UNIVERSITÄT BASEL

UCT for Pac-Man

Bachelor Thesis

Department of Computer Science

Examiner: Prof. Dr. Malte Helmert

Manuel Heusner
manuel.heusner@stud.unibas.ch
08-051-526

November 10, 2011

UNI
BASEL

Abstract

This is the written documentation of the implementation of an agent for Ms-Pac-Man for a Ms. Pac-Man simulation. The agent uses the UCT algorithm (upper confident bounds) which is becoming more prominent in the domain of artificial intelligence research. In this document you will find a description of the UCT algorithm, statements to the implementation and an evaluation of the performance.

Table of Contents

Abstract i

1 Introduction 1

1.1 Preamble 1

1.2 Motivation 1

1.3 Outline 1

2 The Ms. Pac-Man simulator 3

2.1 The game 3

2.2 The software 3

2.3 The rules 4

3 The UCT algorithm for Ms. Pac-Man 6

3.1 The algorithm 6

3.2 Reward 8

3.3 Back-propagation and updating 9

3.4 Rollout policy 9

3.5 Tree policy 10

3.6 Terminal conditions 11

4 The implementation evolution 12

4.1 The basic implementation 12

4.1.1 Rollout policies 13

4.1.2 The parameters 14

4.2 Change the root of the tree 14

4.3 Minimize the tree 14

4.4 Minimize the required memory 15

4.5 Use a discount factor 16

4.6 Use subtree from last tree 16

4.7 Make the controller competitive in the competition mode 17

5 Experiments and results 19

5.1 Experiments 19

5.2 Information extraction and visualization of data 19

5.3 General results 19

5.4 Specific results 21

6 Comparison to the other controllers 24

6.1 Comparison to entrants of last competition (CIG11) 24
6.2 Comparison to other well documented MCTS based Ms. Pac-Man agents . . 25

7 Conclusion **27**

Bibliography **29**

Appendix **30**
.1 Tables with parameter settings and results of the experiments 30

Declaration of Authorship **33**

1 Introduction

1.1 Preamble

Artificial intelligence is a challenging area in computer science. Its playground is often the domain of games. The problem of games can be solved with different AI techniques. One of them is the UCT algorithm [1], which is based upon the Monte-Carlo Tree Search (MCTS) combined with solutions for the Multi-Armed Bandit Problem. UCT raises attention since it was successfully applied for Computer Go [2]. It mostly finds appliance in board games, but other domains also proposed the use UCT for their applications, as for example the arcade game Ms. Pac-Man or modifications of it.

The first tries in that direction came from the Ritsumeikan University [3] and from the University of Essex [4], which were motivated by the Ms. Pac-Man Simulator, developed by the Game Intelligence Group of the University of Essex.

1.2 Motivation

The motivation for this thesis was to create an agent for Ms. Pac-Man. The agent uses an UCT algorithm and is competitive against given ghosts controllers of the Essex' Ms. Pac-Man Simulator.

1.3 Outline

In the second chapter, the Ms. Pac-Man Simulator and the rules of the game against the original rules will be described.

The purpose of the third chapter is to show how the UCT algorithm is working, using the example of the Ms. Pac-Man game.

The fourth chapter explains some more details about the basic implementation and further versions and discusses the ideas and solutions of each version.

The fifth chapter turns to the discussion of the parameter optimization.

The sixth one talks about experiments and results.

The seventh chapter makes a comparison of our agent to other agents.

Finally, the last chapter compares our agent to other Ms. Pac-Man agents.

2
The Ms. Pac-Man simulator

2.1 The game

Ms. Pac-Man is an arcade game, which is based on the original Pac-Man game. It was developed and released by Midway games in 1981. It contains new maze designs and some changes in the gameplay. Five players participate the game. They are Ms. Pac-Man and the four ghosts, which are named Blinky, Pinky, Inky and Sue. Ms. Pac-Man is controlled by an human while the enemies are controlled by the game machine. Ms. Pac-Man earns points by eating pills, power pills, fruits and eatable ghosts, She has to take care of the ghosts, which can takes her lives. Ms. Pac-Man walks through one maze, until all pills are eaten. Then she steps to the next of the four mazes, which are shown in figure 2.1. After the end of the last maze, the first maze will be played again. Each step into a next maze is a new level. The game becomes more difficult with each level, but can run infinitely long. The aim of that game is to get a new high score.

2.2 The software

The software is an efficient simulator of the Ms. Pac-Man game written in Java. It provides simple interfaces with which self-created controllers can be connected. It also offers plenty of useful functions, which allow us to get information about the current state of the game. It is possible to implement controllers for Ms. Pac-Man and for the ghost team. There are two possibilities of execution:

- The synchronous mode and
- The asynchronous mode

The synchronous mode is good for testing the controllers, because its execution time depends on the running performances of the controllers.

The asynchronous mode advances the play every 40 milliseconds, regardless to what the controllers do. With this simulator they want to provide an environment, within which it is possible to implement and test single agent and multi-agent strategies. This simulator makes it possible to compare controllers to each other. The members of the group of the University of Essex set up a competitive environment on the web, where several competitions take place. More detailed information about the simulator and the competitions can be found on *pacman-vs-ghosts.net*.

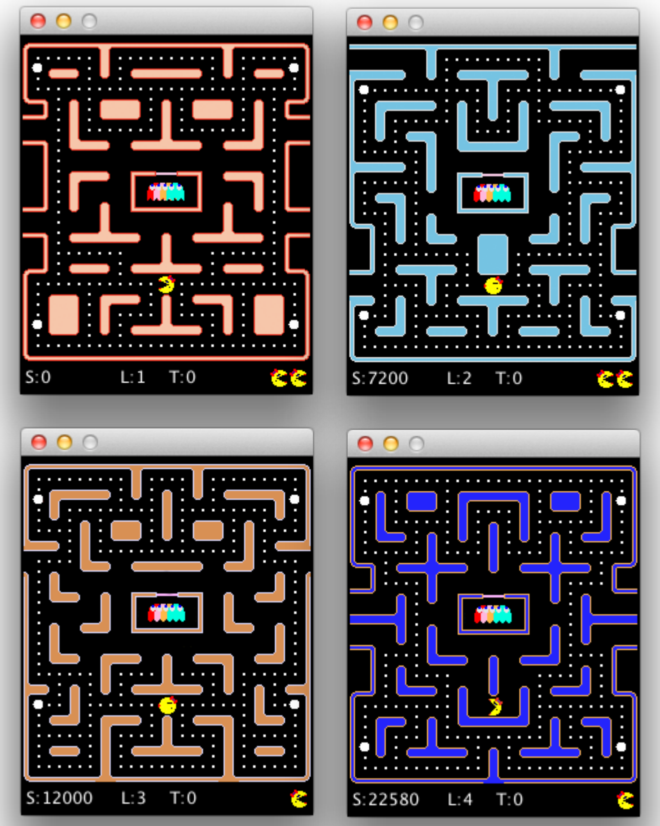

Figure 2.1: The four mazes [A,B,C,D] of the Ms. Pac-Man simulator

2.3 The rules

The game rules are nearly the same as the ones of the original Ms. Pac-Man game. There are some modifications to simplify the game to make programmers' lives easier. Differing from the original game the bonus fruits are omitted. Ghosts and Ms. Pac-Man always have the same speed, except for the time frame in which the ghosts can be killed. Then the speed of the ghosts is half the usual speed. The ghosts have other behaviors than in the original game, as it is possible to compete against other ghosts' strategies than the original one. To prevent the ghosts to spend too much time in blocking the power pills, the level time is limited to 3000 ticks. The remaining pills are awarded to Ms. Pac-Man. For the implementation of an arbitrary strategy, two rules are given to the ghosts to let the plays be fair. First the ghosts are not allowed to go back. That means they only have the choice between directions at an intersection. The second constraint is given by the simulator and cannot be influenced by the controller. It is the random reverse, which occurs with a probability of 0.0015 and gives Ms. Pac-Man chances to escape from tricky situations.

In the game, rules exist for the distribution of points, but for the UCT algorithm these rules are not relevant. Ms. Pac-Mans aim is to maximize the score, while the ghosts are interested in minimizing it.

3 The UCT algorithm for Ms. Pac-Man

In this chapter the UCT algorithm will be explained in connection to the Ms. Pac-Man game. The UCT algorithm is an ad-hoc planning algorithm, which samples future states to get the most optimal action for the current state of the "world". For the sampling of future states, the *Monte-Carlo Tree Search* is combined with a solutions for the *Multi-Armed Bandit Problem*. The solution is the *First Upper Confident Bounds Theorem* (UCB1), which clarifies the origin of the name of this algorithm.

More detailed information can be taken from the work of Levente Kocsis and Csaba Szepesvári [1] in which UCT are beeing introduced as well as from the presentation of Alain Fern about Monte-Carlo-Planning [5].

3.1 The algorithm

This section describes the process of the algorithm, while the next sections will explain in detail some concepts.

At each state Ms. Pac-Man has to decide which *action* to take next. She doesn't know anything about the consequences of any of her actions. She only knows that the aim is to reach the highest *score*. She has to check out all actions to achieve the knowledge about further *states*. Which action she will explore next is defined in a *rollout policy*.

The algorithm selects an action, which brings the play to one of the unknown states. At that next state the algorithm tries out an action again. Those expansions will be repeated up to a *terminal state*.

The trial of actions from the root state to a terminal state is called *simulation*. Plenty of simulations build up a *tree*. Each simulation gives a reward which can be evaluated at the terminal state. The reward will be *back-propagated*. This means, that the *utility* of all nodes, which were visited by the simulation will be *updated* according to an *update function*.

When all possible actions of a state are tried out once, one child node exists for each action. Each child node contains an utility, which is the utility of the action given at the parent state. After a few simulations, Ms. Pac-Man can choose actions of states based on the utility values. The selection of the next action based on the utility values is defined in the *tree policy*.

The tree policy determines the simulations path through the existing tree, due to the fact that all actions of the nodes in the tree are already tried out. After the simulation found a path through the tree, the rollout policy takes place and will discover further states.

The algorithm can be summarized as follow:
The UCT algorithm performs plenty of simulations. Each simulation contains two tasks. The first task is to find an optimal path through the existing tree based on the utilities of state-action pairs. The second task contains the discovery of further states. Each simulation gets a reward at the end, which will be back-propagated to all affected nodes. The nodes update their utility values after each simulation. At the very end, the algorithm gives back the optimal action. This algorithm is also depicted as a pseudo-code in 3.1, whereas the figure 3.1 illustrates one situation.

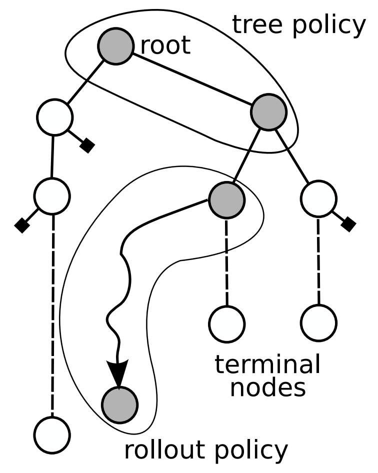

Figure 3.1: One situation of the search tree

Ms. Pac-Man is not the only one, who has interests in this game. The ghosts want to eat Ms. Pac-Man to bring the game to a fast end, because their aim is to minimize the score. Each state requires an action from each participant to advance the game. The tree has to be enhanced by nodes, which belong to the ghosts. The root node of the tree has to be Ms. Pac-Mans node, because the algorithm will select the best action for her after cutoff. A Pac-Mans node means the node, where Pac-Man picks an action. Otherwise, ghosts nodes are the nodes, where the ghost team picks one. The two different nodes alternate in a simulation. Figure 3.2 shows the situation of the nodes among the state.
There is one single node type for all ghosts. The action of the ghost team is the combination of the possible directions, which each ghost can take. Because they share one node-type, the combined action and not each single one will be appraised. One single utility value for each ensemble action may effect the ghosts acts more like a team.

The ghosts also have a rollout and a tree policy, which differs from Ms. Pac-Mans. The reward function and the update function are not affected by the inclusion of the opponents

Algorithm 3.1 UCT-Algorithm

```
Require: state
  create root node which contains the state
  while no timeout do
    start at root node
    while state in node is no terminal state do
      if node is completely expanded then
        get the next action from tree-policy
        select child-node which belongs to the action
      else
        get next action from rollout-policy
        advance the state using the action
        create node which contains copy of advanced state
        insert node to the tree
      end if
    end while
    reward terminal state
    back-propagate reward
  end while
  return action at the root which has the highest utility
```

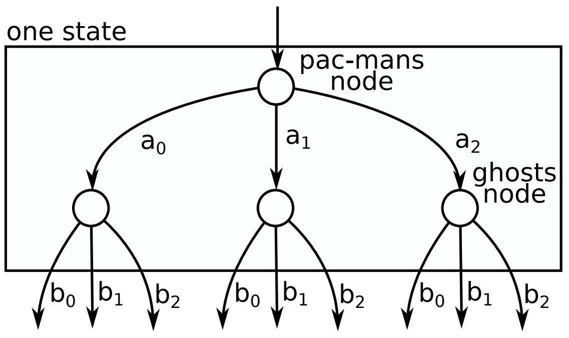

Figure 3.2: State with Ms. Pac-Man and ghost team nodes

into the tree. In figure 3.2 each out-coming arrow leads to a new state. The new state is the result of ghosts' and Ms. Pac-Man's actions.

3.2 Reward

After reaching the terminal state, a reward will be granted. The reward is made up of the score and the time of the simulation. The time in the game is measured with the amount of ticks. Ms. Pac-Man needs about 4 ticks to come from one pill to the next.

Taking the time element into the reward is useful for Ms. Pac-Man and the ghost team. Both will recognize that more time gives higher rewards, moreover they can better distinguish between the utilities of actions. Let me explain the need of the time in the reward, by an example in which no time is used:
When only one pill remains in the maze, but it is too far away from Ms. Pac-Man, eventually no simulation will recognize that pill. All the simulations will have the same scores, as well as the actions' utilities will hold the same value. The algorithm will recognize no differences

between the utility-values of actions. As a consequence, it will choose one action randomly, which is not optimal.

In this game the score is the given measurement of the strength of Ms. Pac-Man and the ghosts. It is not necessary to invent another evaluation scheme for simulations.

The advantage of this rewarding method is, that it is quite intuitive and gives a meaningful evaluation of the simulated steps. The disadvantage is the scalability of the reward. The score has no clear upper boundary. Therefore, it is futile to set clear boundaries for the reward values and scale it. This fact becomes a problem in the rollout policy and will be discussed in section 3.4

3.3 Back-propagation and updating

After the terminal state is evaluated and the reward is determined, the result has to be back-propagated. Back-propagate the result means to update all utility values of the nodes, which were involved in the simulation.

The utility of a node is defined as the average of all rewards of simulations that went through that node. Formula 3.1 and 3.2 show the update function as it was defined in [6], which simply averages over the rewards.

$$n(s) \leftarrow n(s) + 1; \tag{3.1}$$

$$u(s) \leftarrow u(s) + \frac{1}{n(s)}[R - u(s)] \tag{3.2}$$

Note, the utility of a node can also be interpreted as the utility of the action, which was performed on the parents state and which yields to the current state: $n(s) = n(s_{parent}, a)$ and $u(s) = u(s_{parent}, a)$. The update function does not depend on the node-type. It is exactly the same for Ms. Pac-Man's and ghosts' nodes.

3.4 Rollout policy

How to expand the nodes is defined in the rollout policy. To expand a node means, trying out an action and create a new node for that action, which will be added to the tree. At each state in the game, Ms. Pac-Man and the ghosts have a finite set of possible actions. Possible actions mean legal directions they may choose with respect to the rules of the game. These actions are randomly chosen by the rollout policy of the UCT algorithm.

The selection of actions can be optimized by defining another policy. One approach orders the actions in a specific way. A further approach only allows a subset of actions. The best actions of a state will be performed first and the worst last or never. The algorithm can curb the action space by evaluating the current state of the simulation and by deleting the actions which are hopeless. This can be done by defining handcrafted conditions. The actions, which do not satisfy all the conditions, will be deleted. A priority is given to actions, which satisfy subsets of conditions.

These approaches provide early rollouts towards promising states and do not waste time with less promising simulations.

The algorithm uses two rollout policies for expanding the tree. One for Ms. Pac-Man and one for the ghosts team.

The policy also defines the strategy of a competitor. The strategy of the simulated Ms. Pac-Man could be the same as in the real game. But the true strategy of the true opponent is previously unknown and can not be influenced in any way. The random rollout policy for the ghost team can bring the best results in games against an arbitrary ghost's strategy.

Ms. Pac-Man would be much better, if the Ms. Pac-Man agent would know the exact strategy of the current enemy. Then, the agent could respect ghosts' behavior in the planning process. This assumption was tested out, by defining a rollout policy, which contains the true ghost strategy. The scores of the games were actually higher. A well documented example, which shows the improvement of using the *known model* of the ghosts, is shown by S. Samothrakis and D. Robles and S. Lucas [4].

To reach such a state of optimality in plays agains arbitrary ghosts teams, the agent should learn the strategy of the adversary by evaluating their movements from previous states.

3.5 Tree policy

Are there no more actions, which can generate new nodes, the tree policy instead of the rollout policy decides, which action to take. The tree policy gives back the action which may lead to the best result. There is no guarantee that the most promising action ever get back good rewards. The most promising action is the one which gives the highest utility. Sometimes the algorithm should explore another action which has a lower utility but may lead to better results.

The previously described problem is called *exploitation vs. exploration trade-off* and is a concept of the Multi-Armed Bandit Problem. That concept tries to find a mechanism to make the selection of an action, not depending only on the utility value but also depending on the experience. The mechanism gives opportunity to the less promising actions to discover higher rewards. According to the UCT algorithm the *First Upper Confident Bounds Theorem* (UCB1) [1] has to be used here. The UCB1 gives a trade-off between the exploration and exploitation based on the utility-value and the number of times an action was called. The expression 3.3 shows the UCB1 formula, which must be maximized with an action 3.4. The current node selects the action which gives back the highest value from that formula.

$$UCB1(s,a) = u(s,a) + c\sqrt{\frac{\log n(s)}{n(s,a)}} \tag{3.3}$$

$$\arg\max_{a} UCB1(s,a) \tag{3.4}$$

The utility of the action is placed in the left term which is called exploitation term. Comparing exploitation terms between the actions shows us, which is the most promising direction at this moment. The more experience is at hand for an action, the more accurate the utility is.

Actions with less experience should get a chance to be selected. Therefore, the other term is required to make the choice of the action depending on the experience. That term is called exploration term. The less experience an action has, the higher the value of that term will be. On the other hand, more experience shrinks the value towards 0. The constant c inside that term sets the sensitivity of the exploration and depends on the game. The value for c can be evaluated empirically to optimize the algorithm. If the value is too small, the algorithm will take the most promising action too many times and may miss really good rewards. If it is too large, the algorithm will waste time by trying out actions that lead to obviously bad results.

The tree policy is nearly the same for Ms. Pac-Man as for the ghosts team. Formula 3.3 is valid for the simulated Ms. Pac-Man. The left term must be replaced by the negative utility value for the ghosts team.

3.6 Terminal conditions

The most intuitive terminal state of the game is the game-over situation. If the game-over would be the single terminal condition, the simulations would run too long moreover the depth of the tree would be very large. The longer the durations of simulations are, the less simulations can be done in the limited time. The agent can choose the next action more accurately, if more simulations are sampled which check out more different paths through the maze. The number of steps of a simulation must be limited to achieve a higher quantity of simulations. Therefore, we have to introduce additional terminal rules to achieve shorter simulations.

Meaningful terminal conditions are when Ms. Pac-Man is eaten by a ghost or when the time limit for a level is reached.

The first terminal condition prevents Ms. Pac-Man from running into the ghosts, because each other direction would give a higher reward due to the longer simulations. The second terminal condition implies, that Ms. Pac-Man does not waste time at the end of a level by avoiding the ghosts, but that she should eat the remaining pills instead of avoiding the deadly enemy. Despite of the two terminal conditions, the simulations can still be very long.

To avoid long simulations, the set of conditions must be enhanced by a condition that limits the time of a simulation. Limiting the time of the simulations, means the same as limiting the depth of the search tree. It is not easy to find the optimal depth because it depends on the computation time of the algorithm and the performance of the machine. Long simulations are not actually bad. They would be suitable, if there would not be a time limit, which forces us to find a trade-off between collecting knowledge about far away states and collecting knowledge about nearer state.

4 The implementation evolution

Some different versions of the UCT algorithm were implemented during the project. More time was spent optimizing the data structure than changing the UCT algorithm itself. Some versions are based on previous versions and contain some further approaches. This section will discuss the basic implementation, further implementations as well as the ideas and the problems behind them.

The presentation of Alain Fern about Monte-Carlo-Planning [5] gives the main inspirations for this implementation. Unfortunately, the article of S. Samothrakis and D. Robles and S. Lucas [4] which talks about Monte Carlo Tree Search for the same Ms. Pac-Man simulator was missed in the literature research. Therefore, their work could not be included in the designing process of this implementation.

4.1 The basic implementation

The basic implementation is the first implementation of this *UCT for Pac-Man* project. The skeletal structure of this first version is described in chapter 3. This section describes some more details about the implementation.

Ms. Pac-Man only makes decisions, if the rollout policy allows to do so. If she is not allowed, she will keep up her way through the maze. If the policy enables Ms. Pac-Man to make a decision, the planning process will be performed by building up a search tree.

Each action at the root state will be selected equal times during the planning process. In the next section (4.2), a better approach to handle the root state will be discussed.

In the Ms. Pac-Man game a situation exists, which must be addressed. It is the random reversal of ghosts.

First, we ignore that case and do not take it into account for the implementation of the algorithm. There could be two simulations, which select identical actions. After the ghosts randomly reversed in one simulation, the simulations will not be equal anymore. Reversing ghosts' simulations are not usual. If we store such a simulation in the tree, most of the following simulations could also be incorrect.

The random reversal occurs with a probability of 0.0015. In other words, it occurs after about every 666 steps. This doesn't seem to be that much. But if we take into consideration that the algorithm creates about 100 simulations and each simulation advances plenty of states, it would lead to about 10000 states, stored in the tree. Finally, there could be about 15 inconsistent states in the tree. The nearer such an inconsistent state is to the root, the less accurate the decision of the agent will be. In the weirdest case Ms. Pac-Man runs directly into a nearby ghost without any motive. To avoid this behavior the algorithm must detect the random reversals.

One solution would be to expect the ghosts' reversal at each state. The tree would be too complex because at each state the ghosts group could decide between 16 and 256 actions. 16 actions, if all ghosts are in a tube and 256 actions, if all ghosts stand in intersections. The branching factor of the tree would be enormous, which avoids accurate results. Even if the time for planning would be unlimited or the machine performance extremely high, MS. Pac-Man would not perform better results. The agent would expect ghosts, which block all escape ways in every situation. Ms. Pac-Man would remain on her position, because the agent does not know a better solution. In that case, the agent assumes the wrong ghosts' strategy.

The better remedy is to eliminate the random part from the simulation. Unfortunately, the framework does not provide a function to check, if a random reversal happens. The simplest approach to detect random ghosts' reversals is to compare the current state with the previous state. The ghosts can reverse at two events. One event is the random reversal. The other event is, when Ms. Pac-Man gets a power pill. Then, the ghosts turn into the edible state and go back. These two cases can be simply differentiated by checking, if a power pill has been eaten. If no pill was eaten, the reversal was random. If one is detected, the simulation simply retries the step until it can be sure to have a proper next state. It is usually the case after one single repetition.

The complexity of the tree is much smaller, if random events are avoided in the simulations. Ghosts will have the choice between 0 and 81 actions per state. It is quite rare to get 81 choices for the ghosts, because the probability that all ghosts stand at crossroads is very small.

4.1.1 Rollout policies

This and all further versions provide an interface for the rollout policy. It is the same concept as the interface for the controllers. While the controller decides which step to do next in the real game, the rollout policy decides which step to do next in the simulation. The interface simplifies the implementation of different policies. Two different rollout policies were created, which are worth to be noted.

The first rollout policy contains all possible actions which are sorted randomly. Ms. Pac-Man may remain at the position, if the algorithm uses that policy. It is the well known back-and-forth problem of search trees [4].

The second rollout policy eliminates that behavior by restricting Ms. Pac-Mans action-space at some states. Ms. Pac-Man only has free choices, if she or the ghosts stands at an intersection. At these states, really seminal decisions have to be made. This makes Ms. Pac-Man to act more straightforward in the play.

4.1.2 The parameters

This version of the implementation contains the parameter *c* in the UCB1 formula and a parameter, which constraints the depth limit of the tree. Besides that, a parameter for the time limit exists. It is limiting the computation time for a decision. The parameter can be optimized to get better scores in the games. The values for these parameters have to be evaluated empirically. Due the fact that the trees are built up randomly, the scores between the games show big differences. This variance makes it impossible to recognize some tendencies depending on the parameter. A closer look to different parameter settings will be given in the chapter 6.

4.2 Change the root of the tree

In the basic version, the algorithm selects each possible action of the root node equal times during the planing process. That means, each less promising action will be selected as often as the most promising one. In some cases, it is clear after few simulations, which of the actions are disadvantageous.

Let me assume the situation as follow: Ms. Pac-Man stands in a tube, a ghost is following her and one single escape way is free. In this case, the action towards the ghost should not be repeated a lot of times, because taking that action is obviously hopeless. Such a hopeless simulation does not take much time. Nevertheless, that problem should be handled. One solution is given by the definition of the tree policy. The UCB1 formula, which is used by that policy, avoids to select the disadvantageous action too many times. The only thing to change, is to provide the functionality of the tree policy to the root node. This approach is better than the other one, which wastes lots of time in taking wrong actions. All further versions are based on this version.

4.3 Minimize the tree

In order to save time and memory-space, the search tree must be minimized. It is not possible to minimize the tree, when Ms. Pac-Man has the freedom to choose between all actions. Then, each node has at least two actions.

If the action space of Ms. Pac-Man is restricted to one possible action at a state and the ghost also have no choice at the same time. That state does not have to be stored into the tree, because no one have to make decisions. Only states which branch the tree will be stored.

Afterwards, the following simulations can use the existing tree and do not have to care about the irrelevant states. The tree is minimal and contains still the same amount of information as the tree from the previous version.

4.4 Minimize the required memory

In the previous versions of the implementation, copies of game states were stored in the tree. Each node contains a copy of the game state, which is required for the expansion of nodes to build up the tree. The copy is deleted after all the actions of a node were tried out once.

The inspection of a tree, which stands at the end of its life, shows that all nodes at a deep level contain a copy, while just some nodes nearby the root have been deleted. A tree stands at the end of its life when the due time is reached.

The main problem is that the amount of nodes raises rapidly with the depth of the tree. The deeper the tree, the more memory is required. But memory space is limited. The solution will be, to store as little information as possible in the nodes.

The game state is only needed during a simulation. More precisely, it is needed at the rollout part of the simulation. The game state must only be provided to a node, which will be expanded. This can be done by advancing the initial game state at the root to the required state, where a node will explore a new action. It is possible to reproduce a game state for a node instead of saving it in the node. The reproduction of states needs the actions. But until now, only the utilities of actions were stored but not the action itself, which is being represented as a pointer to the next node.

The action itself is encoded as one or a set of integers, which represent the directions. These integers are needed for the game state to advance the game. The action encoded as integers as well as the utility value will be placed in the next node. If we assume, there is no contingency in the simulation, each state will be fully reproducible by using the action/state pairs along one path. Now, only one game state is saved in the tree, which is the initial state at the root. It is needed as the base and must be copied for each simulation before using it for the reproduction of following states.

The most important advantage of this approach is that less memory space is required. An improvement of the time efficiency is doubtful. An advantage of using less memory per time is, that the garbage collector of the Java Virtual Machine must not free the space so often and leaves more time to the algorithm.

So the question remains, which one of the two is more time efficient, the copying state or the advancing game. In the previous version, the game state had to be copied and advanced once for each state. In this version the game state must be copied once per simulation and is advanced for each state within a simulation. For each update, plenty of rules will be checked, which also takes time.

This version may not be more efficient in time than the previous one. Rather the opposite will be the case if we take in to account the copying of states to detect the random reversals of the ghosts.

This version is not stable, because some errors occur due to the reproduction of states in connection to the *minimal tree* approach, which is described in the section 4.3. It can't be excluded that the error originates from the Ms. Pac-Man-Simulator, because a bug exists in the functionality of recording and reproducing the game.

4.5 Use a discount factor

Sometimes, it seems that Ms. Pac-Man behaves fearful in the game. The reason for that behavior may comes from the fact, that the agent plans too far. For long simulations, the difference of the rewards is big. Also the variance grows with the length of the simulations. That problem is known as the horizon problem and there exists an approach to handle it. The approach was found in the Artificial Intelligence book in chapter 17 [7], which proposes to enhance the rewarding function by a discount factor. That discount factor gives weight to the states depending on its depth in the tree. Using this approach should effect, that Ms. Pac-Man will act more purposeful.

Untill now, the reward was simply the sum of the score and the duration of the simulation. That definition can be kept, but we have to reorganize the calculation. The total score must be split up into individual scores for every state of the simulation. The same approach has to be taken for the total time of the simulation. Then, the individual values can be weighted.

The individual value is the difference between the last state value and current state value. The modified reward formula is shown below 4.1.

$$R = \sum_{i=1}^{n} \lambda^{i-1} \cdot \Big((score_i - score_{i-1}) + (time_i - time_{i-1}) \Big) \tag{4.1}$$

The discount factor λ has to be a value greater than 0 and smaller than 1. To get the desired effect, the factor should be near to 1.

If the value is too small, Ms. Pac-Man acts risky and seems to be blind for states that are further away. If, for example, she stands in a tunnel, which is filled with pills and both escape routes are free, she will concentrate to get nearby pills and does not seem to notice the ghost until it is too late. Then, the ghosts have already blocked both escape ways.

If the factor is set to a value greater than 1, it is too big and Ms. Pac-Man seems to be blind for near states. If she is in a tunnel, as in the example above, she will concentrate on not running into a ghost, although the ghosts are far away from her. She will remain on the middle of the tube because each escape route seems hopeless to her. While she remains at that place for a long time, the ghosts are truly blocking both directions. In that situation she seems to have no interests in getting the nearby pills. She even has no interest to eat nearby eatable ghosts. The optimal discount factor has to be evaluated attentively.

4.6 Use subtree from last tree

At each step of the game an entirely new search tree is built up. One part of the tree can be used for the next step. Then, the time which is available for that next step can be used to get more accurate results.

As Ms. Pac-Man and the ghosts step to the next state of the real game, they can step into the corresponding next state in the tree. The next state in the tree should be the same as the actual state of the game except if there was a random reversal.

If the reversal of ghosts occurred, the current tree must be casted away, because the turnaround of ghosts was not respected in the tree. Then, an entirely new tree has to be created.

Rewards and utility values depend on the depth of the tree, respectively on the duration of simulations. There might be some inequalities between utility values of actions. The inequalities are implications of the reusing of trees and the displacing of the root node. The depth limit is relative to the root node at every step of the game. If the depth limit is large, parts of the trees may not be visited by every reusing of the tree. The rewards of simulations, which have their terminal states in such a part and were not visited for a certain time, will be outdated. Those rewards will no longer be equitable to rewards of new simulations.

The solution to remove inequalities will be to advance all terminal states from the last tree and to get the new rewards, which will be back-propagated. Before any reward will be back-propagated, the utility value of all nodes will be set to 0. After the update of utility values, the values will correspond to the current tree.

This solution was not implemented, because it seems more important to use the given running time to find new paths through the maze than to explore the existing paths. This implementation risks inconsistencies of the utility values in the tree, by not handling that problem.

This approach leads to an immense growing of the tree. Therefore, this version requires a memory-light tree. Because of the instabilities, which were mentioned in sectioni 4.4, it is just possible to re-use about 3-10 previous states. Unfortunately, this instability has bigger effects in this version. The re-usability of the nodes depends on the proper creation of the tree. Much time was spent to solve that problem, but the source of the bug is not yet found.

4.7 Make the controller competitive in the competition mode

In contrast to a simple agent, which contains only handcrafted rules, the accuracy of results of the UCT algorithm strongly depends on the computation time. The aim is to use every millisecond up to the due time.

All versions of this implementation encounter the problem of keeping the due time. At most steps, the algorithm cannot give back the chosen direction for Ms. Pac-Man on time. The corresponding thread sleeps some time, overruns the due time and wakes up, if it is too late. The interruption of threads can not be avoided due to the operating system, that has the authority to distribute the processors' resources to processes and threads.

Observing the due time is required in the asynchronous game mode, which is used in the competition against other AI controllers. In that mode each controller runs in a thread. After each 40 milliseconds the game will be updated, regardless if the controller returns a response or not. If the game does not get back a direction from the controller at time, it will simply reuse the last action or select an action randomly, which will lead to very poor results. The late response may then be used for the next step, which is also not correct.

No satisfactory solution was found. One solution would be, to send the most accurate direction to the simulator at every moment. The Ms. Pac-Man Simulator does not supply that functionality. Modification within the simulator do not make sense, because you can not prefer your own version of the simulator at the official competitions. A possible solution to the due time situation is to give the action back to the game before it runs out. Even then, the thread could oversleep the due time. The performance would also decrease, because the agent has less time to find the best action. The best solution would be, if the simulator would provide access to a variable, in which the direction can be stored at every moment within the 40 millisecond time range. Then, the agent could put the most current choice of the direction to the variable.

5 Experiments and results

5.1 Experiments

Some experiments were done to evaluate the stability of the game score. Different versions of the implementation and parameter settings were used. The results of the different experiments are compared to each other.

The ghosts were controlled by the controller, which is named Legacy. This controller contains the ghosts' AI of the original Ms. Pac-Man game. The experiments always run in the synchronous mode. The game was played $20'000$ times for each experiment and its parameter setting.

The configurations of the experiments and their statistical summary can be found in the appendix 7.

5.2 Information extraction and visualization of data

The scores of all plays within an experiment give some information about the expected value and its double sided 95% confidence interval and the standard deviation. A histogram of the distribution of the score was plotted to read out some more interesting results. Also, a histogram over time was created, whereas the time is the amount of ticks.

5.3 General results

The expected values of the scores of each experiment differ from each other. The distinct double sided confident intervals add authority to these differences.

The best expected score of a controller, which responds to the 40 ms time limit, is 15422 points. The highest score is 58350 points.

Independent from the parameter settings, the histograms depict interesting distributions of the game scores, which resemble a mixture of two hills, whereby the first one is cut off at the score of 10000. It is the implication of the bonus life, which Ms. Pac-Man gets with a score of 10000 points. The histogram is shown in figure 5.1 a). It would be interesting to see the shapes of the distributions, if there would be no bonus life. Two additional experiments were triggered to show the two distributions separately. One experiment gives three lives to Ms. Pac-Man without the bonus life. The other experiment gives her four lives, also without a bonus live. The results show, as it was expected, two single distributions, which look like hills. The two histograms are shown in figures 5.1 b) and c).

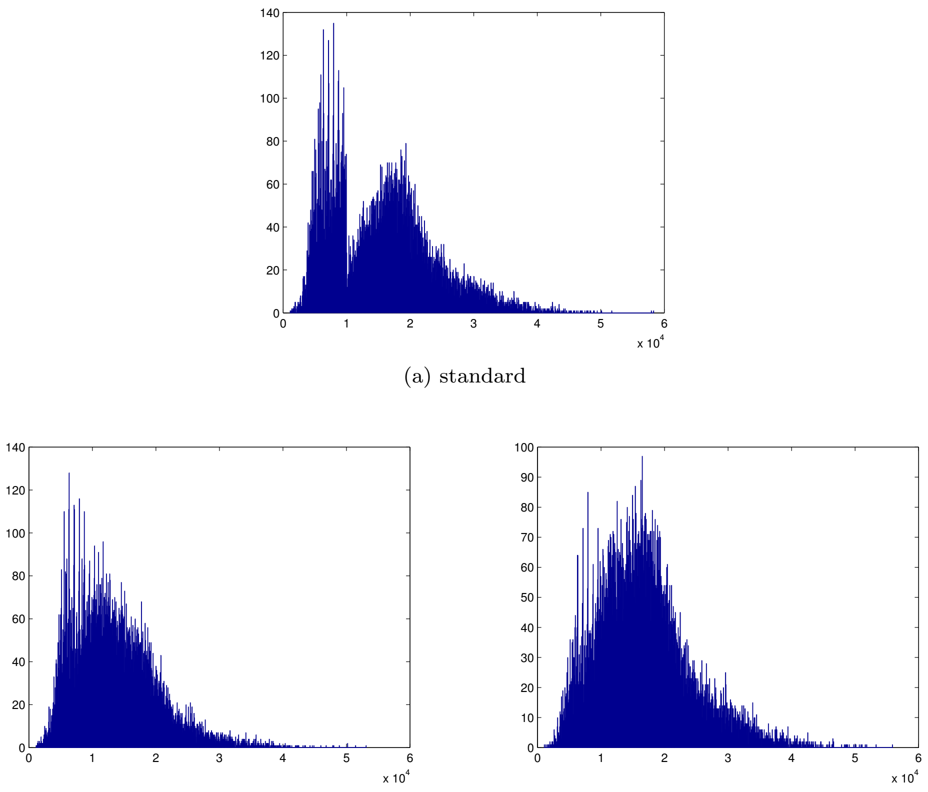

Figure 5.1: Histogram of the score

Figure 5.2 depicts the histogram of the durations of the games. It shows a noticeable peak at around 3000 steps. Some lower peaks are also recognizable at around 6000 and around 9000 steps. The time limit is the reason for these peaks. The investigation of the first peak shows, that the cause of the game over lies at 2998 and 2999 time steps. One of the terminal rules is the reason for that problem. That rule prevents Ms. Pac-Man to enter the next level due to the level time limit. She should rather try to enter the next level by eating all pills.

At the end of the level the following situation could occur: Ms. Pac-Man runs in the middle of a tube and eats pills while ghosts block the exit in the direction of the pills. Due to the level time limit, Ms. Pac-Man thinks about five steps ahead. She will take the direction to the pills and the ghost, because it gives the higher reward within the five remaining steps. For the other direction, only the five steps would be awarded, because there are no pills.

The time limit constraint can be removed, because one rule of the Ms. Pac-Man Simulator was ignored during the implementation . The rule says: If the level time limit is reached, the

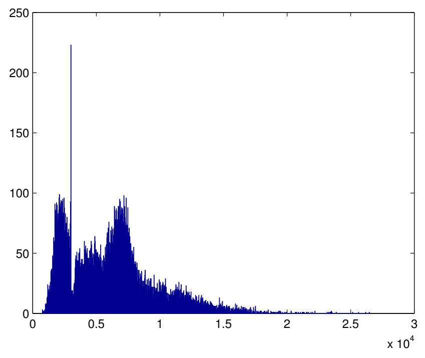

Figure 5.2: Histogram of the time

points of the remaining pills will be awarded to Ms. Pac-Man. Therefore, it is not important for Ms. Pac-Man to get all pills in a maze.

It is interesting to know, how many simulations were performed to build up a tree for one decision. The average number of simulations, which build up a tree, was evaluated as accurate as possible. For the experiment, the depth limit of the search tree was set to 40, which means that each simulation contains a maximum of 40 decisions.

For each game, the average number of simulation per decision state was logged. The values lie in a range between 500 and 4000. The statistical evaluation gives a mean of 1046 and a standard deviation of 331. That seems to be pretty good.

By investigating the number of *simulations per decision state* within a game, it is apparent that the range of the values lies between 10^2 and 10^5 . Figure 5.3 depicts the histogram of all numbers of simulations per decision state of one game. To have a better view, only the range between 0 and 2000 is included. Only 3% of the values lie in the range from 2000 to 10^5 . Nevertheless, there is a high variance. The reached tree depth is the reason for that variance. If all terminal states are nearby the root state, the simulations do not take much time. Then, plenty of simulations can be performed. Otherwise, if the simulations reach the depth limit more often, they take more time, but their amount is smaller. The histogram 5.3 shows, that the algorithm samples between 100 and 300 simulations for each decision, which has to be made for the real game.

5.4 Specific results

The main feature of the UCT algorithm is the constant c in the second term of the UCB1 formula. Two experiments were performed to investigate the consequences of different values for c. Both have the same settings, except for the c constant. One experiment was done with $c = 100$ and the other with $c = 300$. The difference of the expected values is as high as 3034 points, whereas 100 is the better value for c. This is an improvement by factor 1.245. These experiments used the improved rollout policy, which only allows decisions at intersections. It is difficult to say, if a lower or a higher value than 100 would perform better results. To answer that question more experiments would be required.

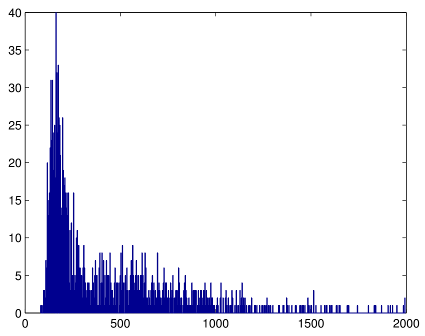

Figure 5.3: Histogram of the number of simulations per state

Other experiments were done by using another policy. This policy does not constrain the action space of Ms. Pac-Man. The parameter settings differ from settings, that were used for the experiments and which are described above. Nevertheless, the results show that the experiments using the improved rollout policy are much better. Looking at the results of the experiments, in which both experiments use the same value for the constant c, the improved policy reached an average score of 12388 points, while the other policy reached 5495Rpoints. The elimination of the back-and-forth problem by creating a new policy increased the mean score by a factor of more than 2.

It would be interesting to know, how the performance of the Ms.Pac-Man controller depends on the depth limit of the tree. One experiment had limited the depth of the tree to 40 decision states, the second experiment had limited it to one million, which can be assumed as infinite. The results show, that the lower depth limit performs better scores. This result was expected, because it is better to try out many short paths through the maze instead of trying out only a few very long paths. A more detailed explanation is given in one of the previous chapters. The difference between the expected values comes up to 1343 points. That is an improvement by the factor 1.122.

The controllers have 40 milliseconds to decide, which action should be taken for the current state of the game. What would be the consequence of increasing that time? To give an answer to that question, an additional experiment was performed, in which the time between two steps was increased by factor 3 upon 120 ms. The results shows a difference between the scores of 2137 points, which increases the expected value by 17.3 percent. The results might even be better, if the depth limit would be adapted to the time limit.

The discount factor was also evaluated. One experiment had been carried out with the discount factor of $\lambda = 0.998$. The highest discount in a simulation is $0.998^{40} = 0.923$, where the depth limit is 40. Unfortunately, no improvement was detected. The differences of the expected values are up to 835 points, which represents a degradation of the factor 0.933. The discount factor has not been investigated in more detail. A further experiment should be done with an enlarged discount factor.

The last experiment investigates the effect of the random reversals of the ghosts to the game. The question is, how does this implementation of the UCT algorithm handle those events.

An additional experiment was started, which does not allow random reversals of the ghosts. The difference of the expected values is 2340 points, whereas the scores of the games with random reversals are worse than the scores of the games from the additional experiment. This big difference was not expected. On one hand, the random reverses should be a handicap to the ghosts and not for Ms.Pac-Man. On the other hand the UCT algorithm is an ad-hoc planning algorithm, which should be able to handle such spontaneous and unforeseeable events as well as possible.

6 Comparison to the other controllers

There exists one "controller", which is unbeatable. It is the human being himself. The world record is hold by Abner Ashman with 921360 points. No AI performed better results yet. The annual competitions on pacman-vs-ghosts.net show a continuous improvement of the controllers. The first section compares the performance of the best controller version from this project with two other controllers, which attended the last competition. The second section shows the differences of this approach to the approaches of two other MCTS based Ms. Pac-Man agents.

6.1 Comparison to entrants of last competition (CIG11)

The controller, which got the third place at the last competition (CIG11), follows an approach, which combines Monte Carlo Tree Search with handcrafted rules. The controller is named *ICE pAmbush CIG11* [8]. It got an average score of 19969 over 10 games competing against the controller, which is named Legacy. M. Nakamura, K. Q. Nguyen and R. Thawonmas met the same time limit problem as mentioned previously in chapter 4. They too found no other solution than to return the output direction 10 ms before due time. The controller of this *UCT for Pac-Man* project used the whole time of 40 ms between the steps to reach the average score of 15422. The experiments in this project have been run in the synchronous mode instead of the asynchronous mode. Therefore, the controllers are not comparable. Nevertheless, the result of the ICE pAmbush CIG11 controller is much better.

The controller which got the first place uses n recursive procedure. It is named Spooks [9] and received an average score of 69800 points over 10 games. Noticeable is the low variance between the scores of the plays. The low variance could be based on a less random character of the algorithm.

6.2 Comparison to other well documented MCTS based Ms. Pac-Man agents

The first approach, which is part of the following comparison, is described by S. Samothrakis, D. Robles and S. Lucas [4]. They follow the max^n approach, which is an algorithmic solution of n-person games [10]. According to that solution, they model the Ms. Pac-Man game as a five player turn-based game. Ms. Pac-Man and each ghost possess own nodes in the tree. Each player tries to maximize its payoff. In this *UCT for Pac-Man* project, the same approach was used. Instead of modeling a five player turn-based game, a two-player turn-based game was modeled.

To reduce the complexity of the tree, they never let Ms. Pac-Man step back in the simulations. In the real game, she is allowed to take every direction. In this *UCT for Pac-Man* project, Ms. Pac-Man has the same rules in the simulations as in the real game.

The group used a different rewarding approach, which distributes some discrete values between 0 and 1 which are independently of the score. Their biggest priority was, to keep Ms. Pac-Man alive by avoiding the ghosts. A long life will give high scores, but not the highest. In a second step they took the pills, power pills and edible ghost into account, to get higher scores. In this *UCT for Pac-Man* project, less attention was given to keeping Ms. Pac-Man alive.

The results of their experiments are not comparable to the results of the experiments of this project. They used a modified version of the Ms. Pac-Man Simulator, in which Ms. Pac-Man has one single life, the edible time is almost nonexistent and reversals of ghosts are eliminated. Even though these restrictions are hard, the experiments produced average scores of 81979 in the synchronous play mode.

Older versions of controllers are also not comparable to my own implementation, because they use screen-capturing to read out the state of the game from the original Ms. Pac-Man game. Screen-capturing needs precious time, so that less time remains for algorithm.

Nevertheless, the best MCTS based Ms. Pac-Man agent today [11] also uses screen-capturing and performs remarkable average scores of 31106 points. That agent was designed by N. Ikehata and T. Ito. They have found a mechanism to avoid pincer moves of the ghosts. The approach is to evaluate the survival rates of Ms. Pac-Man in tubes by using MCTS. They use a rewarding scheme, which depends on death or life and not on the score as in this project. The reason of not using the score as the reward is that the best score does not necessarily guarantee the best behavior of Ms. Pac-Man.

They propose that it is more important to consider the sub-goals. The sub-goals are:

- avoiding contacts with ghosts,
- eating all pills and
- eating as many ghosts as possible.

That proposal makes sense, because the high score seduces Ms. Pac-Man to enter dangerous situations and to ignore the ghosts. They introduced a *danger level map*, in which the pills are labeled with a priority. The map gives prior knowledge to the agent. Pills, which are in dangerous locations, will be eaten in an earlier state of the level, because in a later state the ghosts are more active. In dangerous locations, it is more difficult for Ms. Pac-Man to escape. The agent has no prior knowledge in my implementation. But it would be a useful approach for further optimizations.

7 Conclusion

In this thesis a basic UCT algorithm was implemented. Some optimizations of the data structure have been made, but not all of them were successful. The most promising versions of the implementation have been evaluated.

The evaluation shows the expected results, as the improvement of scores by using a well-considered rollout policy. Other results are not expected as the impairment of the scores by using a discount factor.

During the implementation period, it appeared that no clear value could be found for the constant c, which is in the UCB1 formula. But the results of the evaluation show a big difference between the scores by using different values for c. That result underlines the importance of including the First Upper Confidence Bounds Theorem into the Monte-Carlo Tree Search.

This work is not breaking any record. Nevertheless, it is a motivation to read more about Monte-Carlo Tree Search and its application for Ms. Pac-Man and other games.

Further steps:
In this implementation, the states of the game were copied to detect random reverses and terminal states. A better approach would be to create an own class for simulated game states. That own class would contain all required functionality and data, which are used for the UCT algorithm. This approach would prevent the algorithm to copy each state for comparisons and therefore the performance would hopefully raise.

As it was mentioned in the section about experimental results 5.4, it is hard to find a satisfying value for the c constant in the UCB1 formula, because of the high variance of the rewards. A solution is already discussed by P. Auer, N. Cesa-Bianchi and P. Fischer [12]. It would be interesting to see the result of using the UCB-tuned formula, which is described in their work and gives the possibility to handle the high variances.

Ms. Pac-Man often takes the route through the long tube at the bottom of the maze [C] in figure 2.1 at the end of a level. Ms. Pac-Man steps into that tube, because the agent plans little ahead. If Ms. Pac-Man is in the middle of it, the agent cannot make clear decisions, because one exit seems to be as good as the other. While Ms. Pac-Man stands there, the ghosts block both exits, which leads to the death of Ms. Pac-Man. That pitfall could be easy handled with patterns, which would be defined in the rollout policy. The patterns would prohibit her to step into that tube in some situations. More about the application of patterns in Monte-Carlo Tree search can be found in the work of S. Gelly, Y. Wang, R. Munos and O. Teytaud [13].

It would be easy to design a controller for the ghosts by taking the implementation of Ms. Pac-Mans controller by making a few little changes to the algorithm. The biggest change would be to set the ghost team node to the root. All other changes belong to that restructuring. The result would be interesting, also for the improvement of Ms. Pac-Man controller.

Bibliography

[1] Kocsis, L. and Szepesvári, C. Bandit based Monte-Carlo Planning. *Machine Learning: ECML 2006*, 4212:282–293 (2006).

[2] Gelly, S. and Wang, Y. Exploration exploitation in GO: UCT for Monte-Carlo GO. In *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop Canada (2006)* (2006).

[3] Nguyen, K. Q., Miyama, T., Yamada, A., Ashida, T., and Thawonmas, R. ICE gUCT. Technical report, Intelligent Computer Entertainment Laboratory, Ritsumeikan University (2011).

[4] Samothrakis, S., Robles, D., and Lucas, S. Fast Approximate $Max - n$ Monte Carlo Tree Search for Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):142–154 (2011).

[5] Fern, A. Monte-Carlo Planning: Basic Principles and Recent Progress. In *tutorial*, 20. International Conference on Automated Planning and Scheduling (2010).

[6] Fern, A. and Lewis, P. Ensemble Monte-Carlo Planning: An Empirical Study. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*, pages 58–65 (2011).

[7] Russel, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson Education, 3 edition (2010).

[8] Nakamura, M., Nguyen, K. Q., and Thawonmas, R. ICE pAmbush CIG11. Technical report, Intelligent Computer Entertainment Laboratory, Ritsumeikan University (2011).

[9] Tose, D. Spooks design notes. Technical report, private (2011).

[10] Luckhart, C. and Irani, K. B. An Algorithmic Solution of N-Person Games. In *AAAI'86*, pages 158–162 (1986).

[11] Ikehata, N. and It, T. Monte-Carlo Tree Search in Ms. Pac-Man. *IEEE Conference on Computational Intelligence and Games*, pages 39–46 (2011).

[12] Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2–3):235–256 (2002).

[13] Gelly, S., Wang, Y., Munos, R., and Teytaud, O. Modification of UCT with Patterns in Monte-Carlo Go. Rapport de recherche 6062, Institut National de Recherche en Informatique et en Automatique (2006).

Appendix

.1 Tables with parameter settings and results of the experiments

experiment	c	discount factor	depth limit	time limit	rollout policy	characteristics
A	100	1	100	40	*basic*	
B	300	1	100	40	*basic*	
C	300	1	40	40	*improved*	
D	300	1	10^6	40	*improved*	
E	300	1	40	120	*improved*	
F	300	0.998	40	40	*improved*	
G	100	1	40	40	*improved*	
H	100	1	40	40	*improved*	3 lives, without bonus live
I	100	1	40	40	*improved*	4 lives, without bonus live
J	100	1	40	40	*improved*	no random reversals of ghosts

Figure 1: Parameter settings of experiments

experiment	expected value	standard deviation	min	max
A	5660.2070	3024.6405	460	26970
B	5495.3795	2972.6481	540	30690
C	12388.2730	6669.6078	950	52600
D	11045.5440	5816.6589	820	43030
E	14525.7180	7696.9588	990	56990
F	11553.1470	6165.8353	1050	44270
G	15422.0875	8120.2971	1100	58350
H	13468.0195	6545.8294	1120	53130
I	16675.1955	7355.1583	1060	55920
J	17761.7935	9406.8721	1120	65530

Figure 2: statistical evaluation of experiments

Declaration of Authorship

I hereby declare that this thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the bibliography and specified in the text.
This thesis is not substantially the same as any that I have submitted or will be submitting for a degree or diploma or other qualification at this or any other University.

Basel, date

Author