the good has for certain agents. The common value describes the real value of the good whereas the private value refers to the sentimental value this good has for a bidder. Illustrated by the example of a postal stamp, the common value of the stamp corresponds to the purchase prize. A bidder however may be prepared to pay a much higher price because the stamp is a particularly rare one. The correlated value is a mixture of the common and private value. It is an important element if the bidding agent intends to resell the good at a later point in time and tries to maximise the profit margin by assess the market value based on the common value and the private value the good has for other bidders.

Another distinction between auctions is whether they are first-price or second-price auctions. At a first-price auction, the winner has to pay their highest bid to be allocated the good, whereas at a second-price auction, the winner only has to pay the second-highest bid, which was placed directly prior to the winning one.

There is also a difference between open cry and sealed-bid auctions: If each bidder has a complete knowledge concerning the bid of other agents, it is a open cry auction. A
sealed-bid auction is the case when a bidder has no knowledge about the value of other
agent’s bids.
Lastly, auctions can also be classified according to their proceeding. Descending auctions
are auctions that start with a high value specified by the seller: The value is reduced by
small increments and the good is allocated to the first bidder willing to pay its current
value. Ascending auctions follow the traditional process: The auction starts with a small
value defined by the seller and, subsequently, bidders place their increasing bids until a
winner is determined. At single round auctions, each bidder places one bid and the winner
is the agent that offered the highest value.

[SLBJ09, Page 315 et seq.] and [Woo09, Page 294 et seq.] describe a variety of auc-
tion types which will be introduced in the following sections.
Auctions are called Single-Good auctions, if an arbitrary number of agents want to buy
one single good. If more than one kind of one good is to be allocated in a auction, these
auctions are called multiunit auctions. This could for example be the case in a computer
network where additional computing capacity should be divided amongst the agents.

3.3.2.1. English auctions

English auctions are the traditionally known auction which are established in Auction
Housed. They are classified as first-price, open cry and ascending. They always follow
the same process: The Seller specifies a selling price for the desired good and following
this, the agents that are interested in this good begin to place bid. Each bid has to be
higher than the prior one and the agents that placed the highest bid when the auction
closes is allocated the good. If no agent has placed an offer, the good is again allocated
to the seller. After the end of the auction, the winner of the auction has to pay its last
placed bid.
The closure of the auction can be determined by different mechanisms: The auction is
finished if no agent wants to raise the current bid, the duration of the auction has ex-
ceeded a particular time limit or if no new bids are placed within a specified time period.
The most frequently used strategy to win this type of auction is to raise the bid by
small increments until the agents personal limit is reached. If the value of the offered
good is unknown to the agents, a situation called winner’s curse emerges: Considering
that the winner of the auction can not assess the value of the good, they do not know if
they made a successful transaction and obtained the good for a reasonable fee or less, or
if they sustained a loss.

3.3.2.2. Japanese auctions

The Japanese auction resembles the English auction in many aspects, only the manner of
how bids are placed and incremented differs. The price of the good is raised by the seller
in certain increments and after each increase agents indicate if they are willing to pay the
current price or not. The winner of the auction is the last still remaining agent.

3.3.2.3. Dutch auctions

Dutch auctions are open-cry, descending auctions. They start with a artificially high price,
specified by the seller, that is reduced gradually until a bidder announces their willingness
to pay the current price. This bidder is the winner of the auction and is allocated the good.

### 3.3.2.4. First-price Sealed-bid auctions

In this type of auctions, each bidder respectively places their bid without having knowledge about the extend of other agent’s bids. This auction, since it is a one-shot auction, only has a single round and the winner is the bidder that offered the highest value. There is no general solution to this kind of auction, the best strategy however is to place a bid that is slightly below its real value.

### 3.3.2.5. Vickrey auctions

Vickrey auctions are second-price, sealed bid, one-shot auctions. This has the consequence that all bidders can place only one single bid that is not visible to other agents. The winner is the agent that offered the highest bid, but instead of paying this bid, they have to pay the second-highest bid to be allocated the good. The best strategy for winning this type of auction is to bid the true value of the good.

### 3.3.2.6. Multiunit auctions

Multiunit auctions follow the same process as single-good auctions, but there are differences in the way the bidding are placed and the goods are allocated. Bids can either be all-or-nothing or divisible bids.

In the case of a all-or-nothing bid, the offer placed by the agent applies to a specific number of goods and less than that number are not accepted. An agent would place this kind of bid if they require this specific amount of goods to progress to their goal and a fewer amount of goods would serve no purpose. A divisible bid specifies the amount of goods a agent would like to win, but a fewer amount would be also acceptable.

If an auction follows a discriminatory auction rule, the amount n of goods are allocated to the nth highest bidders, each paying their offered value. Would a uniform pricing rule apply, each winner would pay the same value for their good, determined by the lowest winning or highest loosing bid.
4. Problem Solving and other Fields of Research

The interaction between several different individuals and the resulting effects on work performance is also a significant field of research in the psychological science, particularly in industrial and organisational psychology as well as the communication psychology and the cognitive psychology. Some of the aspects researched in this fields can also be applied on multiagent systems, their construction and their advantages over single agents.

4.1. Cognitive Psychology and Problem Solving

Cognitive science is an interdisciplinary research field that is based on theories and methods of philosophy, linguistic, artificial intelligence, cognitive neuroscience and cognitive psychology, cf. [Mü8, Page 624], and also computer science. Particularly interesting for the topic of problem solving is the cognitive psychology, which researches, among other things, the process of problem solving and decision making.

While the terms problem solving and decision making are related, and thus often interchangeable, and also follow a similar process, there is a slight difference between them, see [Hui92]: While decision making describes a situation where a simple selection out of a variable of distinct available solutions is required to achieve a goal, the term problem solving refers to the process of reaching a goal state by starting from an initial state and applying partially unknown operators to attain subgoals.

A problem solving process is characterised by three different aspects, cf. [And07, 291]:

**Goal-oriented**: The individual has a clear objective which it strives to reach. Illustrated by the Treasure Hunt game, the common goal is to open the treasure chest and thus winning the game.

**Subgoals**: A problem solving process can be divided into subgoals and subtasks. In the game, several subtask have to be achieved to fulfil the conditions for opening the treasure chest, e.g. retrieving the key to the chest.

**Operators**: The solution process to reach the final state is composed of a distinct sequence of operators. With the application of each operator, a subgoal is reached, reducing the distance to the goal state. Operators for the game are for example exploring the territory or producing wooden boards to build a bridge and uncover new areas of the map.

According to [Hui92] and based on [BS93], [Dew33] and [Pol71], the problem solving process has four different phases:

1. **Input Phase**
   The input phase is characterised by clarifying, identifying and stating the problem. Following this, possible solutions are evaluated and additional information is
gathered to improve the problem assessment.

2. **Processing Phase**
   The design and evaluation of solutions is an integral part of the processing phase. First, an as large as possible variety of solutions are developed which then are evaluated to find the best solution to the problem.

3. **Output Phase**
   The output phase is the part of the process where the solution selected in the processing phase is implemented.

4. **Review Phase**
   After the implementation and application of the solution, the implementation as well as the effectiveness are evaluated, identifying problems and improvements, based on which the solution will be modified.

### 4.2. Organisational Psychology and Group Dynamics

Organisational psychology is the science of human experience and behaviour within organisations, focusing, among other things, on group dynamics and behaviour and also the effects on the working process. Communication psychology and communication science blend into each other concerning the communication between individuals and the related implications.

#### 4.2.1. Theoretical basics

An organisation is a system that pursues a specific objective, consists of individual and groups forming a social structure, and has a distinct structure which is characterised by division of labour and a chain of responsibility, see [RN11, Page 6]. Particularly interesting for the topic of cooperative problem solving is the branch of organisational psychology that studies groups and their dynamics.

According to [RN11, Page 283], a group is defined by several attributes: it consists of a plurality of persons that interact directly with each other over a prolonged period. Other characteristics are a differentiation of roles, common values, standards and goals as well as a feeling of unity. While the upper limit is difficult to determine, groups consist of at least three members, although the optimal group is composed of five members: it is small enough for discussions and also small enough to find compromises that are satisfactory to all members [NBS11, Page 96 et seq.].

While the group structure and size has not the same high impact on virtual or computer-based multi-agent systems as it has on a team consisting of multiple persons, it also has an effect on performance and efficiency. Systems with a large quantity of agents with individual and perhaps conflicting objectives require well-constructed and goal-oriented planning to ensure optimal solutions and implementations.

As theorised by the management theorist Belbin in the 1970, there are nine different roles that should be represented in a successful team, categorised in three behavioural patterns [Bel]:
• The activity-oriented members of the group are the Shaper, the Implementer and the Completer/Finisher. The Shaper is the task-oriented leader of the group, characterised by their high motivation to reach the team's objective, while the Completer is responsible for ensuring the compliance with deadlines and the delivery of error-free, accurate work. Finally, the Implementer is efficient and disciplined, albeit conservative, concerning themselves with the development and implementation of ideas.

• The Coordinator, Teamworker and Resource Investigator represent the people-orientated team roles. Monitoring and organising the team activity is the task of the Coordinator whereas the Resource Investigator establishes contacts outside the team, bringing in intelligence and support. The Teamworker is responsible for the people-to-people contact within the team, occupying a diplomatic function by dealing with conflicts between team members.

• Finally, a team also includes solution-oriented roles, e.g. the Plant, the Monitor Evaluator and the Specialist. Whereas the Plant develops new ideas and solutions that are distinguished by an unorthodox and creative way of thinking, the Monitor Evaluator is logical and level-headed, evaluating all possible solutions to work out the best possible solution. Ultimately, the Specialists are experts in their field of study and contribute expertise and specialised abilities.

4.2.2. Problem solving and decision-making in groups

Groups often have an advantage in solving tasks compared to individuals:

"Group performance was often quantitatively superior to the performance of individuals, for example, number of correct solutions (...), insofar as groups benefited from pooling of information, opportunities for one member to correct another's errors, and statistical cancellation of errors in the computation of average member performance. (...) [They also] benefited from member aggregation in that the probability of including an exceptionally competent member increased with group size.” [Hil82, Page 533]

According to [LHSB06], testing the effects of group size on letters-to-numbers problems (ten letters were randomly assigned to numbers 1 - 10, goal was to assign the numbers to the letters in as few steps as possible, achieved by solving equations in the form of $A + B = \text{letter}$) the result was that by enlarging the group size from two to three members, the performance and complexity of the solutions increased noticeably. The study also concludes that for this particular problem, the performance of two-person group resembles that of the best individual and that the performance of groups with three, four or five people hardly differs from each other.

Cooperative problem solving though only makes sense for a certain type of tasks, but fundamental to group work is of course that the projects has the potential to be split in subtasks. There are two categories of tasks: For a start these that benefit from group performance, and on the other hand those tasks group work would pose a disadvantage to. Examples that fall into the last-mentioned category would be routine work as well as short or confidential tasks. The first category of tasks has specific characteristics as well as
demands on the group composition. This difficult problems have several characteristics, outlined by Dörner [Pro]:

- **Intransparency**
  If the problem has multiple possible solutions, is vague or can be solved through different approaches, group work would be beneficial for the process by combining various ideas and opinions.

- **Polytely**
  Cooperative problem solving is also advantageous for tasks that affect many people, and therefore should fulfil various expectations which conflict each other. By combining and controverting different solutions, an as ideal as possible solution can be worked out.

- **Complexity**
  Complex tasks, which for example cover a widespread spectrum of knowledge, are predestined for group work. They would benefit from a group of experts from different disciplines that are related to the problem.

- **Dynamics**
  For work-intensive projects group work also presents a good alternative, progressing faster compared to an individual.

### 4.2.3. Advantages and disadvantages of groups

Approaching problems in a group has several advantages and disadvantages, depending on the category of the problem, the composition of the group and various other factors. As listed by [Had] and [Mai99], there are numerous pros and cons for teamwork, some of them pointed out in the following paragraph:

**Advantages**

- **Greater output and sum of knowledge**
  The more people are involved with a problem, the better and multi-variant are the resulting ideas of how to approach the problem, because each group member has different point of views and experiences, which then can be combined to the best possible result. A group member can complement the knowledge of others, enhancing the ideas by adding theoretical or practical knowledge and aspects of which others are unaware of.

- **Reduced bias and increased risk taking**
  Collaborative problem solving can also lead to new, unconventional and perhaps riskier ways and ideas, that, however, may be superior to conventional methods.

- **Higher acceptance**
  Team work can also increase the acceptance of unconventional or unprecedented approaches, if the affected factions are involved in the solution process and can contribute own suggestions and opinions.

Group work can induce higher motivation for each member compared to individual work. According to [NBS11, Page 99 et seq.], there are three different forms of gaining motivation:
The first aspect is the so-called "Mere Presence", theorising that already the mere presence of group members increases the motivation, caused by two factors: the companionship of others has an activating effect and can also induce self-promotion to enhance the own image.

The second aspect is the "Social Compensation": group members put extra effort if they have the impression that other members are not contributing enough to the project, if is very important to them that the projects is finished with good results, or if their input affects the group to a high extend.

Lastly, the identification with a group causes an increased individual performance, referred to as "Social Labouring".

**Disadvantages**

- **Competition**
  The work flow and development of new ideas suffer if there are conflicts within the group or the group has a bad group dynamics, therefore negating all advantages that collaborative problem solving would cause.

- **Conformity and social pressure**
  The creativity and the combination of a variety of possible new approaches and ideas are affected by the tendency of individuals to be integrated as best as possible into the group, thus withholding ideas which could be met with refusal.

- **Lack of objective direction**
  Team projects also often require one or more leading members, structuring and monitoring the process, to ensure a steady progress.

- **Time constraints**
  In some cases, team works is more time consuming compared to solving the problem individually. That of course depends on the category of the projects: Some are solved faster if addressed by a whole group. Therefore, if a problem is very time sensitive, individual works is favourable, whereas a project that is not time critical may be more convenient for group work.

Team work can also cause an decrease of motivation. There are five different causes for this phenomenon, see [NBS11, Page 102 et seq.]:

"Social loafing" describes a loss of motivation by one of the group members. While the person does not perceive the reduction, their performance nonetheless decreases.

A further cause is the "Social anxiety": the presence of other individuals, especially ones that are perceived as particularly important. The resulting inhibitions decrease the motivation and performance.

If an individual within the group reduces their contribution to the work deliberately, anticipating that the rest of the group can handle the workload themselves, it is called "free riding".

The aforementioned effect is often cause for the "sucker effect": If an individual has the impression that other members do not contribute as much as them, they also lower their effort, even if they could contribute more to the problem.

"Soldiering" describes the behaviour of the group members if they are confronted with external demands, which the group, in their opinion, is not capable to fulfil, thus resulting in lower motivation and performance.
4.2.4. Communication Structure

Particularly important for the cooperation of a group, and likewise the resulting performance and success of problem solving, is the existing communication structure. Suboptimal communication is a factor that often causes problems in group projects and delaying the progress of the project. At this point, the organisational psychology intermingles with the communication science: The communication scientist Harold Dwight Lasswell designed the so-called Lasswell formula, which describes the basic communication process in groups, see [RN11, Page 316 et seq.]:

- Who: The communicator or sender of the message
- What: The message which is supposed to be transmitted
- Whom: The receiver of the message
- Channel: The medium used to transport the message
- Effect: The effect that the message has on the receiver

Certain aspects of these theoretical basics can also be applied to the Treasure Hunt game and its agents. Permanent communication between the agents is essential for the flow of the game. New discoveries and performed actions have an influence on the actions and reactions of other agents, and thus, for bringing them up to date, they are forwarded to other agents.

To demonstrate the application of the Lasswell formula in an exemplary way, it can be applied to a game scenario. Since the game follows a turn-based strategy, the information is not relayed instantly, but rather at the end of each turn. The circumstances are as following: The Explorer finds a map on the playing field which holds important information for the other agents concerning the sites of key fragments. The Explorer therefore poses as the communicator, transmitting the message - discovered fields and the sites - to the other agents that represent the receiver. The channel is in this case an update-method which provides the information for the other agents, that now consequently travel to the sides to retrieve the fragments.

According to [NBS11, Page 60], by sorting formal communication structures by their degree of centralisation, there are three resulting structures, see figure Formal communication structures.

![Figure 4.1.: Formal communication structures](http://www.springerimages.com/Images/Psychology/1-10.1007_978-3-642-16972-4_5-1)
The wheel in the left is the most central structure, where all communication runs through one key instance. That is also the case in the Treasure Hunt game, all agents updating information to and retrieving changes from one central platform.
5. Treasure Hunt Game

Firstly, to clarify the fundamental terminology for the following subsections: The game characters which can be controlled by the player are referred to as agents, while the elements which the player and the agents can interact with are called objects. The term "actors" refers, based on the agents terminology, to both agents and objects.

Furthermore, there are four distinct ways to start this game: The player can either play the game themselves by choosing the player-controlled multiagent game or the player-controlled superagent game, or watching the game’s AI play the game by selecting the autonomous multiagent game or the autonomous superagent game, for further explanations see section Player-controlled Gameplay and Autonomous Gameplay.

5.1. Game Setup

The Treasure Hunt Game has three different screens. If the game is executed, at first the MenuScreen is displayed, which offers four different ways to start the game. These four different possible game styles are called multiAgentHuman, superAgentHuman, multiAgentAutonomous and superAgentAutonomous, see Player-controlled Gameplay and Autonomous Gameplay for additional information. Furthermore, the upper-right button opens a help file that explains the structure and control of the game.

![Figure 5.1.: Screenshot of the MenuScreen](image)
Chapter 5. Treasure Hunt Game

The GameScreen is the main screen of the Treasure Hunt Game and is divided into several parts: The world display and the information display, which are detailed further in the following sections:

Figure 5.2.: Screenshot the GameScreen

5.2. World

The world display, which is region 1 in the GameScreen, see 5.2, shows the playing field with its actors. It is also the area where the actual game activity takes place and consists of a tiled map, further detailed in Tiled, as well as the Agents and Objects with whom the player can interact. The interaction with the game actors is further explained in Player-controlled Gameplay.

The landscape of the game is divided into three different types of territory: The actual playing field which is accessible to the agents, the surrounding cliffs which the agents cannot access, and lastly several barriers, e.g. the river and the lava, which are only accessible if certain preconditions are fulfilled.

5.3. Information Display

Region 2 in the GameScreen, see 5.2, is the information display which shows information regarding the current game status like messages, information about the current turn, skills of agents as well as the turn count and button. This elements are divided as following:

- Region 3
  This display, called message display, is generated from all feedback the player receives from the game. This feedback is composed of information on the agents, e.g.
their name and their current status, information on the objects, e.g. their purpose, operating condition and status, as well as explanations of the actions performed by the agents, e.g. interacting with an object.

- **Region 4**
  The turn display shows information referring to the actual turn, which would be the remaining steps each actor has left over.

- **Region 5**
  This information relate to the selected agents, showing their skill profile. Included in the skill display are the steps which the agent has on their disposal, and the amount of turns the agent needs to dig something out, cut wood, build something, explore the cave, fight, or summon something as well as the information whether they are fireproofed.

- **Region 6**
  With the turn button the player can complete the actual turn and start a new one, resetting the remaining steps and updating the status of each agent. The turn count shows the current number of turns, including the actual turn.

### 5.4. Actors

The actors are the principal features of the game and depending on the chosen game style, there are two different configurations of Agents at the player’s disposal: either a single superagent or a combined group of specialised and non-specialised agents. Notwithstanding the above, the Objects that the player and the agents interact with have always the same configuration.

#### 5.4.1. Agents

If the player decides to start the game with one superagent, this agent combines all skills that all other agents in the multiagent game have at their disposal and that are required for the winning of the Treasure Hunt Game.

If a multiagent game approach was selected, there are seven different agents at the players command. At the current status, the game does not allow for a further modification of the number of agents or their skills. Based on this, a feature that would allow the configuration of the number of agents (1 - n) and skills (1 - f) would be worth to be considered for Further Improvements.

All agents have a skill set that, basically, describes how well they perform on a task. There are five different skill configurations, each containing the same skills but with different graduations: one for the three non-specialised agents and another four for the specialized agents. While there exist three different non-specialised agents, each specialised agent only occurs once in the game.

Hereinafter, a listing of all agents and their skill sets will be presented. To clarify the skill sets: The first value Steps/Turn states the steps which an agent can take each turn while the following values define how much turns the agent require to complete the corresponding task.
Chapter 5. Treasure Hunt Game

Figure 5.3.: Agents of the Treasure Hunt Game

- **Non-specialised agents:**
  There are three agents on the fields that do not have the skills to fulfil specialised tasks like fighting or summoning. These agents, in the game referred to as Agent1, Agent2 and Agent3, see figures 1 to 3 of *Agents of the Treasure Hunt Game*, have all the same set of skills with no graduation within one skill. Their skill set is configured as following:

<table>
<thead>
<tr>
<th>Steps/Turn</th>
<th>Caveexploring</th>
<th>Digging</th>
<th>Fighting</th>
<th>Woodcutting</th>
<th>Fireproof</th>
<th>Bridgebuilding</th>
<th>Summoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.1.: Skills of non-specialised agents

- **Specialized agent: Explorer**
  The Explorer, which is figure 4 of *Agents of the Treasure Hunt Game*, is an agent that specialises in scouting, therefore it has double the number of steps per turn compared to the majority of the other agents. Due to its size, it also has the ability to explore caves, which other agents cannot access.

<table>
<thead>
<tr>
<th>Steps/Turn</th>
<th>Caveexploring</th>
<th>Digging</th>
<th>Fighting</th>
<th>Woodcutting</th>
<th>Fireproof</th>
<th>Bridgebuilding</th>
<th>Summoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.2.: Skills of Explorer

- **Specialized agent: FireImp**
  While also possessing an exceptionally high speed, the FireImp, see figure 5 of *Agents of the Treasure Hunt Game*, it is also fireproof, which enables it to reside in otherwise not accessible territories.
Chapter 5. Treasure Hunt Game

Table 5.3.: Skills of Explorer

<table>
<thead>
<tr>
<th>Steps/Turn</th>
<th>Caveexploring</th>
<th>Digging</th>
<th>Fighting</th>
<th>Woodcutting</th>
<th>Fireproof</th>
<th>Bridgebuilding</th>
<th>Summoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.4.: Skills of Knight

<table>
<thead>
<tr>
<th>Steps/Turn</th>
<th>Caveexploring</th>
<th>Digging</th>
<th>Fighting</th>
<th>Woodcutting</th>
<th>Fireproof</th>
<th>Bridgebuilding</th>
<th>Summoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.5.: Skills of Explorer

<table>
<thead>
<tr>
<th>Steps/Turn</th>
<th>Caveexploring</th>
<th>Digging</th>
<th>Fighting</th>
<th>Woodcutting</th>
<th>Fireproof</th>
<th>Bridgebuilding</th>
<th>Summoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.5.: Skills of Explorer

<table>
<thead>
<tr>
<th>Steps/Turn</th>
<th>Caveexploring</th>
<th>Digging</th>
<th>Fighting</th>
<th>Woodcutting</th>
<th>Fireproof</th>
<th>Bridgebuilding</th>
<th>Summoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Specialized agent: Knight**
  Due to its strength, the Knight, which is displayed in figure 6 of Agents of the Treasure Hunt Game, has two specialised skills: building bridges and fighting. It can also dig, but that takes him an additional turn compared to the non-specialised agents.

- **Specialized agent: Wizard**
  The Wizard is figure 7 of Agents of the Treasure Hunt Game and has the specialised skill to summon the FireImp. She can also cut down trees to retrieve wood, but requires two more turns to complete the task compared to non-specialised agents.

- **SuperAgent**
  By selecting the SuperAgent game style, the RPG is initialised with one single agent, see figure 8 of Agents of the Treasure Hunt Game. This SuperAgent has all skills required for winning the game and specialises in all skills. Comparing its skill set to others, he always posses equal or superior skills.
Chapter 5. Treasure Hunt Game

<table>
<thead>
<tr>
<th>Steps/Turn</th>
<th>8</th>
<th>Caveexploring</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digging</td>
<td>1</td>
<td>Fighting</td>
<td>1</td>
</tr>
<tr>
<td>Woodcutting</td>
<td>1</td>
<td>Fireproof</td>
<td>0</td>
</tr>
<tr>
<td>Bridgebuilding</td>
<td>1</td>
<td>Summoning</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.6.: Skills of SuperAgent

5.4.2. Objects

There are different objects on the map with which agents as well as the player can interact to progress in the game plot. These objects are illustrated in figure Objects of the Treasure Hunt Game.

![Objects of the Treasure Hunt Game](image)

At this point, I will provide a short list of the objects and their intended purpose:

- **Bush**: Within this object, the CavePlan is hidden which is required to explore the cave.
- **Dragon**: Can be slayed and drops a map to the KeyFragments.
- **Ford/Bridge**: The Ford is a natural barrier for the agents, but can be crossed if wood is collected and a Bridge is built.
- **XMark/KeyFragment**: XMarks appear on the map when the map for the KeyFragments is retrieved. If agents dig there, KeyFragments are detected.
- **SummoningShrine**: Used to summon the FireImp.
- **Sword**: Can be collected and is required to slay the Dragon.
- **TreasureChest**: Main goal of the game! The Treasure can be acquired by combining the KeyFragments to a key which opens the chest.
- **Tree**: By chopping down a Tree, wooden boards are produced which are required to build the Bridge.
Chapter 6. Implementation and Evaluation

6. Implementation and Evaluation

This chapter will start with an overview over the used software and graphics, especially describing some basics about the game development framework LibGDX. Following this, there will be a short outline of questions concerning game design as well as an introduction into turn-based strategy games. After that, the implementation of the game will be addressed as well as possible improvements regarding this implementation. Ultimately, there will be a comparison between the different ways to play the game, evaluating the average amount of turns each required until the treasure was retrieved.

6.1. Software and Graphics

The first step in implementing the Treasure Hunt game was to consider which instruments should be deployed for the realisation. The decision for a low-level approach by using the Java 2D API would mean rather unrestricted possibilities, but also the necessity to implement fundamental functionalities like a game loop and the drawing process of the game. A high-level approach would be the use of a game engine, which would provide a vast number of functionalities, but would also pose a restriction for designing the architecture of the game.

While there exists a huge quantity of engines/libraries for game developing, the vast majority of them are C++ based. Only a small number is Java-based, and among them the ones most used for small-scope games and apps are LWJGL, jMonkey Engine/Ardor3D, Slick2D and LibGDX.

For the implementation of the game, I considered LibGDX¹, Slick2D² and jMonkey Engine³. Although each of them has their own advantages and disadvantages, the following will outline why eventually LibGDX proved to be suited best for meeting my requirements.

What LibGDX, Slick2D, and also the jMonkey Engine all have in common, is that they all are based on the open source Java library LWJGL (Lightweight Java Game Library)⁴, which provides simple access to the functionalities of the high performance libraries OpenGL, OpenAL and OpenCL.

Both LibGDX and Slick2D are libraries designed to develop 2D games, but in comparison LibGDX offers more advantages. For one thing, while Slick2D only ports to desktop, LibGDX enables cross-platform programming. Also, in comparison, LibGDX is more up to date, releasing daily Nightly Builds with latest bug-fixes and new features, and also providing a rather extensive API⁵, as well as elaborate tutorials and an active community.

¹LibGDX: http://libgdx.badlogicgames.com/
²Slick2D: http://slick.ninjacave.com/
³jMonkeyEngine: http://jmonkeyengine.org/
⁴LWJGL: www.lwjgl.org
⁵LibGDX API: http://libgdx.badlogicgames.com/nightlies/docs/api/
The jMonkey Engine is designed for developing 3D games, but it is also possible to realise a 2D game structure. LibGDX, too, makes it possible to develop 3D games, which, while it does not apply to the current 2D game, is a nice feature for a possible expansion of the game.

Unlike LibGDX, which is a library, jMonkey Engine is a whole engine. But although a game engine provides more functionality and a higher level of abstraction, it is also more rigid and inflexible. Therefore, I decided to use LibGDX, because it allows for more freedom and own solution approaches.

6.1.1. LibGDX

LibGDX is a widely used framework and was already deployed for the construction of over 700, predominantly small-scale, games [Libb]. It is a java-based open source framework, intended for developing games which operate on multiple platforms: Windows, Linux, Mac OS X, Android, iOS and HTML5, see [Liba]. The basic principle of LibGDX is to implement your application once on your native desktop and then subsequently exporting the code to the other platforms, which is achieved by the means of abstracting the differences between the platforms.

For a listing of all LibGDX-features, taken from the LibGDX Website [Liba], see appendix Features of LibGDX.

To make your application executable on Windows, Android, Linux, Mac OS X and HTML5, the Google Web Toolkit (GWT)\(^6\), as well as the Anroid SDK and Eclipse ADT Plugin\(^7\), are required in addition to Eclipse and a JDK.

A LibGDX project is structured as following, illustrated on the basis of my project, see [Libc]:

- **Core Project ("treasure-hunting")**
  All of the actual implementation takes place in this project, the other projects only linking to this code.

- **Android Project ("treasure-hunting-android")**
  Contains, among other things, the class MainActivity.java, which is used as starter class for the Android application, as well as the assets folder, where all files used for your game, e.g. images, soundfiles, skins, fonts and tiled maps are stored.

- **Desktop Project ("treasure-hunting-desktop")**
  Holds the Main.java class, used to start the projects as a desktop-application. References to the code in the Core Project and the assets within the Android Project.

- **HTML5 Project ("treasure-hunting-html")**
  Responsible for executing the project as HTML5 application, consists of the class GwtLauncher.java and also links to the Core Project and the Anroid Project’s assets folder.

While not used in this game, it is also possible to integrate an iOS RoboVM Project, which would make it possible to run the application on iOS via the Eclipse Plugin RoboVM\(^8\).

---

\(^6\)Google Web Toolkit: http://www.gwtproject.org/usingeclipse.html

\(^7\)Android SDK and Eclipse ADT Plugin: https://developer.android.com/sdk/index.html#ExistingIDE

\(^8\)RoboVM: http://www.robovm.org/docs#eclipse
and the IDE XCode\textsuperscript{9}.

### 6.1.2. Textures and Sprites

All used Textures are either from CGTextures\textsuperscript{10}, Free Stock Textures\textsuperscript{11} or from OpenGameArt\textsuperscript{12}, which provide a vast number of free textures and artworks. Sources for sprites used for agents and objects, as well as the tile sets used for creating the map, are Sithjester’s RMXP Resources\textsuperscript{13}, Reiner’s Tilesets\textsuperscript{14} and TomeTik\textsuperscript{15}. The usage of the sprite sheets for agents will be explained in namerefsec:implementation and the application of the tile sets will be described in the following paragraph.

### 6.1.3. Tiled

For the design of the playing field the free tile map editor Tiled\textsuperscript{16} was used. Tiled that allows you to create orthogonal or isometric multi-layer maps based on tile sets, for which LibGDX already provides a packer, loader and renderer.

![Screenshot of Tiled](http://www.mapeditor.org/)

Figure 6.1.: Screenshot of Tiled

Multi-layered maps allow for an simplified level design: higher-level layers always overwrite underlying layers. With this characteristic, it makes sense to deploy a bottom-up sequence for creating a detailed map. In my case the lowest layer, Ground, contains different soils like grass or swampland, followed by the Texture layer with additional terrain

\textsuperscript{9}XCode: https://developer.apple.com/xcode/
\textsuperscript{10}CGTextures: http://www.cgtextures.com/
\textsuperscript{11}Free Stock Textures: http://freestocktextures.com/
\textsuperscript{12}OpenGameArt: http://opengameart.org/
\textsuperscript{13}Sithjester’s RMXP Resources: http://untamed.wild-refuge.net/rmxpresources.php?characters
\textsuperscript{14}Reiner’s Tilesets: http://www.reinerstilesets.de/
\textsuperscript{15}TomeTik: http://pousse.rapiere.free.fr/tome/
\textsuperscript{16}Tiled: http://www.mapeditor.org/
details, and the Water layer, used for creating the river. Up next the Mountain layer that defines the surrounding cliffs enclosing the actual playing field. Finally, the uppermost Decoration layer adds several details like stones and shrubbery. While not used for this in the game, this feature can also take part in the physical configuration of the game, by adding actors to a specific layer in the future game. Overlaying layers will be drawn over this actors, which would make sense for layers containing objects like trees, houses e.g. that will now cover the actors if they are behind them.

Tiled also offers the possibility to specify so-called collision areas, which can be processed in the actual game, e.g. as unscalable obstacles for the agents or non-accessible areas. While not used in the current game, this feature can be used for further improvements on the game, see Further Improvements.

Tiles are used in game development for a relatively long time, based on their resource-saving characteristics. Compared to maps made from one whole image, which can slow down games due to their file size, tiled maps are usually more performant. Within a tiled map, individual tiles can be used multiple times while they are saved only once, thus decreasing the overall file size.

A tile set is a collection of several, usually uniform sized images that can be used to create a map. As an example, here are two different tile sets, one to draw several grounds, see figure Tileset 01 and the other to create transitions between to different soil structures, see figure Tileset 02.

The best known video games based on tile sets are Civilization, the Heroes of Might and Magic series, Pac-Man and most of the Pokemon games.

![Tilesets](a) Tileset 01  
(b) Tileset 02

Figure 6.2.: Tilesets

### 6.1.4. Additional Tools and Sources

Background music and sounds were created using the free audio editor Nero WaveEditor\(^\text{17}\) and sounds from Freesound.org\(^\text{18}\). For a list of all used sounds, see appendix Sounds. For creating a customised font, the bitmap font packing tool Hiero\(^\text{19}\) was used. The texture


\(^{19}\)Hiero: [https://code.google.com/p/libgdx/downloads/detail?name=hiero.jar&can=2&q](https://code.google.com/p/libgdx/downloads/detail?name=hiero.jar&can=2&q)
Chapter 6. Implementation and Evaluation

pack - on which the skin is based on - was created using the LibGDX TexturePacker\textsuperscript{20} and the program Agama Web Buttons\textsuperscript{21}.

6.2. Game Design

Before starting to implement a game, several design questions concerning the structure and agent behaviour have to be clarified. According to [MF09, Page 809 et seq.], there is a list of points that have to be considered, serving as an rough outline for my game design:

1. **Movement**
   - Individual representation of agents, which is usually used in most games, versus displaying only group effects, for instance used in simulations:
     For this game, a individual representation of each agent is suited best. Each agent can be selected and perform actions and movements, regardless of the other agents.
   - Animation of movement versus simple relocation to new position, for example used in simple board games like chess:
     While the movements of the agents are not particularly sophisticated or realistic, they are on a higher level than a simple relocation which would be displaying agents on their new position without showing the transition between the two positions.
   - Need for Pathfinding algorithms or following predetermined paths:
     Although not needed for the Player-controlled Gameplay, the 6.4.2 requires a pathfinding algorithm to compute the movement of each actor to their next subgoal.

2. **Decision making**
   - Range of actions the agent can perform:
     The scope of actions that agents can perform is relatively small. They can move on the map or interact with certain objects. Or to be more precise, there are nine distinct operations an agent can or can not perform, depending on their personalised skill set, further detailed in Agents.
   - Number of states that the agent can choose between:
     There are two different statuses an agent can have: it is either unemployed or performing an action that lasts more than one turn.

3. **Tactical and strategic AI**
   - Need for teamwork:
     By playing the game with a single superagent that can perform all tasks, no teamwork is required. However, to reach the goal of the game when selecting a multi agent approach, teamwork between the different agents is fundamental because certain actions can only be performed by specialised agents.

\textsuperscript{20}LibGDX TexturePacker: \url{http://code.google.com/p/libgdxtextruepacker-gui/downloads/detail?name=gdx-texturepacker-3.2.0.zip&can=2&q=}

\textsuperscript{21}Agama Web Buttons: \url{http://www.agamabuttons.com/}
6.3. Turn-based Strategy Games

Strategy games can implement two distinct types of time modelling: the continuous approach which is used in real-time games and the discrete approach used for turn-based games. A turn is defined by the actions the player can carry out within a turn, e.g. moving actors by a certain amount of steps or interacting with objects. While turn-based and real-time strategy games have many things in common, often using the same approach to AI and pathfinding, they differ in the time the player has to decide on their actions, cf. [MF09, Page 828 et seq.].

In a real-time game the player has a certain degree of time pressure, considering that the opponent, which is embodied by the computer, continually proceeds with making decisions and performing actions. Thus, the player can not take too much time for their operations, otherwise risking losing the game to their virtual opponent due to slower decision making or reflexes.

If the game follows a turn-based approach, the player has an unlimited time frame to decide on their next actions. By doing this, the player can develop an optimal strategy for the next turns and therefore gaining advantage over their opponent. That also has the consequence that the opponent’s AI has to be very advanced to compensate this advantage. For this, a well-balanced AI is required: it has to be good enough to pose a challenge for the player, but it can not be absolutely perfect, allowing the player to make errors and suboptimal decisions without instantly losing the game.

A combination of this two approaches is also possible, often implemented in so-called RGPs (role-playing games), for instance allowing the player to pause the game to plan and schedule the forthcoming actions and to progress in time again if all planning is finalised.

6.4. Implementation and Code

As a start, the game could only be tested on two different computers, each with the following configuration:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Windows 7, 64 Bit, Service Pack 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>4 GB</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Core 2 Duo 2.1 GHz</td>
</tr>
<tr>
<td>Graphic card</td>
<td>NVIDIA GeForce GT 120M 1 GB GDDR2 SDRAM</td>
</tr>
<tr>
<td>Java</td>
<td>Version 7 update 75</td>
</tr>
</tbody>
</table>

Table 6.1.: System Configuration

It is therefore impossible to exclude the possibility that errors may occur with other system configurations.
In this section details of the implementation will be elaborated. Particularly the illustration of how to animate the agents with the aid of sprite sheets, the details of how input will be processed as well as an overview over some of the more important used LibGDX classes and in which aspects I deviated from them, integrating own approaches. Also some code samples from the autonomous game play will be explained.

Firstly, a explanation of the project structure and the classes will be provided, as well as a short discussion over used LibGDX classes and deviations from them.

- **Package tk.treasurehunting.agents**
  The different agent classes are stored within this package, e.g. Agent1, Agent2, Agent3, Explorer, FireImp, Knight and Wizard. These classes all extend the superclass WorldAgent, which is localised in tk.treasurehunting.superclasses. They all inherit methods from the LibGDX class Actor, but rather than using the predefined way of specifying positions with a float x and a float y value, I created a Coordinate class to manage positions. The agents classes all contain a InputListener for key events and methods for the movement of the agents.

- **tk.treasurehunting.autonomous**
  This package contains the classes GamePlannerSingleAgent and GamePlannerMultiAgent which provide the AI responsible for the strategy of the autonomous versions of the game. Both implement methods for exploring, task allocation and basic pathfinding.

- **tk.treasurehunting.main**
  GameManager, TreasureHunting and TurnManager are located in this package. TreasureHunting is the basic class where the switch to the MenuScreen and the initialising of the skin used for texture in following screens. TurnManager is responsible for managing the remaining steps of each agent and the inventory, e.g. whether a objects is already retrieved or not.
  GameManager is the class where most of the action takes place. It extends LibGDX’s Stage class used for receiving input and distributing it to its actors, e.g. agents, objects or buttons. Here, the game display is initialised and it is also a centralised point to exchange information about movements between the agents.

- **tk.treasurehunting.map**
  This package is used to store classes for objects on the map and as well as the structure responsible for positions on the map, the class Coordinate. Fog is a class containing a sprite for drawing the fog of war. Bush, Dragon, Ford, KeyFragment, SummoningShrine, Sword, TreasureChest and Tree are extending the superclass WorldObject.

- **tk.treasurehunting.screens**
  GameScreen, HelpScreen and MenuScreen are the screens used for the game.

- **tk.treasurehunting.superclasses**
  WorldAgent and WorldActor extend the class WorldActors, which itself extends LibGDX’s Actor class. They are used to store attributes and methods which both agents and objects use.
• **tk.treasurehunting.task**

DiggingTask and WoodCuttingTask are used to manage actions that last more than one turn, storing the related agent, object and a timer for the time the task is finished.

Sprite sheets are the foundation for creating animated sprites. In games, sprites are often used for animated objects, for example characters or other moving objects, e.g. water, trees or flowers. They consist of multiple connected sequences of images, see figure **Example of a sprite sheet**. In this example, each row has four images that illustrate the motion sequence of walking in one direction.

![Figure 6.3.: Example of a sprite sheet](image)

The first step, see appendix **Creation of animations based on an image/sprite sheet**, the sprite sheet is split in different TextureRegions, each containing the four images for the movement in one direction. Based on this TextureRegions and by adding the display time of each frame, the Animations for each direction are created. Subsequently, depending on the walking direction and the actual time, the different frames of each animation are drawn on the calculated position, as listed in appendix **Drawing the animations**.

### 6.4.1. Player-controlled Gameplay

The player-controlled gameplay focuses on the interaction with the human player. The player takes control over the movement of the agents and their interaction with objects.

The game control is quite simple and will be explained in the following:

- To select an agent and moving them, the agent first has to be selected with a mouse click on their position

- The keys W, S, A and D are assigned to respectively execute a movement to the upward, downward, left or right field of the agent, as long as the agent has steps left for this turn
• Interaction with objects is only possible if the agent is standing on the field directly underneath the object, or, in the case of the sword, on the field on the left

• For starting a new turn, the button on the lower right corner of the display has to be clicked

### 6.4.2. Autonomous Gameplay

The control in the autonomous games is quite simple, given that movement and interaction with objects is computed by the planner. To start a autonomous game, the button START GAME has to be pressed and a progress into the next turn is made by clicking NEXT TURN.

The AI that controls the flow of the autonomous SuperAgent game differs from the one of the autonomous MultiAgent game. Subsequently, pseudo code will be provided that describes the task allocation and decision making for the SuperAgent game, see 6.1. The planner for the MultiAgent game is structured similarly, although more extensive. The AI is simply structured and allocates tasks according to the process of the game plot.

Listing 6.1: Planning the task for single SuperAgent

```java
Method calculateTurnActionsSingle() plans the tasks of the agent for each turn

boolean hasTaskSuperAgent = false;
boolean hasTaskFireImp = false;

IF superAgent has no task that lasts more than one turn
  FOR iterating over list of all trees on the map
    IF tree is already discovered and not cut down
      calculate superAgent’s path to tree
      superAgent performs action on tree
      hasTaskSuperAgent = true

  IF bush is discovered and not cavePlan is in inventory
    calculate superAgent’s path to bush
    superAgent performs action on bush
    hasTaskSuperAgent = true

  ELSE IF bridge is not build and enough wood is in inventory
    calculate superAgent’s path to ford
    superAgent performs action on ford
    hasTaskSuperAgent = true;

  ELSE IF bridge is build and a cavePlan is in inventory
    calculate superAgent’s path to cave
    superAgent performs action on cave
    hasTaskSuperAgent = true;

  ELSE IF summoningStone is in inventory
    calculate superAgent’s path to summoningShrine
    superAgent performs action on summoningShrine
    hasTaskSuperAgent = true;

  ELSE IF fireImp was summoned and sword is not in inventory
```

43
calculate fireImp’s path to sword
fireImp performs action on sword
hasTaskFireImp = true;

calculate superAgent’s path to dragon
hasTasksuperAgent = true;

ELSE IF sword is in inventory
calculate superAgent’s path to dragon if he is not there yet
superAgent performs action on dragon
hasTasksuperAgent = true;

FOR iterating over list of all keyFragments on the Map
IF keyFragment is already discovered and not collected
    calculate superAgent’s path to keyFragment
    superAgent performs action on keyFragment
    hasTasksuperAgent = true;

IF all keyFragments are in inventory
    calculate superAgent’s path to treasureChest
    superAgent performs action on treasureChest
    hasTasksuperAgent = true;

ELSE IF superAgent has no current task
    plan superAgent’s path to next undiscovered field

ELSE IF fireImp has no current task
    plan fireImp’s path to next undiscovered field

FOR list of path directions
    IF superAgent has steps left for this turn
        walk along the path

FOR list of path directions
    IF fireImp has steps left for this turn
        walk along the path

Designing an AI that can compete with human planning in a turn-based strategy game is quite complex, even for a small-sized game like the Treasure Hunt game. The AI that is integrated in this version of the game is quite simple and subsequently does not always find the perfect solution. This issue will be further addressed in the section SuperAgentHuman versus MultiAgentHuman versus SuperAgentAutonomous versus MultiAgentAutonomous, where the performance of all four ways to start the game are compared to each other.

6.4.3. Further Improvements

Although the game is finished for now, there are several upgrades that can be made to further improve and expand the game.

For a start, it would be very convenient to reorganise the part of the game that is responsible for the tiled map and the collision detection. At the present time, Tiled and the resulting map are only used as level editor, e.g. creating the graphic background of the playing field.
Collision detection between the agents and natural obstacles, like the river or the lava fields, as well as barriers, e.g. the cliffs surrounding the playing field, is currently implemented by adding the respective fields to a list of so-called "occupiedFields". Every movement of one of the agents is checked against this list to evaluate, if this action is valid. And while this is an acceptable strategy if the game only provides one level, e.g. one map, it is inadequate for a game which makes it possible to choose between varying levels and maps.

Tiled offers two different possibilities to create maps which already include additional information like collision areas. For one thing, there is the option to add a so-called "Object Layer" to your map, where one can plot all collision areas by adding objects like rectangles, polylines etc. to the layer.

For another thing, and this is the more convenient technique for this example, one can assign additional information to the used tile sets. As you can see in figure Adding information to tiles, you can select one or several tiles from a tile set and assign them any number of properties, for instance adding the property "impassable" to the selected rock tile. This properties can later be retrieved and processed further in the program, thus making it possible to employ it for collision detection.

Another improvement for the game, which also uses the aforementioned method, is adding animated details to the map. This would cause the playing field to be livelier and richer in detail, therefore enhancing the gaming experience. To achieve this, one would add a property to the fields which shall be animated and then, by sampling them out later in the code, superimposing them with additional frames.

The insertion of a loading screen between the menu screen and the actual game screen would also be an improvement. At the moment, the game stops at the transition between the two screens for a few second, which is slightly annoying. This lagging is caused by the process that loads all the assets into the game. A solution would be to introduce LibGDX’s AssetManager, which loads all assets when prompted, and that is also very useful for implementing a loading screen with a loading bar, which then would be displayed instead of a frozen menu screen.

There are also a couple of other minor enhancements concerning graphics and game play that would be necessary to hone the overall image:

- The so-called "Fog of War", which is used in many games to obscure the, at this time unexplored, territory, can be improved. Rather than the current square-shaped opaque form, a circular shape with a graduating transparency would be more pleasant to look at.

- Also, the agents could be designed to be livelier, e.g. by adding further animations and gestures, like waving, sleeping or yawning, for the time when they are unoccupied. Likewise, adding small details like personalised voices or short verbal responses
to occurring events. The agents are also not animated if they fulfil task like cutting
wood or digging which would however require superior and more extensive sprite
sheets than those that are currently available.

- Lastly, the map could be embellished by using more extensive and detailed profes-
sional tile sets.

The game also requires further optimisation regarding the performance aspect. Whereas
it does not pose a problem to execute it on a PC, it is, in its present form, too resource-
intensive to run it without stuttering as a HTML5 application.

An additional point for improving the game would be to introduce the possibility for
the player to define the number of agents, their skill sets and the configuration of the
map. It would also be a nice feature to combine the human and autonomous game play
to create the possibility to compete with an opponent for the treasure.

An upgrade concerning the design issue would be to decentralise the planning procedure
from the current central platform to each agent, thus increasing their autonomy.

6.5. **SuperAgentHuman versus MultiAgentHuman versus SuperAgentAutonomous versus MultiAgentAutonomous**

In this section an evaluation between the four different ways to play the Treasure Hunt
game will take place. Also, the results of the experiment will be analysed and connected
with each other.

To differentiate this four game styles and the following terminology, the following list will
once more introduce the concepts:

- **SuperAgentHuman, referred to as SAH**
  The player controls a single SuperAgent that combines all resources required to
  reach the goal.

- **MultiAgentHuman, referred to as MAH**
  A team of multiple agents with different skill sets is controlled by the player.

- **SuperAgentAutonomous, referred to as SAA**
  The SuperAgent fulfils its actions planned by the game autonomously, player input
  is only required to start the game and to proceed to the following turns.

- **MultiAgentAutonomous, referred to as MAA**
  The game plans the actions of a multiagent system that strives to retrieve the
  treasure. Player input is again needed for starting the game and initiating the next
  turn.

To compare SuperAgent performance to MultiAgent performance on the one hand, and
Human performance and Autonomous performance on the other hand, each of the pro-
grams was executed and the number of turn required to reach the goal was noted. The
following figure Evaluation of the game types and table SuperAgentHuman versus MultiAgentHuman versus SuperAgentAutonomous versus MultiAgentAutonomous illustrate the result of this comparisons.

![Graph showing the evaluation of game types](image)

**Figure 6.5.: Evaluation of the game types**

<table>
<thead>
<tr>
<th></th>
<th>SAH</th>
<th>MAH</th>
<th>SAA</th>
<th>MAA</th>
</tr>
</thead>
<tbody>
<tr>
<td>N = 1</td>
<td>45</td>
<td>35</td>
<td>84</td>
<td>63</td>
</tr>
<tr>
<td>N = 2</td>
<td>50</td>
<td>46</td>
<td>90</td>
<td>62</td>
</tr>
<tr>
<td>N = 3</td>
<td>53</td>
<td>45</td>
<td>86</td>
<td>72</td>
</tr>
<tr>
<td>N = 4</td>
<td>48</td>
<td>42</td>
<td>91</td>
<td>59</td>
</tr>
<tr>
<td>N = 5</td>
<td>56</td>
<td>60</td>
<td>97</td>
<td>68</td>
</tr>
<tr>
<td>N = 6</td>
<td></td>
<td></td>
<td>88</td>
<td>65</td>
</tr>
<tr>
<td>N = 7</td>
<td></td>
<td></td>
<td>85</td>
<td>64</td>
</tr>
<tr>
<td>N = 8</td>
<td></td>
<td></td>
<td>89</td>
<td>70</td>
</tr>
<tr>
<td>N = 9</td>
<td></td>
<td></td>
<td>94</td>
<td>66</td>
</tr>
<tr>
<td>N = 10</td>
<td></td>
<td></td>
<td>92</td>
<td>64</td>
</tr>
</tbody>
</table>

**Table 6.2.: Evaluation of the game types**

After providing the results of the test, they will be analysed in regard to their performance. Firstly, to introduce the manner in which the results were derived, the test procedure will be explained.
Chapter 6. Implementation and Evaluation

The SAA and the MAA where evaluated by assigning random positions to the agents and then counting the number of turns the Treasure Hunt AI required to complete the game. As the planning AI implemented for this autonomous game play has been kept on a very simple level, it is quite easy for a player to undercut the autonomous performance by achieving a lower turn count.

For the evaluation of the SAH and the MAH, five different test persons played each a SAH and a MAH game. Initially, it was considered to shortly introduce the game, the control and the plot to the test persons which would then start the game and try to reach the treasure. This approach was selected to prevent the occurrence of a learning effect. It soon became evident that the game is not intuitive enough to play to develop a strategy and realising it in a successful manner. This effect would distort the comparison between human and autonomous gameplay, because the autonomous planner has much more information about the game and its process than the human. Therefore, the human test persons could play the SAH game a first time without integrating this results into the evaluation. This results far exceeded the SAA turn count, because the test persons were not familiar enough with the game control and the plot, and were not able to develop a strategy.

So after finishing a test run with a SAH game, the real experiment took place. Each test person played one game of MAH and following this, a game of SAH.

The following table displays the SuperAgentHuman versus MultiAgentHuman versus SuperAgentAutonomous versus MultiAgentAutonomous. For each of the N = 10 (in SAA and MAA) or N = 5 (in SAH and MAH), i calculated the average count of turn and determined the maximum and minimum turns that were needed to achieve the goal.

<table>
<thead>
<tr>
<th></th>
<th>Ø</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAH</td>
<td>50.4</td>
<td>45</td>
<td>56</td>
</tr>
<tr>
<td>MAH</td>
<td>45.6</td>
<td>35</td>
<td>60</td>
</tr>
<tr>
<td>SAA</td>
<td>89.6</td>
<td>84</td>
<td>97</td>
</tr>
<tr>
<td>MAA</td>
<td>65.3</td>
<td>59</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 6.3.: Analysis of the results

As visible in table SuperAgentHuman versus MultiAgentHuman versus SuperAgentAutonomous versus MultiAgentAutonomous, there exist significant differences between the autonomous and the human performance as well as the Superagent and Multiagent performance.

- **SAH versus MAH**
  When comparing the SAH and the MAH, it becomes apparent that the SAH needed 4.8 additional turns to reach the goal:

  \[
  SAH_\emptyset - MAH_\emptyset = 50,4 - 45,6 = 4,8
  \]

  Although by comparing the MIN turn count, it can be determined that this comparatively low difference is determined by the performance of the test person playing
the game. A test person with a sophisticated strategy can increase the performance of both SAH and MAH as well as increase the difference between the turn count:

\[ SAH_{MIN} - MAH_{MIN} = 45 - 34 = 10 \]

Therefore, with a good player, the performance of the multiagent systems exceeds by far that of the SuperAgent.

- **SAA versus MAA**
  The difference in performance between multiagent systems and a SuperAgent is more visible when comparing the SAA and the MAA:

\[ SAA_\Omega - MAA_\Omega = 89,6 - 65,3 = 24,3 \]

The SAA requires 24.3 additional turns to achieve the treasure compared to the MAA and has a difference of 25 turns between the respective MIN and MAX:

\[ SAA_{MIN} - MAA_{MIN} = 84 - 59 = 25 \]

\[ SAA_{MAX} - MAA_{MAX} = 97 - 72 = 25 \]

Part of this very high difference can be explained by the games AI: planning for a SuperAgent is much simpler than planning for multiple agents; the planner for multiple agents may be not advanced enough to compensate for that. Nonetheless, a significant performance difference between SAA and MAA can be determined.

- **SAH versus SAA**
  The performance difference between SAH and SAA is, as already mentioned, in part caused by the AI that does not always find the perfect solution for processes whereas a human player can plan its strategy very accurately.

\[ SAA_\Omega - SAH_\Omega = 89,6 - 50,4 = 39,2 \]

Therefore, a SAA requires Ø39.2 turns more compared to a SAH.

- **MAH versus MAA**
  The same issue from above, concerning the AI, also applies to this situation.

\[ MAA_\Omega - MAH_\Omega = 65,3 - 45,6 = 19,7 \]

A MAA requires Ø19.7 additional turns to open the treasure chest in comparison to a MAH.
Chapter 7. Conclusion

7. Conclusion

This bachelor thesis provided an overview over the topic of intelligent agents and describes strategies for the process of collaborative distributed problem solving. This process depends on the involved agents, the environment and the communication within the multiagent system. The planning and the synchronisation between agents actions are important elements, as well as the mechanisms to allocate tasks and resources to agents.

The last chapter of the thesis was dedicated to the Treasure Hunt game that I implemented. During the planning and implementation of the Treasure Hunt game I repeatedly noticed aspects of the game that would require improvements or a redesign. It is suffice to say that the more I immersed myself in the topic and in the features of LibGDX, the more starting points for improvement were discovered.

I would even go so far as to say that a personally created game like the Treasure Hunt game may be never finished completely, there are always new ideas that could be integrated. The most important of this ideas were already introduced in the thesis.

The evaluation of the results turned out as anticipated: The system of multiple agents had an advantage over the expert single agent, although this agents skills were always superior to other agents. It also became obvious that the present AI has to be improved to pose a challenge for the player.
Bibliography


[MF09] MILLINGTON, Ian; FUNGE, John: Artificial Intelligence for Games. 2. CRC Press, 2009


[SAA+06] SUN, Ron; ADAMS, William; ANDERSON, John; BEST, Brad; BOTHELL, Dan J.; BROCK, Derek P.; BRODSKY, Boris; BUGAJSKA, Magdalena D.; BURNS, Tom R.; CASSIMATIS, Nicholas L.; CASTELFRANCHI, Cristiano; CLANCEY, William J.; DAMER, Bruce; GILBERT, Nigel; GRATCH, Jonathan; HIATT, Laura M.; JENNINGS, Nicholas R.; JONES, Randolph M.; LEBIERE, Christian; MAHESWARAN, Rajiv T.; MAO, Wenji; MARSELLA, Stacy; MATARIC, Maja J.; MOSS, Scott; NAVEH, Isaac; NOLFI, Stefano; NORLING, Emma; OKAMOTO, Steven; PANZARASA, Pietro; PARISI, Domenico; PERZANOWSKI, Dennis; RITTER, Frank E.; ROSZOWSKA, Ewa; SCERRI, Paul; SCHULTZ, Alan C.; SCHURR, Nathan; SHELL, Dylan A.; SIERHUIS, Maarten; TAATGEN, Niels; TAMBE, Milind; TRAFTON, J. G.; WEST, Robert L.; WRAY, Robert E.; SUN, Ron (Hrsg.): Cognition and Multi-Agent


A. Java-Quellcodes

A.1. Animation

Listing A.1: Creation of animations based on an image/sprite sheet

```java
/** Takes an image file and creates sequences for animation
 * @param image = Image file containing multiple images for animation */
protected void createAnimations(Texture image) {
    TextureRegion[][] walkUp = new TextureRegion[4];
    TextureRegion[][] walkDown = new TextureRegion[4];
    TextureRegion[][] walkRight = new TextureRegion[4];
    TextureRegion[][] walkLeft = new TextureRegion[4];

    // Splitting the image file in TextureRegions according to movement direction
    TextureRegion[][] tmp = TextureRegion.split(image, image.getWidth() / 4, image.getHeight() / 4);
    for (int i = 0; i < 1; i++) {
        for (int j = 0; j < 4; j++) {
            walkDown[j] = tmp[i][j];
        }
    }
    for (int i = 1; i < 2; i++) {
        for (int j = 0; j < 4; j++) {
            walkLeft[j] = tmp[i][j];
        }
    }
    for (int i = 2; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            walkRight[j] = tmp[i][j];
        }
    }
    for (int i = 3; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            walkUp[j] = tmp[i][j];
        }
    }

    actualFrame = walkDown[0];

    // Animations with display time and TextureRegions
    animationWalkUp = new Animation(0.125f, walkUp);
    animationWalkDown = new Animation(0.125f, walkDown);
    animationWalkRight = new Animation(0.125f, walkRight);
    animationWalkLeft = new Animation(0.125f, walkLeft);
}
```

Listing A.2: Drawing the animations
/** Provides the frame for render() in GameScreen *
 * @param batch
 */
@Override
public void draw(Batch batch, float alpha) {
    time += Gdx.graphics.getDeltaTime();

    if (walkingX == 0 && walkingY == 0) {
        walkingFinished = true;
        state = State.STANDING;
    }

    // Depending on state the frame which should be drawn is determined
    switch (state) {
    case WALKINGDOWN:
        actualFrame = animationWalkDown.getKeyFrame(time, true);
        if (walkingY < 0) {
            batch.draw(actualFrame, walking.getPixelXCoordinate(), walking.
                getPixelYCoordinate() - 32 - walkingY);
            walkingY++;
        }
        break;

    case WALKINGUP:
        actualFrame = animationWalkUp.getKeyFrame(time, true);
        if (walkingY > 0) {
            batch.draw(actualFrame, position.getPixelXCoordinate(), walking.
                getPixelYCoordinate() + 32 - walkingY);
            walkingY--;
        }
        break;

    case WALKINGLEFT:
        actualFrame = animationWalkLeft.getKeyFrame(time, true);
        if (walkingX < 0) {
            batch.draw(actualFrame, walking.getPixelXCoordinate() - 32 -
                walkingX, walking.getPixelYCoordinate());
            walkingX++;
        }
        break;

    case WALKINGRIGHT:
        actualFrame = animationWalkRight.getKeyFrame(time, true);
        if (walkingX > 0) {
            batch.draw(actualFrame, walking.getPixelXCoordinate() + 32 -
                walkingX, walking.getPixelYCoordinate());
            walkingX--;
        }
        break;

    case STANDING:
        batch.draw(actualFrame, position.getPixelXCoordinate(), position.
            getPixelYCoordinate());
        break;
    }
}
B. Features of LibGDX

List of LibGDX-features taken from the LibGDX Website [Liba]:

B.1. Graphics

- Rendering through OpenGL ES 1.x and 2.0 on all platforms
- Custom OpenGL ES 2.0 bindings for Android 2.0 and above
- Low-Level OpenGL helpers:
  - Vertex arrays and vertex buffer objects
  - Meshes
  - Textures
  - Framebuffer objects (GLES 2.0 only)
  - Shaders, integrating easily with meshes
  - Immediate mode rendering emulation
  - Simple shape rendering
  - Automatic software or hardware mipmap generation
  - ETC1 support (not available in Javascript backend)
  - Automatic handling of OpenGL ES context loss. Restores all textures, shaders and other OpenGL resources
- High-level 2D APIs:
  - Custom CPU side bitmap manipulation library
  - Orthographic camera
  - High-performance sprite batching and caching, handling OpenGL ES 1.x and 2.0 differences transparently
  - Texture atlases, with whitespace stripping support. Either generated offline or online
  - Bitmap fonts (does not support complex scripts like Arabic or Chinese). Either generated offline or loaded from TTF files (unsupported in Javascript backend)
  - 2D Particle system
  - TMX tile map support
  - 2D scene-graph API
  - 2D UI library, based on scene-graph API, fully skinable
- High-Level 3D APIs:
– Perspective camera
– Decal batching, for 3D billboards or particle systems
– Basic loaders for Wavefront OBJ and MD5
– Work in progress: 3D rendering API with materials and lighting system and support for loading FBX models via fbx-conv

B.2. Utilities

• Custom collections, with primitive support
• Json writer and reader, with POJO (de-)serialization support
• Xml writer and reader

B.3. Tools

• Gdx Setup UI, for easy project setup
• Particle editor
• Texture packer
• Bitmap font generator

B.4. Audio

• Streaming music and sound effect playback for WAV, MP3 and OGG
• Direct access to audio device for PCM sample playback and recording (unsupported in Javascript backend)
• Decoders for OGG and MPG3 formats (unsupported in Javascript backend)
• Pitch shifting, time stretching and playback rate modification (unsupported in Javascript backend)

B.5. Input Handling

• Abstractions for mouse and touch-screen, keyboard, accelerometer and compass
• Gesture detector, detects taps, panning, flinging and pinch zooming
• Gdx remote, to control your desktop application via your Android phone. Useful to test multi-touch gestures while debugging your desktop application
B.6. Math & Physics

- Matrix, vector and quaternion classes. Matrix and vector operations are accelerated via native C code where possible
- Bounding shapes and volumes
- Frustum class, for picking and culling
- Catmull-Rom splines
- Common interpolators
- Concave polygon triangulator
- Intersection and overlap testing
- JNI wrapper for Box2D physics. So awesome, other engines use it as well
- JNI Wrapper for bullet physics

B.7. File I/O & Storage

- File system abstraction for all platforms
- Read-only file system emulation for Javascript backend
- Binary file support for Javascript backend
- Preferences for lightweight setting storage
C. Sounds

This game uses these sounds from Freesound.org\(^1\):

- atmosphere 2 by ERH (http://www.freesound.org/people/ERH)
- meadow ambience by eric5335 (http://www.freesound.org/people/eric5335)
- walking_through_forest_2007_04_15 by reinsamba (http://www.freesound.org/people/reinsamba)
- Cat Meow by TheGertz (http://www.freesound.org/people/TheGertz)
- Hmm Ahh, Hmm question by esperar (http://www.freesound.org/people/esperar)
- cutting_tree by fkurz (http://www.freesound.org/people/fkurz)
- Buffer Spell by Northern_Monkey (http://www.freesound.org/people/Northern_Monkey)
- Fire-breathing dragon by CGEffex (http://www.freesound.org/people/CGEffex)
- DRAGON_ROAR by JoelAudio (http://www.freesound.org/people/JoelAudio)
- dragonhiss by NoiseCollector (http://www.freesound.org/people/NoiseCollector)
- Wood_Knocks by RutgerMuller (http://www.freesound.org/people/RutgerMuller)
- sword 02 by nextmaking (http://www.freesound.org/people/nextmaking)
- sword-01 by audione (http://www.freesound.org/people/audione)
- success_low by grunz (http://www.freesound.org/people/grunz)
- 03383 shovel ground dig by Robinhood76 (http://www.freesound.org/people/Robinhood76)
- success 1 by fins (http://www.freesound.org/people/fins)
- Chest Opening by spookymodem (http://www.freesound.org/people/spookymodem)
- Skyrim Stinger by thecluegeek (http://www.freesound.org/people/thecluegeek)

---

\(^1\)Freesound.org: http://www.freesound.org/
Appendix D. Contents of DVD

D. Contents of DVD

- PDF version of the bachelor thesis: BachelorThesis_KaemmererT.pdf
- Runnable Jar of the Treasure Hunt Game: TreasureHunt.jar
- Zipped version of the LibGDX project: TreasureHunt.rar
Erklärung

Ich erkläre hiermit gemäß §17 Abs. 2 APO, dass ich die vorstehende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

______________________________  ______________________________
Ort, Datum                                     Unterschrift