Institut für Parallele und Verteilte Systeme (IPVS)

Abteilung Maschinelles Lernen und Robotik (MLR)

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Studienarbeit Nr. 2464

Development of a benchmarking framework for Inverse Reinforcement Learning algorithms based on Tetris

Pascal Bock

Studiengang:	Technische Kybernetik
Prüfer:	Prof. Dr. rer. nat. Marc Toussaint
Betreuer:	M. Sc. Peter Englert
begonnen am:	01.09.2015
beendet am:	03.03.2015
CR-Klassifikation:	D.2.8, H.5.2, I.2.1, I.2.6

Abstract

Tetris is one of the oldest, most popular and most well-known video games. The simple rules and scoring options make it a viable choice for benchmarking artificial intelligence, especially in the machine learning department. This work describes a customizable benchmarking framework using a simplified variant of the original Tetris game focused on Inverse Reinforcement Learning algorithms.

Contents

1 Introduction

During early research for this work, one significant complication was the limited comparability of different algorithms and their performance, especially those in the field of Inverse Reinforcement Learning (IRL). Many authors do their best trying to decribe their algorithm's strengths and weaknesses, but even then, comparing different approaches is a tedious and complex task. This problem led to the decision to focus this study thesis on creating an independent framework to assess the performance of IRL algorithms by calculating a score based on how well the respective approach solved a specific, ideally rather simple task.

One of the most defining influences on the direction of this work is [RGB10], which applies advanced IRL techniques to several classic video games. Given appropriate adaptation and simplification, video games can provide such a rather simple task and meet the requirements of this framework. Additionally, since gaming as a tool for learning is a very effective method for humans (as can be seen in very young children, who learn nearly everything through playing), applying that method to artifical intelligence can provide much insight into the relation between machine learning and human learning. Of course, not every game is suitable for this specific application, and video games are exceptionally well suited due to their implicit structural similarity to algorithms and programs. Choosing a video game from the time when they were not as widespread as they are today would ensure a concept that is sufficiently simple to be incorporated into a benchmarking framework without going beyond the scope of this task. Some of the games taken into account as potential candidates were Pong, a simple ping pong simulation, Super Mario, a jump and run game, The Binding of Isaac, a modern retro style shooting game, Space Invaders, a very simple shooting game, and Tetris, a puzzle game. As the title of this work suggests, the latter was chosen in the end.

Tetris, one of the most well-known video games of all time, has been very popular as a benchmark for many different algorithms and methods related to Artificial Intelligence, both in the classical sense ([Gie11], [Fah12]) and the learning department, ranging from evolutionary approaches (e.g. [BKM05]) to classical Reinforcement Learning and related optimization techniques ([SL06], which also lists several other approaches between those extremes). However, its application to IRL is not documented in any of the sources found during research for this thesis.

The framework described in this study thesis is built with modular design in mind, every part of it is replaceable to fit the user's exact needs, although it is advised to make changes only to the algorithm in order to retain comparability, which was the main motivation in developing this framework.

Outline

This thesis is structured as follows:

Chapter 2 – Related work introduces related work and discusses some of it in detail with regards to its influence on this work at different stages.

Chapter ?? – ?? describes the framework developed as well as the interface for creating and testing algorithms with it.

Chapter ?? – ?? outlines the development process and possible future work.

2 Related work

In this chapter, related work and its influence on this work will be discussed. The different stages of development will be described in chronological order, also detailing the development of this work's focus, which has changed over time.

2.1 Early research

Originally, the focus of this thesis was intended to be the preparation of the development of a framework, which itself would be part of a future work. As such, part of this preparation was researching and comparing different algorithms and approaches in the field of IRL and the starting point for this research was the survey of IRL techniques [SE12], complemented by [HYL08] as a general introduction to Machine Learning. This gave an overview over what IRL as a field of study has achieved so far, as well as a general idea of how the different approaches worked.

Following the general overview of state of the art IRL, a partial framework was to be conceived, focusing on one specific, but arbitrarily chosen approach evaluated with a simple toy example as a starting point, later expanding the scope to a more complex problem setting so as to obtain a strong foundation for a future benchmarking framework that would expand these basic tests into a standalone piece of software. The field of this complex problem was decided to be gaming, mainly because of the similarity to human learning mentioned in the previous chapter.

The algorithms and techniques that were most interesting for this task were the ones able to offset limitations in the learning set, avoiding traps and mistakes done by the demonstrator, possibly even outperforming the demonstrator in the long run. These can be grouped into two categories, one of which uses the underlying structure of the problem, such as the approach described in [ML10], which abuses the structure of the Markov Decision Problem (MDP). The other category acts autonomously as far as possible and adapts to information gained while learning, thus finding ways itself from time to time instead of simply following the expert. An example of this approach is described in [ML11], where an autonomous unit adapts to the user and personalizes his experience gradually while it gets to know the user. Other examples of the second category are described in the next section.

2.2 Gaming and Active Learning

The main influence for the decision towards gaming was [RGB10], in which the authors develop a no regrets online learner that plays adaptations of the jump and run game “Super Mario” as well as the racing game “Mario Kart”. The basis of their approach is to reduce the size of the original learning set while offering the algorithm a way to request assistance from the expert or demonstrator in any state. Compared to other approaches used with a learning set based on expert demonstrations created by playing through an entire game, the notable advantage of this approach is the reduction of unknown states and ability to recover from mistakes. A typical offline learner working with a learning set made by an expert might make a mistake and come into a game state where the expert never ended up, for example standing right in front of an obstacle, which the expert would avoid by changing his trajectory long before reaching the obstacle. Once the learner does become blocked by an obstacle, it has no way of freeing itself because the learning set doesn’t account for this situation. If the learner can request assistance from the expert, however, it can learn how to unblock itself, significantly raising its performance in those situations.

Related to this approach is what [SE12] calls “Active learning”. Detailed, among others, in [LMM09], this approach has the main purpose of reducing the amount of samples necessary to reach good performance, which is achieved by complementing the arbitrary samples provided in advance with specific samples requested from the expert. This is the same basic idea as the no regrets online learner, but instead of requesting assistance once the algorithms fails, the ideal states to query are calculated such that the information gain from them is optimal, thus minimizing the amount of queries sent to the expert.

Following this line of thought, [CMDS10] develops a method to determine as efficiently as possible when to request information and when to act autonomously, introducing the concept of Expected Myopic Gain, which is basically a measure of long-term value gain a specific query adds, not necessarily limited to finding the optimal action for the current state, but rather geared towards understanding the MDP better, or in other words, learning what the problem being solved is exactly.

2.3 Tetris

At this point of the research, it was clear the algorithm would be some variation of active learning. What still needed a decision, was the game the framework would make the algorithm play. After evaluating the candidates mentioned above, Tetris was found to be the best suited one, so research on details about this game started, using the community driven Tetris Wiki [Var15] as a starting point. The framework to be developed herein was assigned the working title "Tetrisframe", which will be used in this thesis from this point forward to refer to the software developed as part of the study thesis. One of the most concise and complete collections of information about Tetris, its history and details about its internal structure has been assembled by Colin Fahey [Fah12], who also contributed an early AI program playing Tetris, which he describes in the same article. Most of the technical details of Tetrisframe are based on that article. Design choices are based both on [Fah12] and [Var15] as well as personal preferences.

It soon became clear that The MathWorks' MATLAB was the ideal platform to develop Tetrisframe on, being widely used in both research and industrial applications because of its highly efficient data handling capabilities. MathWorks offers a service called File Exchange where users can upload and share scripts, functions and code snippets developed for their MATLAB software. This is where the projects [Gie11], [Ald08] and [Fig12] originated, all of which served as inspiration and helpful resources while creating Tetrisframe. By far the most influential of these was [Gie11], mainly because it wasn't as complex as [Fig12], which incorporates many cosmetic upgrades with no relevant function besides looking better. [Gie11] also includes a hard coded AI behaviour and is thus geared towards programmatically simple structures. Most notably, it provided the basic data structure and Tetrimino moving algorithms used in Tetrisframe.

3 Tetrisframe

This chapter gives an overview of the Tetrisframe framework and describes how to build an algorithm for use with Tetrisframe. Tetrisframe does not include a default algorithm as adapting an algorithm and including it goes beyond the scope of this study thesis. More about this can be found in the last chapter.

The complete source code of Tetrisframe is included in the appendix. It is free to use and modify as long as the original author is referenced and any modifications are marked as such. The following is an overview over the program, how it works and how the soruce code is organized. Each section in this chapter refers to a section in the source code, which is written in MATLAB's cell mode, thus coloring a section in light yellow when the cursor is placed within that section.

3.1 Constants and base variables

This section contains all constants and generally used variables excluding local variables of nested functions and highly volatile or temporary variables. One of these variables is the "blocks" variable, which contains all blocks to be used. Replace this variable or remove blocks from it to make the game easier.

3.2 Rotation system

The rotation system defines what the Tetriminoes look like in each of their rotational states. The system included in the original Tetrisframe package is based on the so-called "Original Rotation System (ORS)", simplified to fit the needs of the framework. Thus, all rotational states start out with a block in the topmost row, which is where they would spawn if the respective rotation was their starting rotation. In ORS, blocks always spawn with the longest solid line in the topmost row, but some rotational states are not technically possible in that position and rotating the pieces would be impossible until they drop at least 1 space. This behaviour is disregarded in this version.

Tetrisframe assumes a virtual player skilled enough to move and rotate pieces at arbitrary speed. The only limitation is given by unreachable states. Thus, the way of reaching a certain final state for each piece is neglected and the algorithm concentrates on deciding which one of the possible final states to select. Reachable states are calculated by the Rotation system module with the function “makemoves”, which updates the “moves” variable. Each move has a corresponding position and rotation, which are used to index into the variable.

The “makeblock” function takes as input an integer from 1 to 7, encoded with the “Tetr” structure seen in the previous section of the source code. This structure assigns an identifier to each of these integers so that a block can be (and is in all of Tetrisframe’s source code) referenced as, e.g., “Tetr.I”. The function then returns a 20x10xX “block” matrix which holds the starting position of each rotational state of the respective block. X is the number of rotational states the block has and rotating it is merely a matter of indexing in the third dimension.

3.3 Mainframe

The Mainframe contains the functions needed to calculate resulting states from the current field and current piece. The functions are moving and rotating as well as dropping a piece and clearing lines to determine the new score and final field state. The Mainframe also contains the “gameover” function that handles a state where the next block cannot spawn because the field is blocked. These functions are being called by internal functions and are not supposed to be called by the algorithm or related functions directly. The entire Mainframe section is thus considered private (although for technical reasons it’s not actually handled as being private by MATLAB).

3.4 Graphical User Interface

This section generates what the user sees and uses to do anything. The GUI supports different features: creating learning sets, loading a learning set, and running an algorithm (which may or may not request user input to complement its learning set). Figure 1 on the next page shows the GUI upon starting Tetrisframe. Some of the button labels change when clicked to reflect their current function. The area to the left is used to inform the user about things he needs to do or things that happened. The area on the right is where the game is going to happen. Ticking the “Show AI playing” checkbox will update the field after each move the algorithm executes, while leaving it unticked will make only the score update.

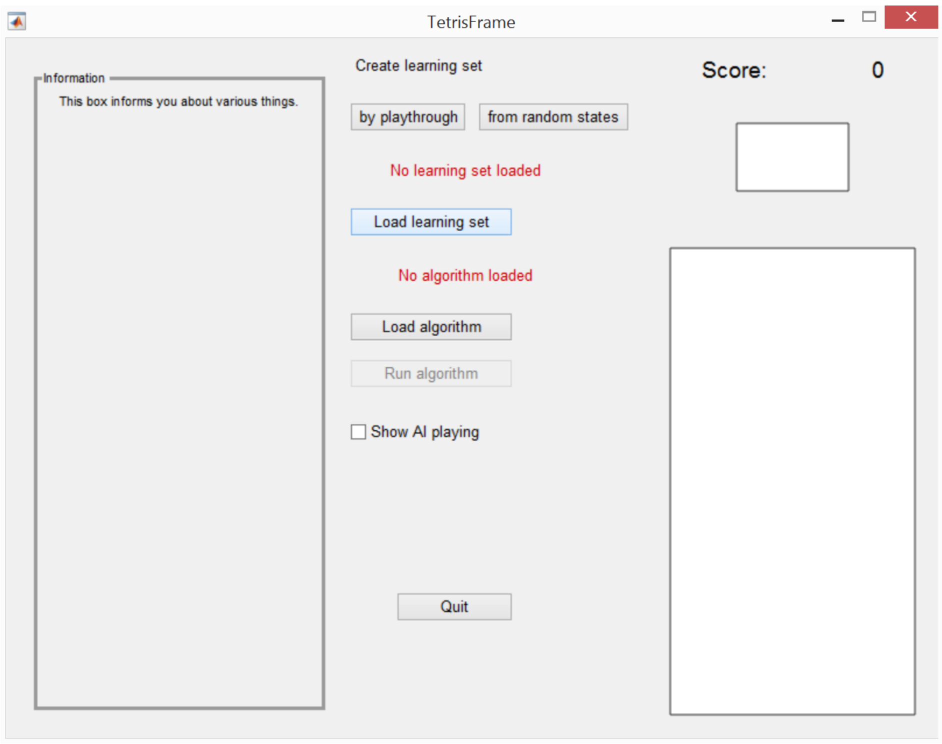

Figure 3.1: This is what Tetrisframe looks like when first started.

3.5 GUI Callback functions

These handle the GUI's functionality. Whenever something is clicked, one of these callbacks is executed. Going into detail about how these callbacks work would serve little purpose as some understanding of MATLAB would be necessary to understand this anyway and with that understanding, the reader can easily understand the source code itself.

3.6 API

The API (application programming interface) contains the functions to be used by the algorithm to communicate with the framework and the user. The functions here are the only ones that should be used by the algorithm, all other functions in other sections of this file are considered private and are not intended to be called except by one another. Technically, the algorithm function itself cannot call these functions anyway. But they are they ones called by the button callback that executes the algorithm. How this API and the algorithm function affect each other is outlined in the last section of this chapter.

The API's "request" function initiates a request to the user. It displays the current field state and lets the user select the appropriate move from all possible moves. Note that in accordance with the scoring system of earlier Tetris games, which rewarded hard dropping Tetriminoes from as high as possible, Tetrisframe only considers moves where the Tetrimino is dropped from the very first line in order to maximize this theoretical score. The actual score measured by Tetrisframe award 1 point for the first cleared row and 2 points for each additional row cleared at the same time, incentivizing building up fields to clear several rows at once. The "request" function is used both by the algorithm in order to request additional learning data and Tetrisframe's learning set creation feature.

The "updatedisplay" function updates all relevant fields in the GUI including the field only if the user wants to see the AI playing and forces MATLAB to update the figure window and display the changes.

The "execute" function commits the currently selected move to the field and updates the field and score.

3.7 Helpers and Wrapper

This last section of the Tetrisframe source code contains usually small functions that help with minor recurring tasks or wrap a function to make is compatible to a specific task. They're all very easy to understand and the interested reader is therefore referred to the source code in the appendix.

3.8 The Algorithm template

In appendix B, the reader will find a template file for the algorithm. It is thoroughly commented and thus rather self-explanatory, which is why this section will only give a short overview of how it works inside the framework.

When the algorithm is run, the algorithm function is called once and returns handles to its nested functions “rewardfcn”, “updaterewardfcn” and “request”. The first one calculates the reward for a specific field state. Tetrisframe will then execute the move which leads to the field with the highest reward. The latter function is used to determine, based on the matrix of each move’s reward, whether to act autonomously or request an additional sample based on the current state. If a sample is requested, it’s added to the learning set and the second function is called, which returns the handle to the new, updated reward function.

4 Experiments

The development of Tetrisframe turned out to be very complex and time consuming, which made the additional adaptation and development of an example or default algorithm too much for the scope of this work. Before this was obvious, several algorithms were partially adapted, but technical limitations made it impossible to develop a working version within a reasonable amount of time.

Therefore, this section will only point out that future work can be done in this area.

Most of the algorithms in [NR$^+$00] were part of the failed attempts to create a working demonstration, as well as a variant of the active learning approach detailed in chapter 2, but ultimately, none of them worked and fixing them would have cost more time than was available for this study thesis.

A Tetrisframe

```
%% TETRISFRAME
% TETRISFRAME is a benchmarking framework for Inverse Reinforcement
% Learning algorithms based on the popular video game Tetris. The source
% code is easily adaptable to AI algorithms outside the realm of IRL.
% Running this function opens a GUI that displays various options to run
% base variables" contains definitions and may be adapted to change options
% like the difficulty level (which blocks can appear).
%
% (c) 2015 Pascal Bock
function tetrisframe

%% Constants and base variables
% This section contains all constants and generally used variables
% excluding local variables of nested functions and highly volatile or
% temporary variables.

% the framework and test the algorithms. The first section "Constants and
% The Tetr struct allows Tetriminoes to be referenced as, e.g., Tetr.I
   Tetr = struct('I',1,'O',2,'J',3,'L',4,'S',5,'Z',6,'T',7);

% difficulty: select all Tetriminoes that will be used
   blocks = [Tetr.I Tetr.O Tetr.J Tetr.L Tetr.S Tetr.Z Tetr.T];

% The following are variables that will be used later
   field = zeros(20,10); % starts out empty, first row is top
   score = 0; % the game state's current score
   scores = []; % tracks scores across multiple runs
   current = Tetr.I; % holds the Tetr identifier of the current block
   next = Tetr.I; % holds the Tetr identifier of the next/preview block
   block = zeros(20,10,4); % holds information about the current block
   rot = 1; % current rotation of current block
   algorithm = ''; % the function (file) containing the algorithm
   learnset = struct([]); % starts out empty, holds the learning set
   pos = 1; % current left/right position of selected move when requesting
   moves = cell(0); % will store possible moves in each step

   busy = false; % this flag is used to properly terminate
   cancel = false; % this flag is used to abort the running algorithm
   showaiplaying = false; % determines whether the display is updated
   learning = false; % determines whether a learnset is being built
   requesting = false; % determines whether user input is being requested
   randomls = false; % determines whether to give random examples or play
```

```
%% Rotation System
% The rotation system defines what the Tetriminoes look like in each of
% their rotational states. The system included in the original TETRISFRAME
% package is based on the so-called "Original Rotation System (ORS)",
% simplified to fit the needs of the framework. Thus, all rotational states
% start out with a solid block in the topmost row, which is where they
% would spawn if the respective rotation was their starting rotation. In
% ORS, blocks always spawn with the longest solid line in the topmost row,
% but some rotational states are not technically possible in that position
% and rotating the pieces would be impossible until they drop at least 1
% space. This behaviour is disregarded in this version.

% makeblock function
% input is a member of Tetr, output is a 20x10xR matrix defining the
% position of the current block relative to the field in each rotation. R
% is the number of rotational states that block has.
   function block = makeblock(blocktype)
      switch blocktype
         case Tetr.I
            miniblock(:,:,1) = [1 1 1 1;
                               0 0 0 0;
                               0 0 0 0;
                               0 0 0 0];
            miniblock(:,:,2) = [0 0 1 0;
                               0 0 1 0;
                               0 0 1 0;
                               0 0 1 0];
         case Tetr.O
            miniblock(:,:,1) = [0 1 1 0;
                               0 1 1 0;
                               0 0 0 0;
                               0 0 0 0];
         case Tetr.J
            miniblock(:,:,1) = [1 1 1 0;
                               0 0 1 0;
                               0 0 0 0;
                               0 0 0 0];
            miniblock(:,:,2) = [0 1 0 0;
                               0 1 0 0;
                               1 1 0 0;
                               0 0 0 0];
            miniblock(:,:,3) = [1 0 0 0;
                               1 1 1 0;
                               0 0 0 0;
                               0 0 0 0];
            miniblock(:,:,4) = [0 1 1 0;
                               0 1 0 0;
                               0 1 0 0;
                               0 0 0 0];
         case Tetr.L
            miniblock(:,:,1) = [1 1 1 0;
```

```
                     1 0 0 0;
                     0 0 0 0;
                     0 0 0 0];
        miniblock(:,:,2) = [1 1 0 0;
                     0 1 0 0;
                     0 1 0 0;
                     0 0 0 0];
        miniblock(:,:,3) = [0 0 1 0;
                     1 1 1 0;
                     0 0 0 0;
                     0 0 0 0];
        miniblock(:,:,4) = [0 1 0 0;
                     0 1 0 0;
                     0 1 1 0;
                     0 0 0 0];
    case Tetr.S
        miniblock(:,:,1) = [0 1 1 0;
                     1 1 0 0;
                     0 0 0 0;
                     0 0 0 0];
        miniblock(:,:,2) = [0 1 0 0;
                     0 1 1 0;
                     0 0 1 0;
                     0 0 0 0];
    case Tetr.Z
        miniblock(:,:,1) = [1 1 0 0;
                     0 1 1 0;
                     0 0 0 0;
                     0 0 0 0];
        miniblock(:,:,2) = [0 0 1 0;
                     0 1 1 0;
                     0 1 0 0;
                     0 0 0 0];
    case Tetr.T
        miniblock(:,:,1) = [1 1 1 0;
                     0 1 0 0;
                     0 0 0 0;
                     0 0 0 0];
        miniblock(:,:,2) = [0 1 0 0;
                     1 1 0 0;
                     0 1 0 0;
                     0 0 0 0];
        miniblock(:,:,3) = [0 1 0 0;
                     1 1 1 0;
                     0 0 0 0;
                     0 0 0 0];
        miniblock(:,:,4) = [0 1 0 0;
                     0 1 1 0;
                     0 1 0 0;
                     0 0 0 0];
end
```

```
      rots = size(miniblock,3);
      block = [zeros(4,3,rots) miniblock zeros(4,3,rots) ;
             zeros(16,10,rots)];
   end % function makeblock

% makemoves calculates the possible final positions of the current block
   function makemoves
      moves = cell(0); % reset moves container
      if anyone(block(:,:,1)+field > 1) % block cannot spawn
         gameover(); % might reset the field so it goes on
         if anyone(block(:,:,1)+field > 1) % still game over
            return;
         end
      end
      % loop through all rotations, setting the global rot so other
      % functions (moveblock, mainly) can use it properly
      for rot = 1:size(block,3) %#ok<FXUP>
         while moveblock() % move all the way left using fieldmask
            % this loop terminates when the current rotation block hits
            % the left edge of the field or a blocking field space
         end % the block is now as far left as possible
         pos = 1; % initialise position
         hitwall = false; % we didn’t hit the wall yet
         while ~hitwall % stop when we hit the wall
            [~,thismove] = drop; % drop and store final position
            if all(thismove(1,:)+thismove(20,:) < 2)
               moves{pos,rot} = thismove; % no overflow while dropping
            end
            while ~moveblock(false,false); % try moving right w/o mask
               if ~moveblock(false) % try moving right with mask
                  hitwall = true; % if both don’t work, it’s over
                  break; % because we hit the wall
               end % if it worked with mask, continue moving
            end % block now moved 1 step right, jumping over obstacles
            pos = pos+1; % increase pos
         end % if we didn’t hit the wall, loop
      end % otherwise go on with the next rotation
      pos = 1; % pick first position as initial selection
      rot = 1; % pick first rotation as initial selection
   end % function makemoves

%% Mainframe
% The Mainframe contains the functions needed to calculate resulting states
% from the current field and current piece. The functions are moving and
% rotating as well as dropping a piece and clearing lines to determine the
% new score and final field state.

% moveblock moves the current block with current rotation to either the
% left or the right. Optional input is the direction, where true is left
% and false is right. Returns true if moving did not result in hitting the
% wall. Checks for collision with block specific fieldmask. The flag
% forceMove determines whether movement is forced through a collision.
   function success = moveblock(isMoveLeft,useFieldMask,forceMove)
      if ~nargin
         isMoveLeft = true; % default is moving left
      end
      if nargin < 2
         useFieldMask = true; % default is using the fieldmask
      end
      if nargin < 3
         forceMove = false; % default is reverting impossible moves
      end
      fieldmask = field; % remove parts of the field to get the mask
      if useFieldMask % provided we're using it
         switch current
            case Tetr.I
               fieldmask(2:4,4:8) = 0;
            case Tetr.O
               % no fancy rotate-skipping obstacles with O
            case Tetr.J
               if rot == 2 % only Jrot2 is blocked by (3,8)
                  fieldmask(3,3:7) = 0;
               else
                  fieldmask(3,3:8) = 0;
               end
            case Tetr.L
               if rot == 4 % only Lrot4 is blocked by (3,3)
                  fieldmask(3,4:8) = 0;
               else
                  fieldmask(3,3:8) = 0;
               end
            case Tetr.S
               fieldmask(3,4:9) = 0;
            case Tetr.Z
               fieldmask(3,3:8) = 0;
            case Tetr.T
               fieldmask(3,3:8) = 0;
         end
      end
      success = true; % assume it's gonna work
      if isMoveLeft
         if any(block(:,1,rot)) % a space in the left col is occupied
            success = false; % hit the left wall
            return; % set error flag and stop execution
         end
         block(:,:,rot) = block(:,[2:10 1],rot);
         if anyone(block(:,:,rot)+fieldmask > 1) % collision
            success = false; % hit the fieldmask
            if ~forceMove
               block(:,:,rot) = block(:,[10 1:9],rot); % move back
            end
         end
      else % moving right
         if any(block(:,10,rot)) % a space in the right col is occupied
            success = false; % hit the right wall
            return; % set error flag and stop execution
         end
         block(:,:,rot) = block(:,[10 1:9],rot);
         if anyone(block(:,:,rot)+fieldmask > 1) % collision
            success = false; % hit the fieldmask
            if ~forceMove
               block(:,:,rot) = block(:,[2:10 1],rot); % move back
            end
         end
      end
   end % function moveblock

% drop returns the field state reached after dropping the current block
% with current rotation into the current field.
   function [newfield,droppedblock] = drop
      bottom = false; % flag for hitting the floor
      droppedblock = block(:,:,rot); % temporary copy of block
      while all(field+droppedblock < 2) % no spaces overlap
         if any(droppedblock(20,:)) % if the bottom row has elements
            bottom = true; % set the flag
            break; % stop dropping
         end
         % move last (empty) row to top => dropped block by one space
         droppedblock = droppedblock([20 1:19],:);
      end % terminates if either the block intersects the existing field
         % or drops onto the floor
      if ~bottom % unless bottom line was hit
         % revert last drop step: move first (empty) row to bottom
         droppedblock = droppedblock([2:20 1],:);
      end
      % insert dropped block into field
      newfield = field+droppedblock;
   end % function drop

% clearlines removes full rows and calculates score gain
   function [newfield,scoregain,rows] = clearlines(oldfield)
      if ~nargin
         oldfield = field;
      end
      rows = find(sum(oldfield,2) == 10); % find full rows
      numrows = numel(rows); % number of full rows to clear
      scoregain = max(0,2*numrows-1); % first row=1 point, rest=2 points
      oldfield(rows,:) = []; % delete full rows
      newfield = [zeros(numrows,10);oldfield]; % fill top with empty rows
   end % function clearlines

   function gameover()
      field = [0 1 1 1 0 0 1 1 1 0
```

```
             0 1 0 0 0 0 1 0 1 0
             0 1 0 1 0 0 1 0 1 0
             0 1 1 1 0 0 1 1 1 0
             0 0 0 0 0 0 0 0 0 0
             0 0 1 0 0 0 1 0 1 0
             0 1 0 1 0 0 1 0 1 0
             0 1 1 1 0 0 1 0 1 0
             0 1 0 1 0 0 0 1 0 0
             0 0 0 0 0 0 0 0 0 0
             0 1 1 1 0 0 1 1 1 0
             0 1 0 1 0 0 1 0 0 0
             0 1 0 1 0 0 1 1 0 0
             0 1 0 1 0 0 1 0 0 0
             0 0 0 0 0 0 1 1 1 0
             0 1 1 1 0 0 0 0 0 0
             0 1 0 0 0 0 1 1 1 0
             0 1 1 0 0 0 1 0 1 0
             0 1 0 0 0 0 1 1 0 0
             0 1 1 1 0 0 1 0 1 0];
      pos = 1; rot = 1; % reset moves for current display
      requesting = false; % block further user input
      scores = [scores;score]; % add score to tracker
      updatedisplay(true); % display the game over message
      if learning % learning by playthrough
         set(handles.createplay,'String','Continue','Callback',@cont);
         set(handles.createrand,'String','Finish','callback',@savels);
         inform('GOlearn');
         uiwait(handles.figure);
      else
         inform('GOrun');
         field = zeros(20,10); % reset field so it starts a new game
      end
   end % function gameover

%% Graphical User Interface
% This section generates what the user sees and uses to do anything.
% The GUI supports different features: creating learning sets, loading a
% learning set, and running an algorithm (which may or may not request user
% input to complement its learning set).

% The figure window
   screensize = get(0,'ScreenSize');
   handles.figure = figure('Resize','off','Toolbar','none',...
      'Menu','none','Name','TetrisFrame','NumberTitle','off',...
      'Units','pixels','CloseRequestFcn',@quitgui,...
      'WindowKeyPressFcn',@key,... % still wirks after clicking buttons
      'Position',[screensize(3)/2-400 screensize(4)/2-300 800 600],...
      'Visible','off'); % not visible until fully created

   set(0,'DefaultAxesYDir','reverse','DefaultAxesLineStyleOrder','s',...
      'DefaultAxesNextPlot','replacechildren','DefaultAxesBox','on',...
```

```
    'DefaultLineMarkerSize',18,'DefaultLineMarkerEdgeColor','k',...
    'DefaultLineMarkerFaceColor','r');

handles.ax = axes('Parent',handles.figure,'Units','pixels',...
    'Position',[571 21 210 400],'XLim',[0 11],'YLim',[0 21],...
    'XTick',[],'YTick',[],'NextPlot','add');

handles.preview = axes('Parent',handles.figure,'Units','pixels',...
    'Position',[628 470 96 58],'XLim',[3 8],'YLim',[0 3],...
    'XTick',[],'YTick',[]);

handles.scorelbl = uicontrol('Parent',handles.figure,'Style',...
    'text','Units','pixels','String','Score:','FontSize',14,...
    'Position',[596 561 60 25]);

handles.score = uicontrol('Parent',handles.figure,'Style','text',...
    'Units','pixels','String','0','FontSize',14,...
    'HorizontalAlignment','right','Position',[656 561 100 25]);

handles.createlbl = uicontrol('Parent',handles.figure,...
    'Style','text','Units','pixels','String','Create learning set',...
    'FontSize',10,'Position',[296 561 120 25]);

handles.createplay = uicontrol('Parent',handles.figure,...
    'Style','pushbutton','Units','pixels','String','by playthrough',...
    'Position',[296 521 100 25],'Callback',@createls,'FontSize',10);

handles.createrand = uicontrol('Parent',handles.figure,'Units',...
    'pixels','Style','pushbutton','String','from random states',...
    'Position',[406 521 130 25],'Callback',@createls,'FontSize',10);

handles.loadlbl = uicontrol('Parent',handles.figure,'Style','text',...
    'Units','pixels','String','No learning set loaded','FontSize',...
    10,'Position',[296 471 200 25],'ForegroundColor',[.8 0 0]);

handles.loadset = uicontrol('Style','pushbutton','Units','pixels',...
    'Parent',handles.figure,'String','Load learning set','Position',...
    [296 431 140 25],'Callback',@loaddat,'FontSize',10);

handles.alglbl = uicontrol('Parent',handles.figure,'Units','pixels',...
    'Style','text','String','No algorithm loaded','FontSize',...
    10,'Position',[296 381 200 25],'ForegroundColor',[.8 0 0]);

handles.loadalg = uicontrol('Style','pushbutton','Units','pixels',...
    'Parent',handles.figure,'String','Load algorithm','Position',...
    [296 341 140 25],'Callback',@loaddat,'FontSize',10);

handles.runalg = uicontrol('Style','pushbutton','Units','pixels',...
    'Parent',handles.figure,'String','Run algorithm','Position',...
    [296 301 140 25],'Callback',@runalg,'FontSize',10,'Enable','off');
```

```
   handles.showai = uicontrol('Style','checkbox','Units','pixels',...
      'Parent',handles.figure,'String','Show AI playing','Position',...
      [296 251 200 25],'Callback',@showai,'FontSize',10);

   handles.quit = uicontrol('Style','pushbutton','Units','pixels',...
      'Parent',handles.figure,'String','Quit','Position',...
      [336 101 100 25],'Callback',@quitgui,'FontSize',10);

   handles.infopnl = uipanel('Title','Information','Units','pixels',...
      'Parent',handles.figure,'Position',[25 25 250 550],...
      'BorderType','line','BorderWidth',3,'HighlightColor',[.6 .6 .6]);

   handles.infotxt = uicontrol('Parent',handles.infopnl,'Style','text',...
      'Units','pixels','Position',[11 11 224 516],'String',...
      'This box informs you about various things.');

% all done, display GUI
   set(handles.figure,'Visible','on');

%% GUI callback functions
% These handle the GUI's functionality. Whenever something is clicked, one
% of these callbacks is executed.

   function quitgui(~,~) % Quit program
      if busy
         cancel = true; % this will make the main functions abort asap
         % In case something happens and the GUI freezes, abort after
         % 2 seconds regardless
         start(timer('StartDelay',2,'TimerFcn',@abort));
      else
         delete(handles.figure); % this is used as CloseRequestFcn!
      end
   end

   function abort(~,~) % Not really a callback; called if canceling
      if ~busy % abort has been called already
         return; % this catches cases where cancel is clicked, ...
      end % but the GUI is not frozen and can abort itself
      busy = false; % reset the flag so subsequent calls to abort return
      if cancel % the call results from the quit timer
         delete(handles.figure);
      else % abort is called via button
         cancel = true; % only set cancel flag
         set(handles.runalg,'Str','Run algorithm','Callback',@runalg);
      end
   end

   function showai(hObject,~) % Toggles showing the ai playing
      showaiplaying = get(hObject,'Value');
   end
```

```
function createls(hObject,~) % start creating a learning set
   set(handles.createplay,'String','Save and finish',...
      'Callback',@savels);
   set(handles.createrand,'String','Cancel','Callback',@clearls);
   learning = true; % now building a learning set
   showaiplaying = true; % update field after move execution
   current = blocks(randi(length(blocks))); % random current block
   block = makeblock(current); % load data for current block
   next = blocks(randi(length(blocks))); % random next block
   if hObject == handles.createplay
      field = zeros(20,10); % reset field
      randomls = false; % only update field after each move
      inform('learnplay');
   else
      randomrows = randi(16); % randomly decide how many rows to fill
      field = [zeros(20-randomrows,10);randi([0 1],randomrows,10)];
      randomls = true; % make a new random state after each move
      inform('learnrand');
   end
   makemoves(); % create possible final moves from field and current
   updatedisplay(true); % display field and selected final state
   requesting = true; % now requesting input from the user
   uicontrol(handles.scorelbl); % take focus from button so keys work
end % function createls

function savels(~,~) % save the currently creating learnset to file
   uiresume(handles.figure); % for game over handling
   [filename,filepath,fileformat] = ...
      uiputfile({'*.mat','Learning set (*.mat)'},...
      'Please choose a filename and saving location.',...
      ['learnset_' datestr(now,'yyyymmddhhMMss') '.mat']);
   if fileformat == 0 % user clicked the cancel button
      return;
   end
   save(fullfile(filepath,filename),'learnset');
   set(handles.loadlbl,'ForegroundColor',[0 .8 0],...
      'String',['Learning set "' filename '" loaded.']);
   set(handles.createplay,'String','by playthrough',...
      'Callback',@createls);
   set(handles.createrand,'String','from random states',...
      'Callback',@createls);
   learning = false;
end % function savels

function clearls(~,~) % Clear/delete the currently creating learnset
   uiresume(handles.figure); % unlock
   learnset = struct([]); % reset learnset
   set(handles.loadlbl,'ForegroundColor',[.8 0 0],...
      'String','No learning set loaded.');
   set(handles.createplay,'String','by playthrough',...
      'Callback',@createls);
      set(handles.createrand,'String','from random states',...
         'Callback',@createls);
      learning = false;
   end

   function cont(~,~) % continue making a learning set
      set(handles.createplay,'String','Save and finish',...
         'Callback',@savels);
      set(handles.createrand,'String','Cancel','Callback',@clearls);
      if randomls
         randomrows = randi(16); % randomly decide how many rows to fill
         field = [zeros(20-randomrows,10);randi([0 1],randomrows,10)];
         inform('learnrand');
      else
         field = zeros(20,10); % reset field
         inform('learnplay');
      end
      uiresume(handles.figure); % continue with makemoves
   end

   function loaddat(hObject,~) % load either a learnset or an algorithm
      if hObject == handles.loadset
         if strcmp(get(handles.createrand,'String'),'Cancel')
            response = questdlg(['You are creating a learning set,',...
               ' but did not save it yet. Do you want to save it ',...
               'before loading a new learning set?'],'TetrisFrame',...
               'Save','Discard','Cancel','Cancel'); % save question
            switch response
               case 'Save'
                  savels();
               case 'Cancel'
                  return;
               case '' % user closed the dialog -> assume Cancel
                  return;
            end
            clearls(); % case Discard
         end
         [filename,filepath,fileformat] = ...
            uigetfile({'*.mat','Learning set (*.mat)'},...
            'Please select a previously saved learning set file.');
         if fileformat == 0 % user clicked the cancel button
            return;
         end
         tmp = load(fullfile(filepath,filename));
         learnset = tmp.learnset;
         set(handles.loadlbl,'ForegroundColor',[0 .8 0],...
            'String',['Learning set "' filename '" loaded.']);
      else % handles.loadalg
         [filename,filepath,fileformat] = ...
            uigetfile({'*.m','Algorithm script file (*.m)'},...
            'Please select a script file containing the algorithm.');
      if fileformat == 0 % user clicked the cancel button
         return;
      end
      addpath(filepath); % ensure the algorithm file is on the path
      algorithm = filename(1:end-2); % cut off .m extension
      set(handles.alglbl,'ForegroundColor',[0 .8 0],...
         'String',['Algorithm "' filename(1:end-2) '" loaded.']);
      set(handles.runalg,'Enable','on');
   end
end % function loaddat

function runalg(~,~) % run the selected algorithm with loaded learnset
   set(handles.runalg,'String','Cancel','Callback',@abort);
   busy = true; % flag the algorithm as running
   scores = []; % reset stored scores
   alghandle = str2func(algorithm); % convert the loaded algorithm
   try % failsafe since the algorithm is potentially external
      % extract (partially initial) function handles
      [updaterewardfunc,rewardfunc,deciderequest] = alghandle();
      if ~isempty(learnset) % learning data exists
         rewardfunc = updaterewardfunc(learnset,@makemoves);
      end
      while ~cancel % cancel is used to kill the loop
         rewards = zeros(size(moves)); % initialise rewards as 0
         for posr=1:size(moves,1) % loop through positions
            for rotr=1:size(moves,2) % and rotations
               if isempty(moves{posr,rotr}) % empty move ->
                  rewards(posr,rotr) = -1; % impossible reward
               else
                  [newfield,scoregain] = ... % result of move
                     clearlines(field+moves{posr,rotr});
                  rewards(posr,rotr) = ... % is evaluated
                     rewardfunc(newfield,scoregain,next);
               end
            end
         end % rewards now has the reward for each move
         bestmoves = find( rewards == max(rewards(:)) ); % highest
         if deciderequest(rewards) % query for more examples?
            request(); % wait for user to show best move
            rewardfunc = updaterewardfunc(); % update reward
         else
            thebest = bestmoves(randi(length(bestmoves))); % random
            [pos,rot] = ind2sub(size(moves),thebest);
         end
         execute(); % apply the selected move
      end % repeat until cancelled
   catch ex % something went wrong during algorithm execution
      inform(['The loaded algorithm "',algorithm,'" is erroneous',...
         ' and threw the following error: ',ex.message]);
      set(handles.runalg,'Str','Run algorithm','Callback',@runalg);
   end
   busy = false; % done, clear the flag
end % function runalg

function key(~,event) % handle keyboard usage
   if ~requesting
      return; % ignore if no input request is running
   end
   usedkey = event.Key; % store the key
   if strcmp(usedkey,'return') % outsource return handling
      learnset = [learnset ...
         struct('field',field,'current',current,'next',next,...
         'pos',pos,'rot',rot)];
      if learning
         if randomls
            current = blocks(randi(length(blocks))); % random
            block = makeblock(current); % load data for current
            next = blocks(randi(length(blocks))); % random next
            randomrows = randi(16); % random num of rows to fill
            field = [zeros(20-randomrows,10);
               randi([0 1],randomrows,10)];
            makemoves(); % create possible final moves
            updatedisplay(true); % update what the user sees
         else
            execute(pos,rot); % apply move, continue
         end
         % continue requesting in the new state
      else
         requesting = false; % request done
         uiresume(handles.figure); % continue execution
      end
      return; % that's it for return
   end
   % now for the arrows
   switch usedkey
      case 'uparrow'
         rotup();
      case 'downarrow'
         rotdown();
      case 'leftarrow'
         posdown();
      case 'rightarrow'
         posup();
   end
   originalpos = pos; % stored to catch loops
   while isempty(moves{pos,rot}) % there's no new move here
      if strcmp(usedkey,'rightarrow')
         posup(); % go more right if going right
      else
         posdown(); % in all other cases, go left
      end
      if pos == originalpos % back where we started
```

```
            rotup(); % rotate some more to see if that works
         end
      end
      updatedisplay(true); % new move selected -> update display
   end % function key

%% API
% The API (application programming interface) contains the functions to be
% used by the algorithm to communicate with the framework and the user. The
% functions here are the only ones that should be used by the algorithm,
% all other functions in other sections of this file are considered private
% and are not intended to be called except by one another.

   function request() % requests user input
      updatedisplay(true); % show the user what’s currently going on
      requesting = true; % set the flag so user input is processed
      uiwait(handles.figure); % wait for user input
   end

   function updatedisplay(includingfield) % updates the GUI’s Tetris board
      set(handles.score,’String’,num2str(score));
      if score > 100000000
         set(handles.score,’String’,’Over 1e8’);
      end
      if includingfield % update field display
         delete(get(handles.ax,’Children’)); % clear main axes
         nextblock = makeblock(next); % load data for next block
         [y,x] = find(nextblock); % extract indices
         plot(handles.preview,x,y); % display next block
         [y,x] = find(field); % extract indices
         plot(handles.ax,x,y); % update field display
         rows = find(sum(field+moves{pos,rot},2) == 10); % find fullrows
         if ~isempty(rows) % if there are any full rows, plot them
            plot(handles.ax,1:10,repmat(rows,1,10),...
               ’MarkerFaceColor’,’y’);
         end
         [y,x] = find(moves{pos,rot}); % extract indices
         plot(handles.ax,x,y,’MarkerFaceColor’,’m’); % current
      end
      drawnow; % actually update graphics
   end % function updatedisplay

   function execute(expos,exrot) % executes selected move
      [field,scoregain] = clearlines(field+moves{expos,exrot});
      score = score + scoregain; % update score
      current = next; % update current block identifier
      block = makeblock(current); % create new block data
      next = blocks(randi(length(blocks))); % select new block at random
      makemoves(); % update possible moves
      updatedisplay(showaiplaying); % update score and possibly field
   end
```

```
%% Helpers and Wrappers
% These are small functions that make some lines more concise and easy to
% understand. Most of them wrap a conversion or overflow.

   function result = anyone(input) % wrapper to apply any() to any matrix
      result = any(input(:));
   end

   function posup
      pos = pos+1;
      if pos > size(moves,1)
         pos = 1;
      end
   end

   function posdown
      pos = pos-1;
      if pos < 1
         pos = size(moves,1);
      end
   end

   function rotup
      rot = rot+1;
      if rot > size(moves,2)
         rot = 1;
      end
   end

   function rotdown
      rot = rot-1;
      if rot < 1
         rot = size(moves,2);
      end
   end

   function inform(message) % writes text into the Information box
      switch message % handle default messages
         case 'learnplay'
            message = ['You started creating a learning set by pla',...
               'ying. Use the arrow keys to select a final positi',...
               'on for the current Tetrimino. Press Enter/Return ',...
               'to lock in your choice and continue with the next',...
               ' Tetrimino. When you''re done, click the "Save an',...
               'd Finish" button. Pressing the "Cancel" button cl',...
               'ears the learnset you created. When you lose, you',...
               ' are offered the choice to save the learning set.',...
               ' If you cancel that dialog, you can still save or',...
               ' clear the learning set later.'];
         case 'learnrand'
```

```
            message = ['You started creating a learning set by sol',...
               'ving random states. Use the arrow keys to select ',...
               'a final position for the current Tetrimino. Press',...
               ' Enter/Return to lock in your choice and continue',...
               ' with the next random state. When you''re done, c',...
               'lick the "Save and Finish" button. Pressing the "',...
               'Cancel" button clears the learnset you created. Y',...
               'ou cannot lose in this mode, clicking one of thes',...
               'e buttons is the only way to stop creating a lear',...
               'ning set.'];
         case 'GOlearn'
            message = ['Game over! Click "Continue" to reset the f',...
               'ield and start a new game, adding to the current ',...
               'learning set. Click "Finish" to save the learning',...
               ' set. If you want to discard the learning set, cl',...
               'ick "Finish", then cancel the saving dialog.'];
         case 'GOrun'
            message = sprintf(['The current game is over. New game',...
               ' started. The last game yielded a score of %d. Th',...
               'e algorithm''s average score over the last %d gam',...
               'es is %d.'],score,numel(scores),mean(scores));
            score = 0; % reset score for new game
         case 'clear'
            set(handles.infotxt,'String',''); % clear message
            return; % don't flash red
      end
      set(handles.infotxt,'String',message); % update message
      set(handles.infopnl,'HighlightColor',[1 0 0]); % flash red
      pause(0.05);
      set(handles.infopnl,'HighlightColor',[.6 .6 .6]); % revert
      drawnow;
   end

end % function tetrisframe
```

B Algorithm template

```
function [updaterewardfunc,rewardfunc,deciderequest] = algorithmtemplate()

   % parameters to be trained by the update and used be the reward fcns
   fieldweights = zeros(20,10,7); % weighting each space in the field
   scoreweight = 1; % weighting the score gain

   % all the algorithm function really does is return handles to these
   updaterewardfunc = @updaterewardfcn;
   rewardfunc = @rewardfcn;
   deciderequest = @decide;

   % and of course define them

   % The update function takes as inputs the (changed) learnset and a
   % handle to the function to calculate all final positions so it can
   % determine which moves were passed up in favor of the best move in
   % each learning example. It return the handle to the reward function
   % now using the updated parameters.
   function hNewRewardFcn = updaterewardfcn(learnset,hMakeMoves)
      % train parameters that affect rewardfcn
      % this template does nothing, add your actual learning
      % algorithm here (mostly, depends on your algorithm)
      hNewRewardFcn = @rewardfcn; % using the new parameters
   end

   % The reward function determines a numerical reward for a target field
   % with a specific score gain to reach, taking into account the next
   % piece to spawn. It takes as inputs the target field, score gain and
   % next piece identifier and uses the parameters trained by the
   % updaterewardfcn. It returns the reward as a positive integer.
   function reward = rewardfcn(targetfield,scoregain,next)
      % depending on your approach, possibly adapt this as well
      % this trivial example applies the weights corresponding to the
      % next piece to the field and neutral weigth to the score
      fieldreward = fieldweights(:,:,next) .* targetfield;
      scorereward = scoreweight * scoregain;
      reward = fieldreward+scorereward;
   end

   % The decide function decides whether to take a decision with the
   % rewards given as input or request a new sample from the user. It
   % returns true if a new sample is to be requested.
   function doRequest = decide(rewards)
      best = find( rewards == max(rewards(:)) ); % best moves
      % trivial example: request more only if no single best move exists
      if numel(best) == 1
         doRequest = false;
      else
         doRequest = true;
      end
   end
end
```

Bibliography

[Ald08] H. Aldahiyat. Tetris for dummies, 2008. URL http://de.mathworks.com/matlabcentral/fileexchange/21246-tetris-for-dummies. [Online; accessed 2-March-2015]. (Cited on page 9)

[BKM05] N. Böhm, G. Kókai, S. Mandl. An evolutionary approach to tetris. In *The Sixth Metaheuristics International Conference (MIC2005)*. 2005. (Cited on page 5)

[CMDS10] R. Cohn, M. Maxim, E. Durfee, S. Singh. Selecting operator queries using expected myopic gain. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 2, pp. 40–47. IEEE, 2010. (Cited on page 8)

[Fah12] C. Fahey. A lot of information about Tetris, 2012. URL http://colinfahey.com/tetris/tetris.html. [Online; accessed 2-March-2015]. (Cited on pages 5 and 9)

[Fig12] M. Fig. Matlabtetris, 2012. URL http://de.mathworks.com/matlabcentral/fileexchange/34513-matlabtetris. [Online; accessed 2-March-2015]. (Cited on page 9)

[Gie11] J. de Gier. Tetris (vs. AI), 2011. URL http://de.mathworks.com/matlabcentral/fileexchange/33701-tetris--vs-ai. [Online; accessed 2-March-2015]. (Cited on pages 5 and 9)

[HYL08] K.-Z. Huang, H. Yang, M. R. Lyu. *Machine learning: modeling data locally and globally*. Springer Science & Business Media, 2008. (Cited on page 7)

[LMM09] M. Lopes, F. Melo, L. Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, pp. 31–46. Springer, 2009. (Cited on page 8)

[ML10] F. S. Melo, M. Lopes. Learning from demonstration using mdp induced metrics. In *Machine Learning and Knowledge Discovery in Databases*, pp. 385–401. Springer, 2010. (Cited on page 7)

[ML11] M. Mason, M. Lopes. Robot self-initiative and personalization by learning through repeated interactions. In *Human-Robot Interaction (HRI), 2011 6th ACM/IEEE International Conference on*, pp. 433–440. IEEE, 2011. (Cited on page 7)

[NR$^+$00] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pp. 663–670. 2000. (Cited on page 17)

[RGB10] S. Ross, G. J. Gordon, J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *arXiv preprint arXiv:1011.0686*, 2010. (Cited on pages 5 and 8)

[SE12] Z. Shao, M. J. Er. A survey of inverse reinforcement learning techniques. 2012. (Cited on pages 7 and 8)

[SL06] I. Szita, A. Lörincz. Learning Tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941, 2006. (Cited on page 5)

[Var15] Various. Tetris Wiki community project, 2015. URL `http://tetris.wikia.com/wiki/Tetris_Wiki`. [Online; accessed 2-March-2015]. (Cited on page 9)

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 03.03.2015

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature:

Stuttgart, 03.03.2015