Bachelor thesis
Computing Science

Radboud University

Human-in-the-loop Strategy Synthesis: PAC-MAN verified

Author:
Manuela Bergau
s4543645

First supervisor/assessor:
Dr. Nils Jansen
n.jansen@science.ru.nl

Second assessor:
Prof. Dr. Mariëlle Stoelinga
m.stoelinga@cs.ru.nl

July 8, 2019

Abstract

Machine learning algorithms lack human intuition when exploring an environment. The aim of these algorithms is, that an autonomously acting agent learns desired behavior. During the initial exploration phase the agent can show useless or even harmful behavior.
In this thesis we show a method to compute a strategy by observing humans playing the arcade game PAC-MAN. A strategy is a collection of states together with actions that were chosen by the human player. We focus on the reduction of possible states because the available game data is limited.
Furthermore we present various agents that handle the previously computed strategies differently. In a last step, we compare the computed strategies to a randomly choosing strategy to rate the strategies and agents. Thereby we are able to observe a significant difference in terms of won games and length of the games between random choice and using a strategy derived from recorded human behavior.

Contents

1 Introduction **3**
- 1.1 Contribution . . . 4
- 1.2 Structure of the thesis . . . 5

2 Preliminaries **6**
- 2.1 Environment . . . 6
- 2.2 State . . . 7
- 2.3 Agent . . . 7
- 2.4 Reinforcement Learning . . . 8
- 2.5 Important Definitions . . . 8

3 Strategy **9**
- 3.1 Terminology . . . 9
- 3.2 Test Environment . . . 9
- 3.3 Strategy Optimization . . . 10
  - 3.3.1 Smart Data Saving . . . 10
  - 3.3.2 During The Game . . . 13
  - 3.3.3 Postprocessing Of The Data . . . 13

4 Agents **15**
- 4.1 Simple Agents . . . 15
  - 4.1.1 Greedy Agent . . . 15
  - 4.1.2 Ghost Agent . . . 16
  - 4.1.3 Food Agent . . . 16
- 4.2 Combined Agents . . . 16
  - 4.2.1 Mixed Agent . . . 16
  - 4.2.2 Safe Agent . . . 16

5 Verification **18**
- 5.1 Experimental Setup . . . 19
- 5.2 Experimental Results . . . 19
  - 5.2.1 Test Runs . . . 20
  - 5.2.2 Environments . . . 23

6 Related Work **29**

7 Conclusions And Future Work **30**

A Test Environments **33**

B Test Data **35**

Chapter 1

Introduction

"Computers are programmed, so are the humans, but the computers can't act outside their programming, whereas the humans can."[1] In this thesis we use this statement as an inspiration to implement a new method for computers to learn from human behavior.
Commonly, machine learning algorithms are used to let a computer learn a given task. Depending on the complexity of the problem, it might be difficult to establish an optimal learning algorithm. Take for instance the well-known machine learning technique reinforcement learning. RL obtains an optimal strategy in an agent-based setting via episodic exploration of an environment [10]. However, depending on the complexity [9] of the task, the exploration may require multiple (or even infinitely many) attempts. Due to initial absence of information about the environment (*exploration phase*), the agent may display – in the beginning – useless or even harmful behavior. An agent determines its next move (*action*) either randomly or using a learned policy. Most of the time the agent makes choices in a random way during the exploration phase.

Due to human's ability to intuitively identify actions as unreasonable, some of these problems can be mitigated by learning from human behavior. There are many possible ways to integrate human skills into the learning cycle, e.g., using human feedback to guide the agents behavior during learning. An example is *modular inverse reinforcement leaning* [1]. It is a method to use observed human visuomotor behavior for a navigation task in a reinforcement leaning setting [1]. [11] provides an algorithm learning simple tasks by imitation.
To integrate human feedback in our setting we make use of strategies. A so-called *strategy* determines all possible action choices an agent can make. Specifically, a strategy in our setting provides probability distributions over such actions, i.e., how likely the agent chooses which action. A human strat-

[1] Abhijit Naskar, The Constitution of The United Peoples of Earth

egy is then a strategy imitating typical choices a human would make for the agent.

The central research questions in this paper are:

- fully determine such a human strategy from a (potentially low) number of observations of human behavior.
- verify the quality of these strategies.

More specifically, in order to construct a full human strategy, we need to observe human input for all potential states of certain scenario. A state here represents a specific configuration of the environment at hand. As such a number of observations is infeasible, *the goal is to build good strategies using as little human input data as possible.* We want to asses if the imitation of human behavior will improve the purely random exploration of the environment. Verification techniques, implemented in the model checker storm [5], provide methods to rigorously assess the quality of a strategy and a model. We use such a technique to compare the strategies against random choosing. We focus on the famous arcade game PAC-MAN [3] to demonstrate our approach.
In order to solve the task at hand the human must be able to understand the environment. To achieve this aim, the concept of gamification can be helpful, which uses elements of game design to achieve desired behavior [6]. The principle of gamification supports the human in understanding the task intuitively and in playfully finding a solution. Moreover, it keeps the test persons motivated [6].
PAC-MAN is a complex task for an agent to solve as there are multiple tasks (avoid ghosts, collect food). We have the capability to observe humans solving that (or sufficiently similar) tasks (multiple times). With the recorded behavioral data we can build a *strategy*. We let different test persons solve our problem (playing the game) using different environments. For each test person we build an individual strategy.
After the computation of a strategy we augment the data to achieve better results, i.e., we aggregate action data of similar states.
Using model checking code from [7] we verify the performance of our strategies.

1.1 Contribution

The thesis provides the following main contributions:

- a version of the PAC-MAN game environment [3] for data collection
- a procedure for state data augmentation to compute a strategy

- a definition of different agents that use the computed strategies
- a verification framework for the strategies

1.2 Structure of the thesis

Our research is divided into three phases: first obtaining the strategy data, second optimizing the strategy, and finally verifying the obtained strategy. We programmed a number of agents to test our strategy visually as well.
We will provide the necessary background in chapter 2. After that we will describe all steps we undertook to obtain and optimize the strategy in chapter 3. In chapter 4 we will give information on the different agents we programmed to use our strategies. Finally, we will explain how we verified the strategies and show the results in chapter 5.

Chapter 2

Preliminaries

In the following chapter we will explain all relevant background material and give all definitions that are necessary to understand the research.

2.1 Environment

The environment we are using is a grid of size $x \times y \in \mathbb{N} \times \mathbb{N}$. n agents are operating in this environment. The position of each agent is given as a tuple (x_n, y_n) where n is the agent's number. One agent is pacman which can be operated by a human player. $n - 1$ agents are ghosts. Pacman must avoid the ghosts in order to win the game.
Every position of the field is either a wall or a passage. Passages can contain food. Pacman must collect all the food pieces to end the game successfully.

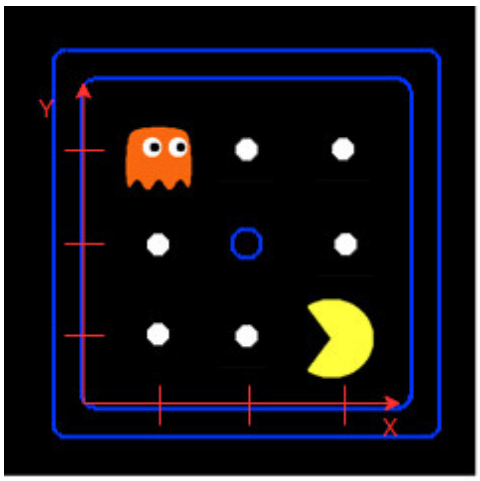


Figure 2.1: An example environment

2.2 State

A state is a configuration of the given environment at a given time. It contains all information about the changing elements of the game which are all position tuples of the n agents as well as which passage contains food. For example the state of picture 2.1 is:

- pacman position: (2,0)
- pacman direction: West
- ghost position: (0,2)
- ghost direction: East
- walls: (1,1)
- food: (0,0), (1,0), (0,1), (1,2), (2,2), (2,1)

The number of possible states (excluding losing states) can be determined by:

$$(\prod_{i=0}^{n-1}(x \cdot y) - |walls| - i) \cdot n \cdot directions \cdot |food| \cdot 2$$

If we store the state information using this style there are 5376 states possible for the environment from picture 2.1.
We use a different format to store the state information which is described in section 3.3.1.

2.3 Agent

As Michael Wooldridge states in [12], there are multiple definitions of the term agent. In this paper we will make use of his definition that an agent is a computer program that is situated in an environment and is capable of autonomous actions.

Figure 2.2: different agents of PAC-MAN

2.4 Reinforcement Learning

Reinforcement learning is a method to let an agent learn desired behavior. The agent does not know the goal but learns through rewards and punishment (negative rewards). Its goal is to maximize the reward. The reward function $R : S \to \mathbb{R}$, where S is the set of states, with $R(s) = r$ is not known to the agent in many cases. With each action taken the agent receives immediate (negative) reward as a grading for his action. Examples of different algorithms can be found in [8]. Mostly there is an *explore* and an *exploit* part of the algorithm. Exploit is using the currently best action according to the strategy learned so far. Exploration is choosing a new action to see if it is better.

2.5 Important Definitions

Definition 1 (Markov Decision Process)**.** *An MDP M is defined as $M = (S, A, P, R)$, where S is a finite set of all possible states, A is a finite set of all possible actions, $P : S \times A \to Distr(S)$ is a probability function that determines a state and an action to a probability distribution over the next state. R is a reward function $R : s \to r \in \mathbb{R}$ with $R(s) = r$. $A(s) = \{\alpha \in A | P(s, \alpha) \neq \emptyset\}$*

Definition 2 (Markov Chain)**.** *A discrete-time Markov chain is an MDP if $\forall s \in S : |A(s)| = 1$*

A Markov chain describes that the probability of a event happening in the next state only depends on the current state and not on the path to the current state. That implies that if we know the current state we have enough information to make estimates about the future.

Definition 3 (Strategy)**.** *A strategy σ for an MDP M is a function $\sigma : S \to Dist(A)$.*

We use strategies that are memoryless and randomized. A given strategy σ for a MDP will remove the non-determinism resulting in an induced Markov chain.

Definition 4 (Induced Markov Chain)**.** *Given an MDP M with initial state $s_I \in S$ and a strategy $\sigma \in \sum^M$, the induced Markov chain is given by $M^\sigma = (S, A, P^\sigma, R)$ where $P^\sigma(s, s') = \sum \sigma(s)(a) \cdot P(s, a)(s')$*

Chapter 3

Strategy

In this chapter we will give all the technical details of the test setup and how the data is processed.

3.1 Terminology

State: We differentiate between state (as explained in chapter 2) and *relative state*. A relative state contains information about the distance between pacman and other elements in the environment, such as ghosts and food. More information can be found in section 3.3.1. If not explicitly stated, in the following sections, "state" refers to *relative state*. We call the full game configutarion *game state*.

Strategy: A strategy is a collection of relative states together with actions and their probability weight. A strategy file is a strategy of one test person. It is computed by letting the test person play multiple games and collect the data in the strategy file.

Agent: Agents, in our setting, are computer programs that are able to use a given strategy to determine its next actions for pacman. Therefore an agent determines its state and utilizes the probability weights in the strategy to find a fitting action.

3.2 Test Environment

To find good strategies we experimented with different designs. We want to ensure the occurrence of common situations. Intuitively, in a small environment, dangerous situations with multiple ghosts coming close will occur more often than on a large field. Data of those dangerous situations are the most valuable ones as these are the ones we want to prevent with our strategies.

The number of ghost agents in the environments varies. The different number is necessary due to the different size of the game fields to keep the game difficult enough for the test person to remain interesting but not too difficult to get too frustrating. For example a small field may already be difficult enough with one ghost due to the limited movement space.
Apart from the size, most of the standard pacman fields do not contain every possible field piece (see picture 3.1) equally often. Instead of using positions we categorize each position in one of the "field pieces" that are shown in picture 3.1. For a good strategy it is important to have a lot of information about these field pieces. Consider for example crossings, here the agent has all four possible actions at its disposal. Consequently, more data is needed to cover all possibilities. Our solution to this problem was to build an environment with many crossings. We do the same for t-crossings, corners and long straight corridors as well.
Each environment will be played by the test player multiple times in random order. The variation ensures that our test person will stay focused on the game without getting bored.
We have 7 different layouts for our tests. All layouts can be found in the appendix.

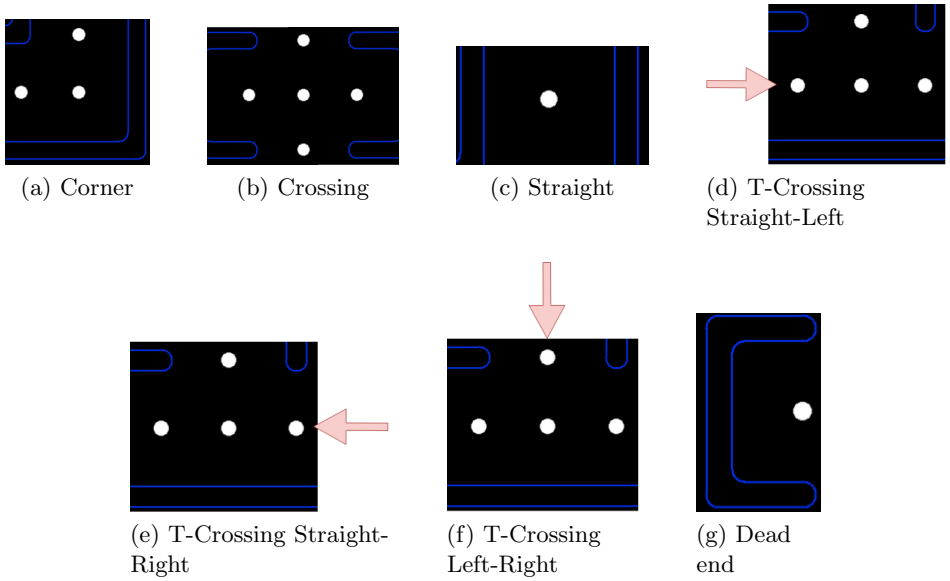
3.3 Strategy Optimization

As we discussed previously, we do not have a lot of data as the information collected for each game is limited. This leads us to a problem: in order to obtain a good strategy we either need information about all possible states or we need to save the available data in an efficient way and make use of (post)processing. In order to gain the information we need, we use various optimizations and tricks turning our initial data into good strategies. We do not change the strategy itself by removing bad decisions as we want to faithfully imitate the behavior of the human. By optimizing we refer to smart data storage or to the merging of similar states. "Merging" refers to adding up action counts of two or more states. A more detailed explanation can be found in section 4.1.2.
The following subsections describe the steps we used to get meaningful data to be handed to the different agents that we describe in chapter 4. First we need to save the game data in a convenient way then we prepare the data to be given to an agent.

3.3.1 Smart Data Saving

We store the states in a way so that we do not need to save positions on the field. Instead we save distance measures between the changing elements on the field (ghosts, pacman, food). This makes the strategy independent from the field we are playing on, thus the same strategy may be used on a

field that was not used for the strategy computation. We call these states *relative states*.
A relative state looks like this: Way: Corner Food: [('Go', 7), ('Reverse', 20)] Ghosts: [('Reverse', 5), ('Go', 6)]. Where "Way" contains the information about the kind of "field piece" pacman is located. This information is used to determine the possible actions. All possible kinds of "field pieces" are shown in picture 3.1. The types are "Corner" (a), "Crossing" (b), "Straight" (c), "T-Crossing-Straight-Left" (d), "T-Crossing-Straight-Right" (e), "T-Crossing-Left-Right" (f), "Dead end" (g).



(a) Corner (b) Crossing (c) Straight (d) T-Crossing Straight-Left

(e) T-Crossing Straight-Right (f) T-Crossing Left-Right (g) Dead end

Figure 3.1: Different field pieces of a PAC-MAN environment

"Food" and "Ghosts" give information how far away the closest food piece or ghost are in each possible direction. The relative directions are "Go", "Reverse", "Left" and "Right". The directions are determined from the point of view of pacman and shown in picture 3.2. Using these directions we achieve independence of the orientation of the environment. "Go" is the direction in which pacman is traveling at the moment, Reverse is the opposite direction. The numbers next to each direction refer to how many steps are between pacman and the ghosts or food pieces. As distance measure we take the number of steps pacman needs to travel to reach the position.
The agent uses the "field piece" information to speed up the search for a fitting state in the strategy and to transfer the relative directions to actual directions. Considering corners, the label information gets necessary as we want to store information independent from the orientation of the environment. If we do that using the relative directions from 3.2 we have two kinds of corners (going Left or going Right). We decided to use the label "Go"

instead as there are only two choices of actions. However, when the agent now wants to translate the relative direction it needs to know if it is a corner or a straight passage.
For a similar reason, we differentiate between three kinds of t-crossings. Depending on the entrance point of pacman the possible actions are different. Using the passage labels supports easy lookup of the strategy entries as well. Looking at our example from chapter 2 picture 2.1 the relative state looks like this:

- Way: Corner
- Food: (Go, 1), (Reverse, 1)
- Ghost (Go, 4), (Reverse, 4)

The number of possible relative states can be roughly calculated using

$$\prod_{s \in fieldpieces} (max\, distance)^{\#possible\, actions} \cdot 2$$

Thus, we have a maximum of 7070 states but not all distance combinations are possible. For example in field 2.1 the sum of the ghost distances must be equal to 8. So, for our example environment 2.1 the number of states is reduced from 5376 (from 2) to 224 possible states.
During the game we collect a statistic of relative states together with how often the actions are taken (score) in this relative state. Later we use this statistic to compute our strategy, where these scores represent the probability weights we give to the actions.
Attached to each relative state there is a counter counting how often each possible action was taken. Here we use the same relative directions "Go, Reverse, Left, Right".

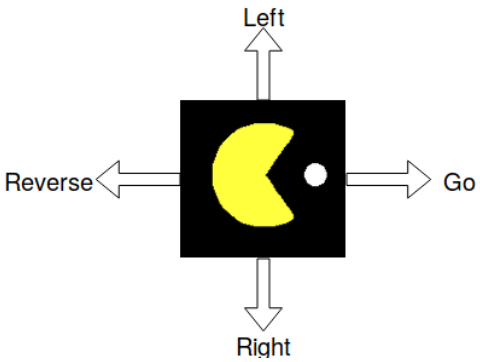


Figure 3.2: Directions relative to pacman

3.3.2 During The Game

When a test person plays, each action choice that occurs is saved together with the state in the strategy file. There is one exception: We do not save the action choice if the next state is a failing state. The intuition behind that is that, different to reinforcement learning, we only save the situation and the action, not the outcome of the action. Our goal is to find an efficient strategy, thus saving decisions that clearly lead to a failing state would result in undesired behavior of our agents as they mirror the human behavior. On the other hand, saving exclusively the decisions made in won games would result in more problems gathering sufficient data. Concluding the game with a defeat does not implicate that all decisions made in-game were wrong. Therefore, we decided to exclude only the worst decision, i.e. the one directly before being eaten by a ghost.
As stated earlier our test players play multiple games in different game environments. The data of all games of each person is stored in the test persons personal strategy file.

3.3.3 Postprocessing Of The Data

Before the strategy is handed to the agent it is processed to get better performance results. We want to minimize the number of possible states without loosing necessary information. Due to the test setup it is impossible to get enough data of every possible situation.
In a first processing step we label the distances of the food and ghost with labels for distance ranges (No, Far, Medium Far, Medium, Close, Very Close) instead of numerical distances. Action counts of states with the same labels are summed up. The intuition behind the labeling is that especially when a ghost is far away the exact distance is not important for the decision. While testing our agents we found that the labeling from 3.3 worked the best. This labeling leaves us with 6 possibilities for each direction. The number of possible states can be calculated with

$$\sum_{s \in fieldpieces} (\#Distance\,labels)^{\#possible\,actions} \cdot 2$$

Resulting in a maximum of 2022 possible states, but depending on the environment not all states are possible. Considering our small example 2.1, only three of the distance labels are possible as the environment contains only 2 different field pieces, reducing the number of possible states to 36.

Depending on the agent we are using the data needs to be processed a second time to remove food or ghost distance information. For example, the ghost agent (see section 4.1.2) only uses the "Ghost" information of the state. Again action counts of the new states are summed up if they are equal.

Label	Distance
No	None
Far	> 15
Medium Far	$15 - 9$
Medium	$8 - 4$
Close	$3 - 2$
Very Close	1

Figure 3.3: The labels with the distance ranges

Chapter 4

Agents

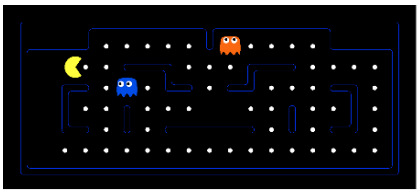
We programmed different agents to test the strategies. The agents use the given information differently. There is no direct interaction between the test persons and the agents. An agent gets a strategy file, that is a previously computed strategy of one test person. In all cases, if no fitting state is found in the strategy file, the action is chosen randomly from all possible actions. As we will see in the test results, the way the strategy data is used will have an impact on the performance of the agents. All agents have in common that they use only one strategy file at a time. The performance difference is based purely in the usage. How they make use of the information to determine their choices is explained in the following sections for each agent.

4.1 Simple Agents

Simple agents make a direct choice based on the strategy they get.

4.1.1 Greedy Agent

The Greedy agent will search for an exact match of its current state in the strategy file. Since there are 2022 possible states (see section 3.3.3), this agent will require larger amounts of information than other agents and it will display the worst overall performance. The lack of data will lead to randomly choosing an action in many situations. Due to the test setup it is hard to get enough information about every possible state.



State	Action
Straight Food: {Go: 5, Reverse: 6} Ghost{Go: 1, Reverse: None}	Go: 1, Reverse: 5
Straight Food: {Go: 1, Reverse: 6} Ghost{Go: 4, Reverse: 12}	Go: 4, Reverse: 6
Straight Food: {Go: 1, Reverse: 10} Ghost{Go: 9, Reverse: 4}	Go: 5, Reverse: 1

Figure 4.1: An example how the Greedy agent chooses a strategy entry

4.1.2 Ghost Agent

The Ghost agent reduces its state to the kind of field piece (see picture 3.1) and the ghost distances. It then searches for matches with the closest two ghosts in the strategy. As there can be more states in the strategy file that match, all action counts are combined (see Strategy Optimization). Take the following two states as an example:

- Way: Straight Food: [('Go', 7), ('Reverse', 20)] Ghosts: [('Reverse', 5), ('Go', 6)]: {Go: 2, Reverse: 1}
- Way: Straight Food: [('Go', 6), ('Reverse', 19)] Ghosts: [('Reverse', 5), ('Go', 6)]: {Go: 5, Reverse: 2}

They will be combined to:
Way: Straight Ghosts: [('Reverse', 5), ('Go', 6)]: {Go: 7, Reverse: 3}
An advantage is, that the agent will stay the longest in the game but it will not reach a high score as it neglects the food.

4.1.3 Food Agent

The Food agent works the same way as the Ghost agent, but, as the name suggests, searches for food pieces. The obvious disadvantage is that it does not consider ghosts when choosing an action and therefore makes no attempt to avoid them. This can lead to the agent failing early in the game.

4.2 Combined Agents

Combined agents are higher order agents that make use of two simple agents. They chose between the agents to determine the next action.

4.2.1 Mixed Agent

The Mixed agent combines the Food agent and the Ghost agent. Given a probability (e.g 20%, 80%) the agent chooses the action of the Food Agent with a probability of 20% and the action of the Ghost agent with probability 80%. The choosing probability can be changed. A disadvantage is that the agent could fail in critical situations because the action of the Food agent is chosen.

4.2.2 Safe Agent

The safe Agent is also a combination of Food and Ghost agent. In contrast to the Mixed agent, the actions of the Ghost agent is chosen if at least one ghost in the current state has the distance label 'close' or 'very close'. Otherwise the actions of the Food Agent are chosen.

This is probably the most optimal agent as it resolves the disadvantage of the Mixed Agent and still looks for food as often as possible to reach a high score.

Chapter 5

Verification

Verifying the obtained strategies can be done using different methods. Using purely a model checker proves to be hard to achieve. Considering that the state depends on the distance of the agents to each other modeling the state transitions proves to be hard to achieve. Furthermore, there is no goal position that the agent needs to reach, but the goal is defined as reaching all positions containing food at least once.
To circumvent direct modeling we used code from [7]. The code we are using is calculating the risk of loosing given a direction. The model is looking ten steps ahead using the model checker storm.
We decided to calculate for each state and each possible pacman decision one following state. That gives us a tree of various possible game paths. As we have limited calculating capacity we are modeling a maximum of 30 steps. Depending on the strategy, the number of states will grow exponentially with each step (worst case).
We use two measures to verify our strategies: *death probability* and *death ratio* as well as test runs of all agent-strategy combinations. After every step the two measures are calculated. The *death probability* is calculated as follows:

$$\frac{\sum P(State) \cdot \sum\limits_{d \in \ directions} P(death,\ d) \cdot P(d,\ strategy)}{Nr\ of\ States}$$

$P(State)$ is the probability of reaching the state, calculated by multiplying the action probabilities of the ghost agents that are needed to reach this state. $Directions$ is the set of all the possible directions of the given state. $P(death, d)$ is the likelihood, given by the model checker [7], to collide with a ghost when taking the given direction. $P(d, strategy)$ is the probability we get from our strategy to choose a given direction. $Nr\,of\,States$ is the total number of states computed.
As a second measure we count the number of states that are generated and

divide the number of lost end states with the total number of generated states, we call this number the *death ratio*.
Both numbers give us an estimate on how safe our strategy is, where a high *death probability* together with a low *death ratio* can be considered safe. A high death probability occurs when pacman is close to one or more ghost agents. Now, when the *death ratio* stays low we can conclude that the strategy is able to resolve dangerous situations.
To compare our values we calculate a baseline, which is an agent that is randomly choosing its actions.
Our test setup does not give information about the number of random chosen actions as well as the number of steps. Therefore we run 100 games to get a first performance impression.
There is a difference in running test games and running the verification procedure. Take for example the state:
Way: Straight Food: [('Go', 7), ('Reverse', 20)] Ghosts: [('Reverse', 5), ('Go', 6)]
with the strategy entry: {Go: 2, Reverse: 1} we calculate one new state for the action "Go" and one for "Reverse". While in test games pacman chooses one of these actions, resulting in one sequence of states. So if we run 100 test games it can happen that two or more games have the same sequence of states.

5.1 Experimental Setup

Four test persons played multiple games on all environments described in section 3.2. The exact numbers of games can be found in the appendix. The game data of each game is recorded as described in section 3.3.2. The number of observations can be found in table 5.1.
Form all the recorded game data we compute an individual strategy for each test person using the steps described in section 3.3.3.

Person 1	Person 2	Person 3	Person 4
1597	2397	4016	6085

Table 5.1: The number of observations made of each test person

5.2 Experimental Results

In the following subsections we show the results of our verification procedure using different environments. We cannot show all results here. All verification data can be found in the appendix. We used a Lenovo ThinkPad T480s with an Intel i7 and 24 GB RAM.

We are using two environments: a small one with one ghost (picture 5.1) and a larger one with four ghosts (picture 5.2). The results of the test runs can be found in section 5.2.1. Based on this data we chose the safe agent to calculate the *death probability* and *death ratio* for each strategy. The resulting plots for each environment can be found in section 5.2.2.

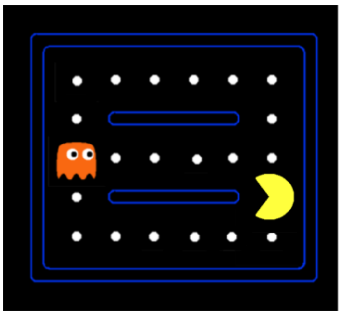


Figure 5.1: the small environment

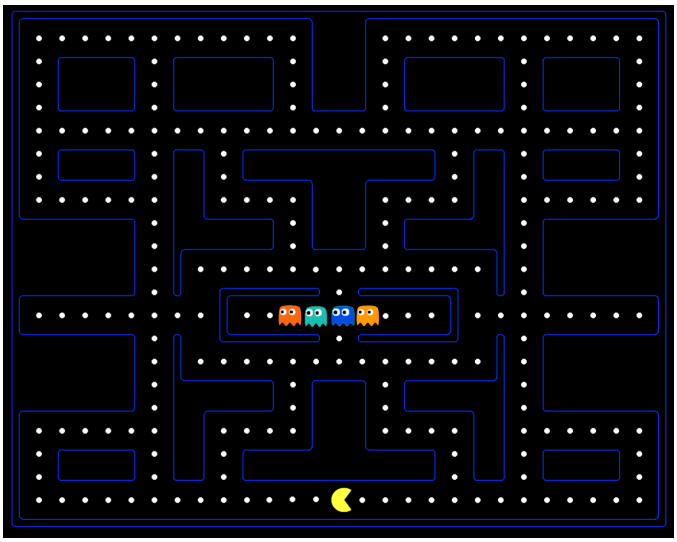


Figure 5.2: Big test field

5.2.1 Test Runs

We ran 100 games for every strategy-agent combination in both test environments to get information about the average score and how many action choices are not based on the strategy (random choosing). Furthermore we computed the average score of the test games.

Strategy	Agent	Win rate	average score	# Steps	# random choices	# random ghost choices
Random	-	0/100	-465.62	1342	1342	-
1	greedy	3/100	-389.28	1638	719	-
2	greedy	3/100	-384.82	1742	510	-
3	greedy	18/100	-227.64	1754	329	-
4	greedy	17/100	-224.3	1970	263	-
1	ghost	13/100	-259.7	2230	89	-
2	ghost	10/100	-303.87	1896	83	-
3	ghost	24/100	-153.51	2331	26	-
4	ghost	34/100	-49.06	2426	0	-
1	food	17/100	-223.45	1785	75	-
2	food	22/100	-178.73	1343	6	-
3	food	25/100	-150.68	1528	0	-
4	food	31/100	-82.05	1455	0	-
1	mixed	9/100	-310.8	1940	93	72
2	mixed	12/100	-289.58	1588	60	60
3	mixed	27/100	-119.84	2314	18	18
4	mixed	37/100	-17.49	2259	0	0
1	safe	10/100	-299.99	1799	72	8
2	safe	27/100	-126.77	1527	15	12
3	safe	31/100	-78.66	1816	3	3
4	safe	37/100	-16.72	1822	0	0

Figure 5.3: Results of test runs using the small environment

In table 5.3 we can see, that most of the time the human strategies have a higher step number compared to random choosing. Comparing the agents we can identify various performance differences. As expected, the greedy agent has a far higher amount of randomly chosen steps compared to the other human strategies. Furthermore the ghost and mixed agents have the highest number of steps followed by the safe agent. The safe agent has also the highest average score and the highest number of games won.

Strategy	Agent	Win rate	average score	# Steps	# random choices	# random ghost choices
Random	-	0/100	-440.03	5903	5903	-
1	greedy	0/100	-196.15	7035	2943	-
2	greedy	0/100	-221.24	6623	1933	-
3	greedy	0/100	-170.01	8241	2082	-
4	greedy	0/100	-100.36	8415	1552	-
1	ghost	0/100	-199.71	6951	616	-
2	ghost	0/100	-119.62	9992	553	-
3	ghost	0/100	-89.02	10442	265	-
4	ghost	0/100	-2.11	13191	254	-
1	food	0/100	-193.4	7130	350	-
2	food	0/100	-105.46	6436	131	-
3	food	0/100	-140.39	5909	105	-
4	food	0/100	-85.41	6431	106	-
1	mixed	0/100	-197.86	6546	538	467
2	mixed	0/100	-67.77	8897	400	363
3	mixed	0/100	-71.77	10047	285	241
4	mixed	0/100	8.0	10620	181	146
1	safe	0/100	-178.87	6597	337	79
2	safe	0/100	-27.32	8592	330	169
3	safe	0/100	-78.31	7851	304	159
4	safe	0/100	60.28	10472	315	130

Figure 5.4: Results of test runs using the large environment

In table 5.4, the largest difference compared to the small environment is that we have no games won in the large environment. Nevertheless we still can observe performance differences. First we see that we have two positive scores. Again we can observe that the greedy agent has a high amount of random chosen steps as well as the worst score. The food agent has the lowest number of random choices. If we compare the two combined agents (mixed and safe agent) we see that the number of random choices made by the ghost agent is smaller using the safe agent. Still, the difference between these two agents is small.
Looking at the number of steps we can note again that the ghost agent is the best one. The worst human score is -221.24 which is far better compared

to -440.03 of random choosing.

Summarizing our observations we can conclude that the agents have performance differences that do not depend on the strategy used. Especially the greedy agent can be identified as not optimal while the safe agent is the best one looking at score and number of won games. When it comes to the number of steps the ghost agent is the clear winner. We consider the safe agent the best in choosing between safeness of the ghost agent and point collection of the food agent.

5.2.2 Environments

In the following two sections we want to compare the different strategies using the safe agent and the two measures *death probability* and *death ratio* as described in section 5.

Small Environment

The environment 5.1 can be solved within the 30 steps that we are simulating with our verification procedure. The minimum number of steps needed to win the game (collect all food without being eaten by a ghost) is 21, by moving in a spiral pattern collecting all food pieces. First we look at the number of winning end states that were computed during the verification (picture 5.5).

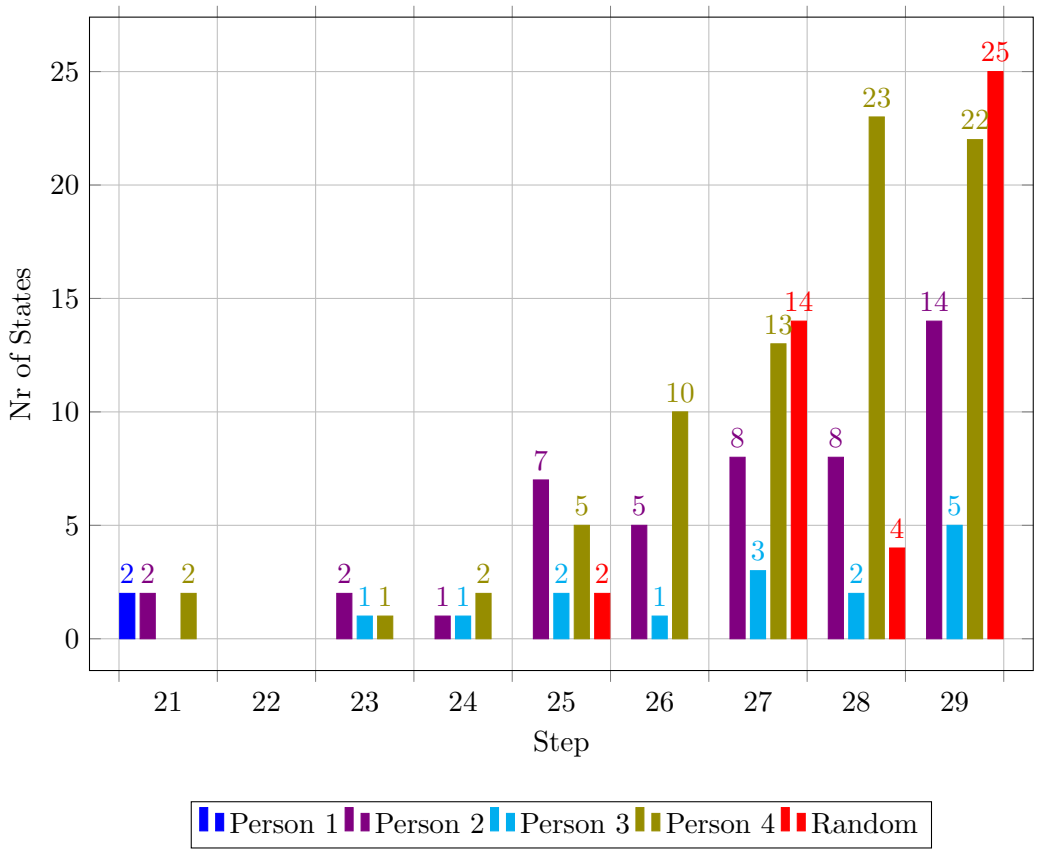


Figure 5.5: Games Won

Person 1	Person 2	Person 3	Person 4	Random
2	47	15	78	45

Figure 5.6: total number of games won

Noticeably, person 1 has only 2 won games in total and person 3 and the random strategy have won no games with maximum score. Person 4 has a higher number of games won late in the game compared to person 2. Person 3 has the lowest number of wins after person 1.

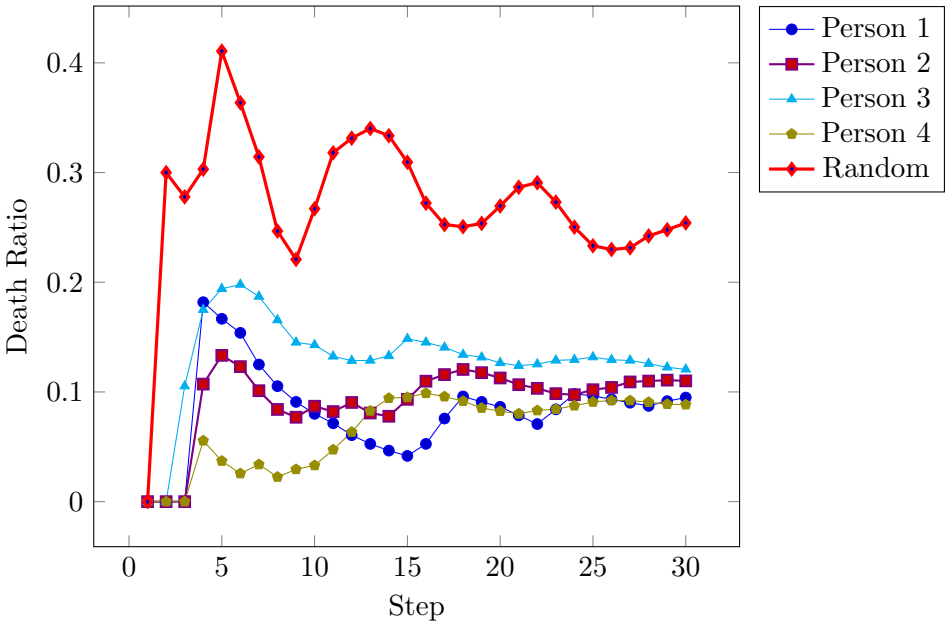


Figure 5.7: death ratio

Now we take a closer look at the “death ratio” (picture 5.7) and the “death probability” (picture 5.8). As stated before, a high death probability together with a low ratio is optimal. All four human strategies have a death ratio below 0.2 while random choosing is always above 0.2. We can observe a peak within the first few steps at all strategies. This is a possible result of the small field. The ghost agent and pacman start close together so the first steps are crucial. Person 1 and person 4 have the highest peak in the first few steps compared to the other human strategies. We can detect a clear difference between person 4 and the other human strategies. It is significantly lower than the others.

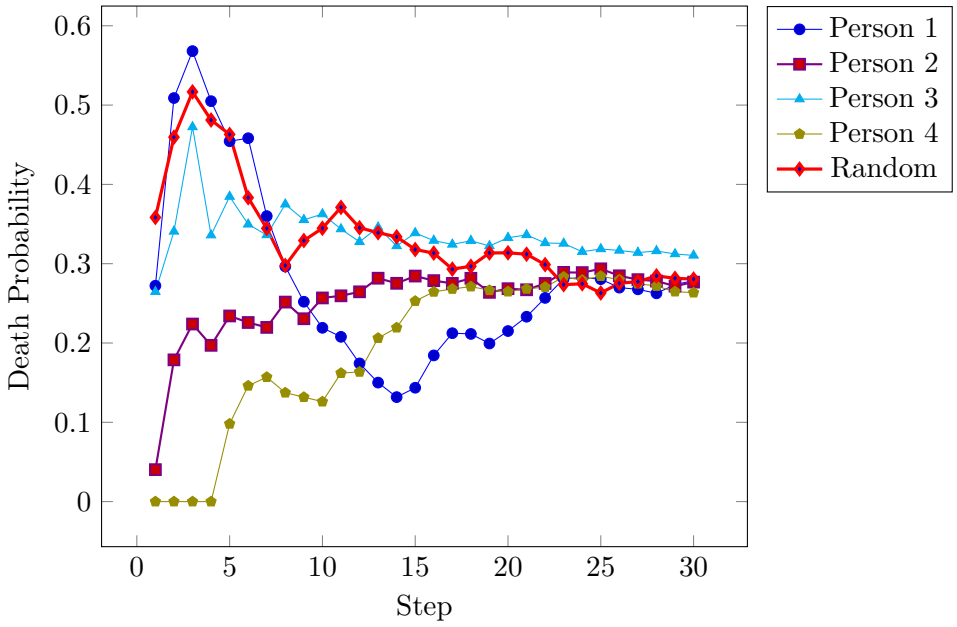


Figure 5.8: average death prop

Considering picture 5.8 we can observe that person 3 has a death probability close to random. However, the death ratio is lower than random. Person 1 has the same similarity with random choosing in the first steps but then the death probability drops severely as well. Possibly due to the large field the agent is far away from the ghost agents. We can observe the opposite pattern when looking at person 4: beginning with step 4 the death probability rises while the death ratio is decreasing at the same time. Summarizing we can conclude that the strategies of person 4 and person 3 are similarly good.

Large Environment

Using the larger environment with more ghost agents we can no longer compare the won games and their step number with our verification framework. Nevertheless we can compare the death ratio (picture 5.9) and death probability (picture 5.10).

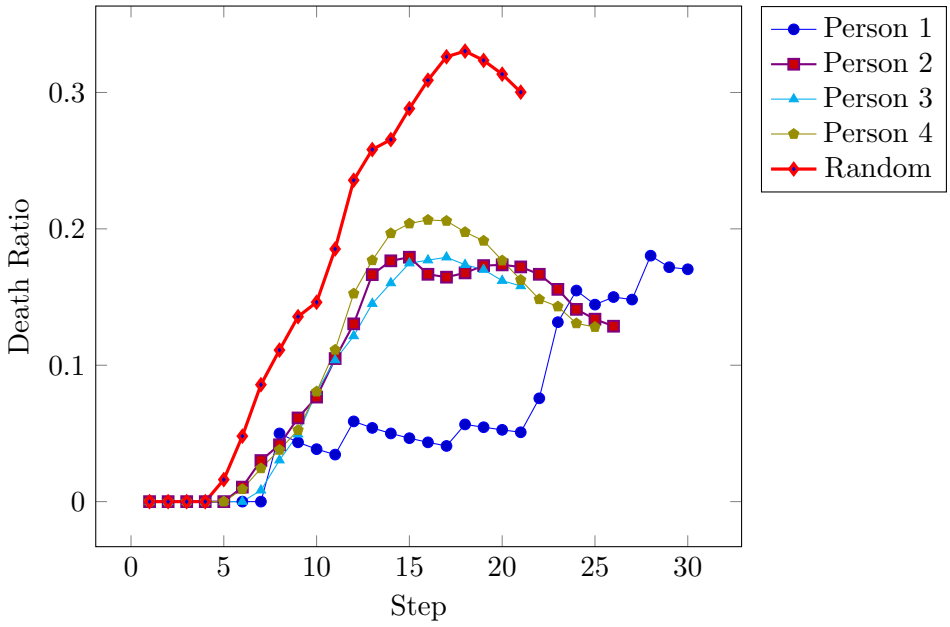


Figure 5.9: death ratio

In picture 5.9 we can see that all human strategies are again below the random strategy. Three of the human strategies are similar while the strategy of person 1 is low and rises step wise.

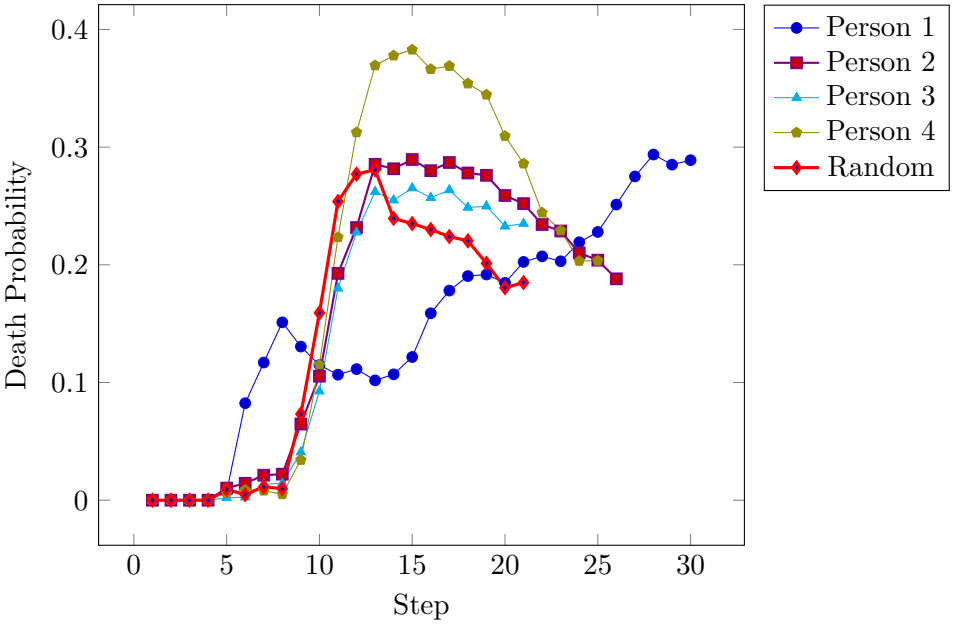


Figure 5.10: death probability

If we now compare the death probabilities from picture 5.10 we see that strategy 2 and 3 are close to the random one while strategy 4 is higher after step 12 and person 1 has a nearly linear rising death probability. Looking closer at person 4 we see that the death ratio rises equally to the death probability. While with person 4 we have the highest death probability but the death ratio is similar to person 2 and 3. Summarizing we can see a clearer difference between person 3 and person 4, so we can conclude that person 4 has the best strategy.

Chapter 6

Related Work

Learning from human behavior is an idea that is attracting more and more attention. There are two general approaches on how to use human input for computational learning. The passive approach is that the human observer approves or disapproves the computer generated behavior. The OpenAI blog [2] published an example on how this approach works. The AI tries to build a reward function that fits the human input. At each iteration, two outcomes of behavior are presented for the human to decide which option is the best. The AI then makes a number of iterations to adapt its behavior.
Closest to our work is [4], which explores the active approach that we used as well. Using a gridworld environment with fixed and moving obstacles and a goal area, the paper explores the quality of human strategies. Strategies need to be specified for all possible feature combinations.
The paper of [1] developed a way to divide visiomotor tasks in the estimated contributions to the reward, using inverse reinforcement learning. In this paper human walk through a parkour while solving different tasks. To the agent the reward is known but it is not known which aspect of the movement is rewarded how much.

Chapter 7

Conclusions And Future Work

Summarizing we showed that behavior imitation is possible. By closely analyzing the behavior data there is a clear improvement over random choosing. Also the different handling of the strategy data has an impact on the performance of the agents. Surprisingly, with a fairly small number of input data-items we could achieve an improvement over random choosing of actions. One reason for this result is our system of storing the states in a relative form. We showed that we can reduce the number of possible states significantly.
For future work we would like to test the actual difference when using strategies for reinforcement learning in terms of attempts needed to learn a policy.

Bibliography

[1] Constantin A. Rothkopf and Dana H. Ballard. Modular inverse reinforcement learning for visumotor behavior. *Biological Cybernetics*, 2013.

[2] Dario Amodei, Paul Christiano, and Alex Ray. Learning from human preferences. https://openai.com/blog/deep-reinforcement-learning-from-human-preferences/, 2017. Online, accessed 1-June-2019.

[3] UC Berkeley. http://ai.berkeley.edu/project_overview.html, 2019.

[4] Steven Carr, Nils Jansen, Ralf Wimmer, Jie Fu, and Ufuk Topcu. Human-in-the-loop synthesis for partially observable markov decision processes. *CoRR*, abs/1802.09810, 2018.

[5] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. *CoRR*, abs/1702.04311, 2017.

[6] Sebastian Deterding. Gamification: Designing for motivation. *Interactions*, 19(4), 2012.

[7] Nils Jansen, Bettina Könighofer, Sebastian Junges, and Roderick Bloem. Shielded decision-making in mdps. *CoRR*, abs/1807.06096, 2018.

[8] Andrej Karpathy. Reinforcejs. https://cs-stanford-edu.ru.idm.oclc.org/people/karpathy/reinforcejs/index.html. Online, accessed 8-June-2019.

[9] Sham Machandranath Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, 2003.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2015.

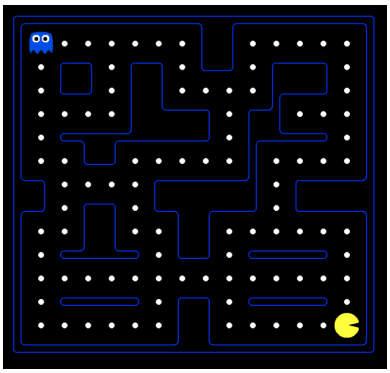
[11] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *CoRR*, abs/1805.01954, 2018.

[12] Michael Wooldridge. Intelligent agents: The key concepts. In Vladimír Mařík, Olga Štěpánková, Hana Krautwurmová, and Michael Luck, editors, *Multi-Agent Systems and Applications II*, pages 3–43, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

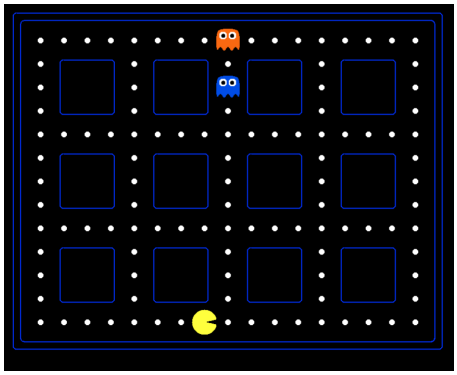
Appendix A

Test Environments

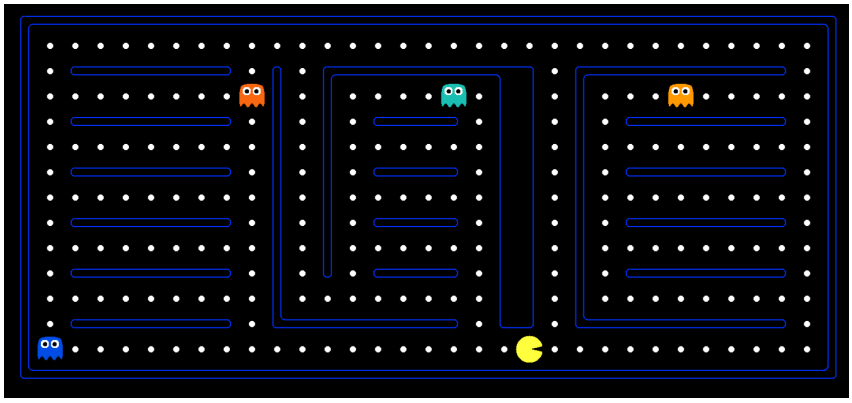
The following test environments where used to collect game data.



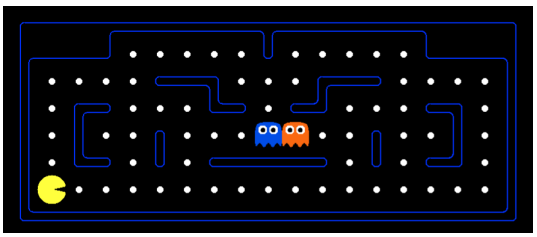
(a)



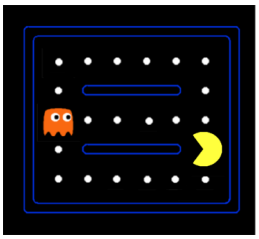
(b)



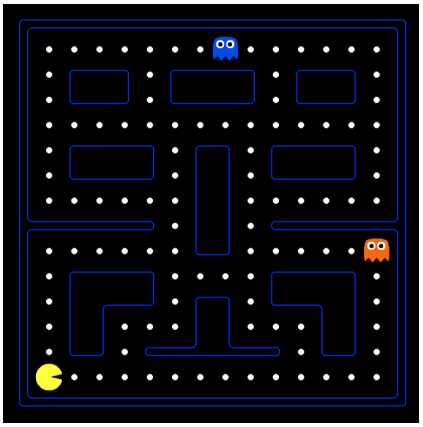
(c)



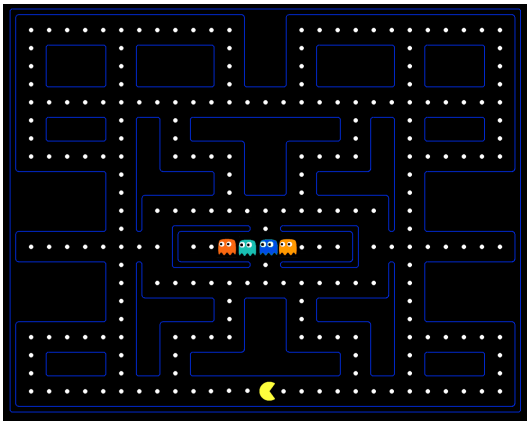
(d)



(e)



(f)



(g)

Appendix B

Test Data

The following tables contain the complete test data for each test person and a table about the number of games in each test environment.

Test Person	Environment						
	(a)	(b)	(c)	(d)	(e)	(f)	(g)
1	4	3	3	3	6	3	3
2	3	3	3	3	7	3	3
3	3	6	3	7	11	6	3
4	6	10	5	14	12	7	5

Table B.1: Number of games

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0.2722	0	0	0
2	3	5	0.5089	0	0	0
3	3	8	0.5681	0	0	0
4	3	11	0.5049	2	0.1818	0
5	1	12	0.4544	2	0.1667	0
6	1	13	0.4582	2	0.1538	0
7	3	16	0.36	2	0.125	0
8	3	19	0.2965	2	0.1053	0
9	3	22	0.252	2	0.0909	0
10	3	25	0.2191	2	0.08	0
11	3	28	0.2078	2	0.0714	0
12	5	33	0.1743	2	0.0606	0
13	5	38	0.1501	2	0.0526	0
14	5	43	0.1318	2	0.0465	0
15	5	48	0.1436	2	0.0417	0
16	9	57	0.1843	3	0.0526	0
17	8	66	0.2124	5	0.0758	0
18	6	73	0.2114	7	0.0959	0
19	4	77	0.1994	7	0.0909	0
20	4	81	0.2151	7	0.0864	0
21	8	89	0.2331	7	0.0787	0
22	8	99	0.2568	7	0.0707	2
23	6	107	0.2809	9	0.0841	0
24	4	112	0.2825	11	0.0982	0
25	2	114	0.2805	11	0.0965	0
26	4	118	0.2698	11	0.0932	0
27	4	122	0.2677	11	0.0902	0
28	4	126	0.2626	11	0.0873	0
29	4	131	0.2736	12	0.0916	0
30	5	137	0.2788	13	0.0949	0

Table B.2: Test Person 1: Small Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0.2722	0	0	0
2	3	5	0.5089	0	0	0
3	3	8	0.5681	0	0	0
4	3	11	0.5049	2	0.1818	0
5	1	12	0.4544	2	0.1667	0
6	1	13	0.4582	2	0.1538	0
7	3	16	0.36	2	0.125	0
8	3	19	0.2965	2	0.1053	0
9	3	22	0.252	2	0.0909	0
10	3	25	0.2191	2	0.08	0
11	3	28	0.2078	2	0.0714	0
12	5	33	0.1743	2	0.0606	0
13	5	38	0.1501	2	0.0526	0
14	5	43	0.1318	2	0.0465	0
15	5	48	0.1436	2	0.0417	0
16	9	57	0.1843	3	0.0526	0
17	8	66	0.2124	5	0.0758	0
18	6	73	0.2114	7	0.0959	0
19	4	77	0.1994	7	0.0909	0
20	4	81	0.2151	7	0.0864	0
21	8	89	0.2331	7	0.0787	0
22	8	99	0.2568	7	0.0707	2
23	6	107	0.2809	9	0.0841	0
24	4	112	0.2825	11	0.0982	0
25	2	114	0.2805	11	0.0965	0
26	4	118	0.2698	11	0.0932	0
27	4	122	0.2677	11	0.0902	0
28	4	126	0.2626	11	0.0873	0
29	4	131	0.2736	12	0.0916	0
30	5	137	0.2788	13	0.0949	0

Table B.3: Test Person 1: Large Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0.0403	0	0	0
2	4	6	0.1787	0	0	0
3	8	14	0.2239	0	0	0
4	13	28	0.197	3	0.1071	0
5	15	45	0.2341	6	0.1333	0
6	20	65	0.2258	8	0.1231	0
7	24	89	0.2197	9	0.1011	0
8	30	119	0.2516	10	0.084	0
9	37	156	0.2306	12	0.0769	0
10	49	207	0.2568	18	0.087	0
11	62	268	0.2595	22	0.0821	0
12	86	354	0.2645	32	0.0904	0
13	105	458	0.2817	37	0.0808	0
14	151	604	0.2753	47	0.0778	0
15	198	815	0.2844	76	0.0933	0
16	276	1,084	0.2786	119	0.1098	0
17	330	1,416	0.2752	164	0.1158	0
18	402	1,801	0.2816	217	0.1205	0
19	484	2,245	0.2638	264	0.1176	0
20	554	2,768	0.2685	312	0.1127	0
21	689	3,423	0.2671	365	0.1066	0
22	899	4,280	0.2751	442	0.1033	2
23	1,135	5,384	0.289	530	0.0984	0
24	1,574	6,908	0.2888	673	0.0974	2
25	2,013	8,948	0.2935	913	0.102	1
26	2,718	11,480	0.2847	1,195	0.1041	7
27	3,299	14,663	0.2799	1,600	0.1091	5
28	4,020	18,324	0.2781	2,014	0.1099	8
29	4,718	22,771	0.2717	2,522	0.1108	8
30	5,644	28,031	0.2768	3,086	0.1101	14

Table B.4: Test Person 2: Small Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0.2722	0	0	0
2	3	5	0.5089	0	0	0
3	3	8	0.5681	0	0	0
4	3	11	0.5049	2	0.1818	0
5	1	12	0.4544	2	0.1667	0
6	1	13	0.4582	2	0.1538	0
7	3	16	0.36	2	0.125	0
8	3	19	0.2965	2	0.1053	0
9	3	22	0.252	2	0.0909	0
10	3	25	0.2191	2	0.08	0
11	3	28	0.2078	2	0.0714	0
12	5	33	0.1743	2	0.0606	0
13	5	38	0.1501	2	0.0526	0
14	5	43	0.1318	2	0.0465	0
15	5	48	0.1436	2	0.0417	0
16	9	57	0.1843	3	0.0526	0
17	8	66	0.2124	5	0.0758	0
18	6	73	0.2114	7	0.0959	0
19	4	77	0.1994	7	0.0909	0
20	4	81	0.2151	7	0.0864	0
21	8	89	0.2331	7	0.0787	0
22	8	99	0.2568	7	0.0707	2
23	6	107	0.2809	9	0.0841	0
24	4	112	0.2825	11	0.0982	0
25	2	114	0.2805	11	0.0965	0
26	4	118	0.2698	11	0.0932	0

Table B.5: Test Person 2: Large Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0.2646	0	0	0
2	5	7	0.3407	0	0	0
3	10	19	0.4724	2	0.1053	0
4	19	40	0.3359	7	0.175	0
5	25	67	0.3846	13	0.194	0
6	28	96	0.3496	19	0.1979	0
7	27	123	0.3362	23	0.187	0
8	26	151	0.3749	25	0.1656	0
9	36	186	0.3552	27	0.1452	0
10	48	238	0.3626	34	0.1429	0
11	70	302	0.3437	40	0.1325	0
12	92	389	0.3276	50	0.1285	0
13	122	505	0.3459	65	0.1287	0
14	160	655	0.3224	87	0.1328	0
15	192	842	0.3387	125	0.1485	0
16	213	1,040	0.3292	151	0.1452	0
17	253	1,280	0.3243	180	0.1406	0
18	322	1,589	0.3289	213	0.134	0
19	429	1,991	0.3224	262	0.1316	0
20	561	2,522	0.3327	319	0.1265	0
21	763	3,236	0.3363	401	0.1239	0
22	1,023	4,189	0.3262	525	0.1253	0
23	1,315	5,398	0.3255	695	0.1288	0
24	1,618	6,847	0.3151	886	0.1294	1
25	1,943	8,629	0.3185	1,137	0.1318	1
26	2,341	10,730	0.3167	1,387	0.1293	2
27	2,898	13,338	0.314	1,717	0.1287	1
28	3,555	16,567	0.3161	2,084	0.1258	3
29	4,600	20,653	0.3122	2,529	0.1225	2
30	5,923	25,964	0.3104	3,131	0.1206	5

Table B.6: Test Person 3: Small Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0.2646	0	0	0
2	5	7	0.3407	0	0	0
3	10	19	0.4724	2	0.1053	0
4	19	40	0.3359	7	0.175	0
5	25	67	0.3846	13	0.194	0
6	28	96	0.3496	19	0.1979	0
7	27	123	0.3362	23	0.187	0
8	26	151	0.3749	25	0.1656	0
9	36	186	0.3552	27	0.1452	0
10	48	238	0.3626	34	0.1429	0
11	70	302	0.3437	40	0.1325	0
12	92	389	0.3276	50	0.1285	0
13	122	505	0.3459	65	0.1287	0
14	160	655	0.3224	87	0.1328	0
15	192	842	0.3387	125	0.1485	0
16	213	1,040	0.3292	151	0.1452	0
17	253	1,280	0.3243	180	0.1406	0
18	322	1,589	0.3289	213	0.134	0
19	429	1,991	0.3224	262	0.1316	0
20	561	2,522	0.3327	319	0.1265	0
21	763	3,236	0.3363	401	0.1239	0

Table B.7: Test Person 3: Large Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0	0	0	0
2	3	5	0	0	0	0
3	5	10	0	0	0	0
4	8	18	0	1	0.0556	0
5	9	27	0.0981	1	0.037	0
6	12	39	0.1461	1	0.0256	0
7	20	59	0.157	2	0.0339	0
8	32	89	0.1373	2	0.0225	0
9	52	136	0.1317	4	0.0294	0
10	80	212	0.126	7	0.033	0
11	131	338	0.162	16	0.0473	0
12	212	536	0.1636	34	0.0634	0
13	290	810	0.2062	67	0.0827	0
14	388	1,156	0.2193	109	0.0943	0
15	442	1,557	0.2529	148	0.0951	0
16	538	2,024	0.2645	200	0.0988	0
17	603	2,559	0.2681	245	0.0957	0
18	726	3,197	0.2711	293	0.0916	0
19	919	4,019	0.2665	343	0.0853	0
20	1,226	5,142	0.2648	424	0.0825	0
21	1,691	6,713	0.2683	538	0.0801	0
22	2,359	8,847	0.2704	735	0.0831	2
23	3,038	11,586	0.2833	977	0.0843	0
24	3,996	15,043	0.2793	1,316	0.0875	1
25	4,801	19,335	0.2843	1,757	0.0909	2
26	6,091	24,599	0.2797	2,268	0.0922	5
27	7,430	30,994	0.2749	2,855	0.0921	10
28	9,257	38,942	0.2718	3,531	0.0907	13
29	11,686	48,915	0.2648	4,347	0.0889	23
30	14,846	61,700	0.2635	5,454	0.0884	22

Table B.8: Test Person 4: Small Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0	0	0	0
2	3	5	0	0	0	0
3	5	10	0	0	0	0
4	8	18	0	1	0.0556	0
5	9	27	0.0981	1	0.037	0
6	12	39	0.1461	1	0.0256	0
7	20	59	0.157	2	0.0339	0
8	32	89	0.1373	2	0.0225	0
9	52	136	0.1317	4	0.0294	0
10	80	212	0.126	7	0.033	0
11	131	338	0.162	16	0.0473	0
12	212	536	0.1636	34	0.0634	0
13	290	810	0.2062	67	0.0827	0
14	388	1,156	0.2193	109	0.0943	0
15	442	1,557	0.2529	148	0.0951	0
16	538	2,024	0.2645	200	0.0988	0
17	603	2,559	0.2681	245	0.0957	0
18	726	3,197	0.2711	293	0.0916	0
19	919	4,019	0.2665	343	0.0853	0
20	1,226	5,142	0.2648	424	0.0825	0
21	1,691	6,713	0.2683	538	0.0801	0
22	2,359	8,847	0.2704	735	0.0831	2
23	3,038	11,586	0.2833	977	0.0843	0
24	3,996	15,043	0.2793	1,316	0.0875	1
25	4,801	19,335	0.2843	1,757	0.0909	2

Table B.9: Test Person 4: Large Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0.3583	0	0	0
2	5	10	0.4595	3	0.3	0
3	7	18	0.5167	5	0.2778	0
4	12	33	0.4812	10	0.303	0
5	16	56	0.463	23	0.4107	0
6	17	77	0.3832	28	0.3636	0
7	27	105	0.3445	33	0.3143	0
8	46	150	0.2977	37	0.2467	0
9	79	231	0.3292	51	0.2208	0
10	139	397	0.3447	106	0.267	0
11	184	610	0.371	194	0.318	0
12	224	845	0.3453	280	0.3314	0
13	244	1,120	0.3391	381	0.3402	0
14	308	1,451	0.3339	484	0.3336	0
15	403	1,852	0.3179	573	0.3094	0
16	583	2,415	0.3133	657	0.272	0
17	982	3,389	0.2932	856	0.2526	0
18	1,504	4,876	0.2967	1,222	0.2506	0
19	2,160	6,971	0.3135	1,768	0.2536	0
20	3,002	9,979	0.3137	2,689	0.2695	0
21	3,811	13,732	0.312	3,936	0.2866	0
22	4,615	18,147	0.2988	5,276	0.2907	0
23	5,783	23,476	0.2736	6,410	0.273	0
24	8,002	30,807	0.2747	7,708	0.2502	0
25	12,381	41,764	0.2631	9,740	0.2332	0
26	17,070	57,685	0.2753	13,261	0.2299	2
27	24,896	79,381	0.2774	18,369	0.2314	0
28	31,213	108,066	0.2846	26,178	0.2422	14
29	40,125	142,570	0.2816	35,336	0.2479	4
30	47,334	184,579	0.2806	46,898	0.2541	25

Table B.10: Random: Small Environment

Step	# States in Step	# Total States	Death Probability	# Death States	Death Ratio	win States per Step
1	2	2	0.3583	0	0	0
2	5	10	0.4595	3	0.3	0
3	7	18	0.5167	5	0.2778	0
4	12	33	0.4812	10	0.303	0
5	16	56	0.463	23	0.4107	0
6	17	77	0.3832	28	0.3636	0
7	27	105	0.3445	33	0.3143	0
8	46	150	0.2977	37	0.2467	0
9	79	231	0.3292	51	0.2208	0
10	139	397	0.3447	106	0.267	0
11	184	610	0.371	194	0.318	0
12	224	845	0.3453	280	0.3314	0
13	244	1,120	0.3391	381	0.3402	0
14	308	1,451	0.3339	484	0.3336	0
15	403	1,852	0.3179	573	0.3094	0
16	583	2,415	0.3133	657	0.272	0
17	982	3,389	0.2932	856	0.2526	0
18	1,504	4,876	0.2967	1,222	0.2506	0
19	2,160	6,971	0.3135	1,768	0.2536	0
20	3,002	9,979	0.3137	2,689	0.2695	0
21	3,811	13,732	0.312	3,936	0.2866	0

Table B.11: Random: Large Environment