

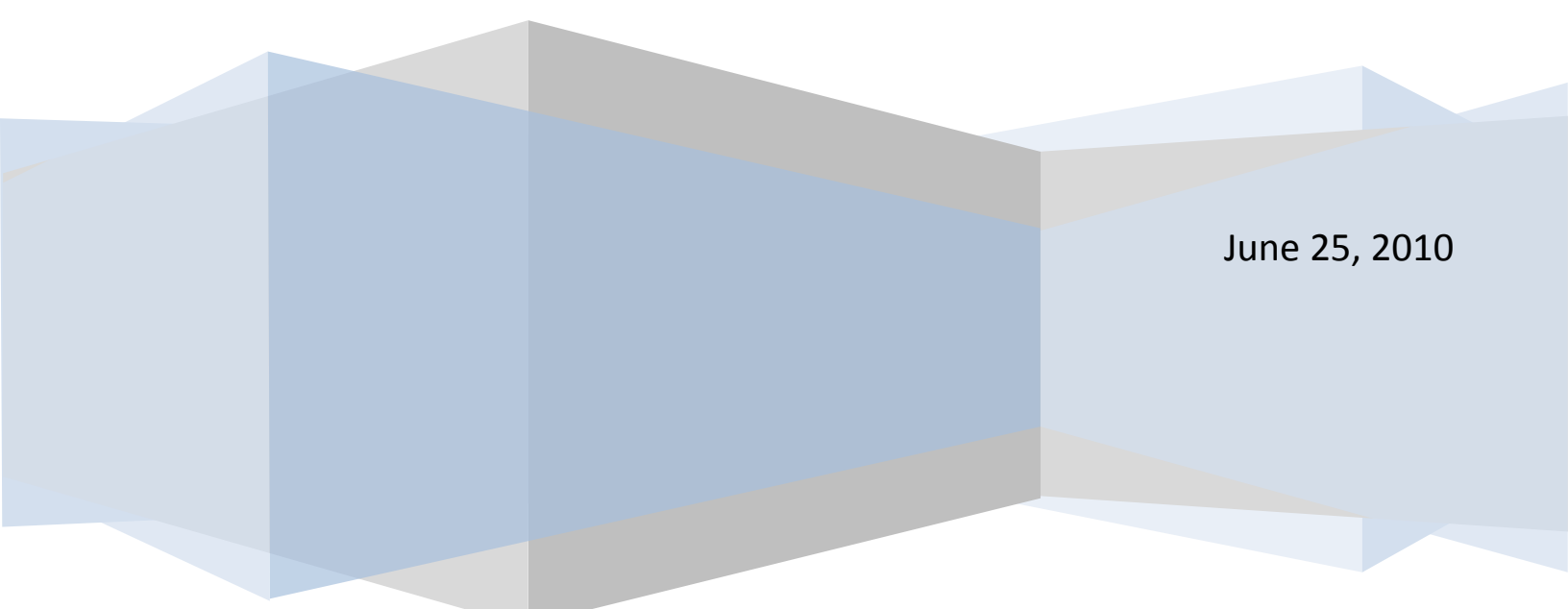
Delft University of Technology – Computer Science

# Pacman

## Bachelor Project

D.W.H. Wilmer, G.R. de Ridder, A.A. Kol and D.C. Harkes

June 25, 2010



## Table of Contents

---

Table of Contents .....	2
1 Introduction .....	3
2 Process .....	5
3 Requirements .....	9
4 Project Design .....	12
5 Architecture .....	14
6 Image Processing .....	18
7 Robot control .....	22
8 Artificial Intelligence .....	26
9 Results .....	29
10 Conclusion .....	31
Appendices .....	32

## 1 Introduction

---

The Pacman project was initiated by, and is now displayed at the Science Centre Delft. With this project, as with many others, the Science Centre Delft is showing what Delft University of Technology is capable of. They wish to do this by displaying interesting, fun, active and informative exhibits. Our project is exhibited in the robotics hall to show the talents of the computer science department of Delft University of Technology. The projects within the Science Centre are coordinated by a company named Tinker Imagineers. Budget issues or changes we wished to make to the original game were first discussed with Tinker.

### 1.1 The Game

---

The project entails simulating the classic Pacman computer game using real robots. One robot, Pacman, is controlled by a player and has to fulfil certain assignments. Simultaneously the other robots, monsters, try to catch Pacman. A camera captures the movements of all the robots and with the help of image processing the camera input is used to run a control program. The same image is also used to display information on monitors. This gives the player and other visitors an insight in the inner workings of the game.

Other than being fun and informative, an important feature of the game is being robust. The game will be exhibited in the Science Centre and played day-in and day-out and this must run without a glitch.

### 1.2 The Set-up

---



Figure 1-1: Artist Impression

An artist's impression of the complete set-up is shown in the Figure 1-1. First of all there is a table for the robots to drive on. The table has the layout of the original Pacman game including a maze made up of walls. Behind the table, monitors show details about the game. The Science Centre proposed to display details such as: the code executed, the robots' strategy and a map of the game layout. The most important detail is the webcam hanging above the table. This key element is the main input for the program. Also visible in the artist's impression are the robots. Obviously these are necessary, although nine robots will be too many for the table with an area of 1.6 square metres. The addition of the word

MOWAY in combination with the robot is much more significant. Moway is a Spanish company which makes the robots the Science Centre advised us to use.

### 1.3 The Project

---

The project description left room for interpretation. Our knowledge on many aspects was needed to set-up and to implement this project, especially on the topics: software engineering, artificial intelligence, image processing and embedded systems. Decisions on the aspects of hardware and software were completely up to us as long as we ended up with a working game. For example, it was our choice to use Java as our main programming language for this project. Also the choice of controller, the Xbox controller, and the use of Moway robots are examples of decisions we had to make.

These choices were mostly made during the design period; however some turned out to be unpractical and were changed during the project implementation. An example of this is the absence of walls on the table. Visitors are invited to create obstacles by, for example, placing their arms above the table. This resulted in a game that is much more interactive as we found when testing the game with a group of children – referred to as test group in the rest of this document. Another example is the addition of many different levels, which has made the game more fun to play. Changes such as these were possible due to the incremental and iterative planning; as the next section will explain.

## 2 Process

---

This section our approach to the Pacman project, the highlights and the pitfalls of the project will be discussed. The creation of the plan and planning will be explained first. After that, it is discussed how the process continued and how we did or did not stick to our original plan, and why.

### 2.1 Iterative Plan

---

Our full-time project started on April 19<sup>th</sup>, but to be able to start full-time on this date and to get approval from Tinker a plan was made in advance. We took artist's impression and project description as guidance and brainstormed about what features the game should have and what would be possible to build in ten weeks – the time set for this project.

We made a list of functional and non-functional requirements (See also Section 3) and divided them into subsystems. For example, one of the requirements was that the robots should be able to drive to a specific position on the table.

The process was chosen to be iterative and incremental, divided in several phases. This would ensure a result after each phase of two weeks and allow us to change the next phase, if necessary. We divided the requirements in phases and made a list of milestones for each phase. All milestones introduce new functionality and only build on previous milestones. For example, one of the milestones was that robots should be able to drive to a specific position on the table, parallel with the corresponding requirement. This milestone is built on the previous milestones of recognizing the robots using the webcam image and instructing the robots to drive.

We made a graph of milestones and their dependencies. In strategy games, this is often called a tech tree. Our tech tree is divided into four phases as can be seen in Figure 2-1 (A larger version of the tech tree is attached as Appendix C):

- *Phase I:* First working version
- *Phase II:* Artificial Intelligence
- *Phase III:* Walls on the Table
- *Phase IV:* High Score and robustness

The milestones – which are explained in more detail in Section 3 – are divided into four subsystems:

- Robot Control (green in the Tech Tree);
- Graphical User Interface (GUI) (shown in red);
- Image Processing (shown in blue);
- Game (black in the Tech Tree).

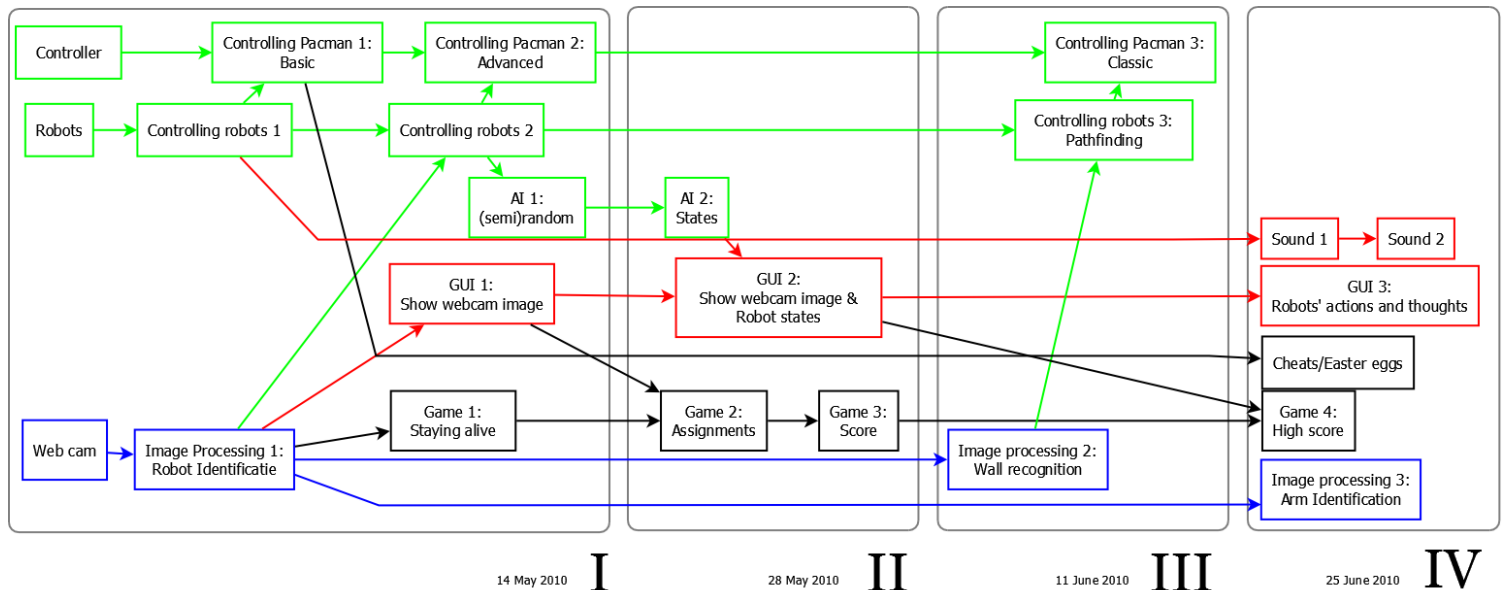


Figure 2-1: The Tech Tree, divided into four phases. Phases are to scale, milestones are not.

The first phase was stripped down too much. Therefore it was not actually able to run a day without crashing. For example, if robots collided with each other or with the wall, they would no longer be recognised. The result of Phase I was a game that would often crash; the milestones were met but the game did not run smooth until Phase II.

As we progressed, we came to new insights about what was feasible and what could improve the game. Also, our minds were changed by the test group. Not only did the children give very useful suggestions, also how they reacted on the game was very helpful for further development of the game.

Because we made our plan iterative in the beginning, it was easy to change the milestones for Phase III and Phase IV based on our new insights. With the observations of the test group we decided to remove the walls from Phase III. This made it possible for visitors to use their arms and hands as walls instead.

## 2.2 Iterative System Design

---

Not only the plan and the milestones were subject to change. The programming itself was iterative too. All subsystems – explained in more detail in Section 5 – were rewritten at least once.

### 2.2.1 Subsystems Redesign

---

What started as test code evolved into subsystems, which were hooked together to create a working application. The necessity of creating structured subsystems was noted quite early on. Test code grew beyond what was feasible without structure therefore design patterns were used to rewrite the code. This made the subsystems easier to comprehend, use and implement. The number of errors in the code was greatly reduced by a clear and simple structure of each subsystem.

As an example we can take a look at the control of the robots. What started out as a list of wrapping methods for sending commands directly to the robots, evolved into a command pattern with a high level of abstraction. Now a robot can be told to drive to a specific point or into a specific direction. These commands can even be queued and repeated so patrolling behaviour can be created.

Of course we could not have designed or redesigned the subsystems without the knowledge we gained by the preceding iterations of exploring what is possible for a certain case and what is not. Next to that, if the system is going to be improved in the future it is good to redesign subsystems. When new functionality is added the subsystems may again grow too large for their current structure. This iterative loop will therefore continue as long as software is being built and improved. To maintain the possibility of restructuring subsystems, they should be loosely coupled and their interfaces should be clear to ensure that the inner workings of a subsystem can be changed without major problems for other subsystems.

### 2.2.2 Architecture Redesign

---

While structuring and rewriting the different subsystems was quite easy, designing the complete architecture was harder. While outer subsystems (Robot Control, Image Processing and Xbox Control) were easily moulded into layers, the central subsystems (Engine and Game) were spaghetti code (an unorganized mess of statements) until we redesigned them completely in week seven of our project. Because the software was not properly structured, there were bugs that would occur sometimes based on thread timing and execution sequence. It took a week to rewrite it all, but after that it worked perfectly.

## 2.3 Getting Help

---

At first we were mostly trying things ourselves, using the internet and our own knowledge as main information sources. When we could not find out how to get robot control to work, we got the advice to seek help. This really helped us, because after we contacted Moway we received the help we were seeking. This is explained with more detail in Section 7.

The Image Processing went fine at the start, however once we got it up and running there were a lot of different options to improve it and make it more stable. We asked Dr. E.A. Hendriks for advice on which

options we should use. He suggested we should use histograms to detect the different colours, which we did as explained in detail in Section 6.



## 3 Requirements

---

As in any other project, there is a set of requirements that must be fulfilled to make it a success. The requirements mentioned in the introduction were those from the Science Centre and they were not very detailed. During the first week, an extensive planning was made evaluating the requirements made by the Science Centre and creating the missing, but necessary, requirements. The planning, available in appendix B, shows all requirements as milestones. These milestones were divided over the different subsystems and phases.

### 3.1 Subsystems

---

An overview of the subsystems and phases is depicted in the tech tree, Figure 2-1. The following subsections will review the requirements by their corresponding subsystem. The hardware requirements are discussed in a separate subsection.

#### 3.1.1 Robot Control

---

The Robot Control subsystem has four milestones in Phase I. The first requirement was that the robots must be controlled by the computer. This was done by instructing each wheel to turn in a certain direction (forward or backward) with a certain velocity. Using these simple instructions and knowing the robot's position with help of the image processing, the next requirement was to have the robots drive to specific positions. The third and fourth requirements are specific to Pacman. Pacman is driven by a controller, but we came up with different methods of control. The third and fourth requirements are therefore two different steering methods: the basic method and the advanced method. The basic method works like a remote controlled car would: pushing the stick forward and back causes the robot to drive forward and back, while pushing the stick sideways makes the robot turn. The advance method works more like the original computer game. When the stick is pushed left, the robot drives to the left of the table. Pushing the stick to the right makes the robot drive to the right of the table. The same holds when pushing the stick forward and back. Although both methods were implemented, the advanced method was eventually chosen as the control of choice and the basic method is no longer used.

In Phase III, the milestone was to have the robots drive around other robots and avoid walls. However, we noticed that this was much more important than we initially thought. Due to the lack of this feature in Phase I, the game could not be considered a working game, because robots would frequently "crash" into each other.

#### 3.1.2 Artificial Intelligence

---

In the original plan the artificial intelligence was spread out over the first two phases of the project. Within the first phase the requirement was that robots should seek for Pacman. In the second phase the robots should avoid robots and walls. This subsystem underwent great changes. The search-for-Pacman function was not implemented until Phase III. Before this, robots drove to random points on the table and this made the game difficult enough.

The second requirement for artificial intelligence was to drive around walls and other barriers. Although planned to be implemented in Phase II, this was much more important than we thought and should have been part of Phase I in order to have a working game.

### 3.1.3 Image Processing

---

The image processing subsystem was the largest subsystem in the planning. Many requirements for image processing had to be met in Phase I. These include reading the camera image, recognising the table, recognising the robots, distinguishing robots from each other and see in which direction a robot is heading. To make the image processing more robust, the block of image processing planned for Phase III was moved to Phase II.

As mentioned in the previous section, at the end of Phase II our test group came by. The way they played the game made us realise that we could make the game much more interactive by not adding walls to the table. In phase IV the requirements for image processing included making it more robust, such as dealing with arms and other items blocking the camera view. This requirement was moved into phase III and became a very important requirement. Visitors of the exhibit, not just the player, can now take part in the game. By moving objects on the table or moving their arms under the camera, they create virtual walls, which are avoided by the robots. This has made the game much more interactive.

### 3.1.4 Game

---

According to the original plan, the game was simply to fulfil assignments and to stay alive as long as possible. This would be realized in the first two phases of the project. When our test group had tested our game at the end of Phase II, it became clear that more thought was needed in assignments. They needed to be more interesting. We decided to make a total of eight levels, including targets with a timer and moving targets in the form of catching a robot. The total game time, from start to finish, would thereby not exceed its requirement of 5 minutes. The high score, which was a feature in Phase IV, was considered to be more important and moved up in the planning. This was also done to make the game more attractive to play.

### 3.1.5 GUI

---

The GUI was a subsystem that spread out over the entire duration of the project. Each addition to the game meant that the GUI needed updating. It is obvious that changes made in the game also caused changes in the GUI. For example, more complex assignments meant changes to the GUI so that it could be displayed correctly. In the end, the only requirement was that both monitors show information that is interesting to the visitors.

## 3.2 Hardware

---

Looking at the artist's impression of the set-up in the introduction, most of the hardware requirements are obvious. A table, computer, monitors, webcam and robots must be present. Further requirements are: the controller is easy to use and to replace, the computer is powerful enough to run the program and the webcam has a resolution of 1920x1080. This high resolution was initially chosen so that each square millimetre of table would be covered by a single pixel. Details about exactly which hardware we

chose and why can be read in the Section 4. As for the robots, the Moway robots were advised. Although there was no particular budget requirement, but it had to be within reason.

Most of the requirements were met. These requirements entailed: low cost and small enough to easily fit on the table. The only concern was that the batteries did not last long enough. The requirement is that the robot can run a full day of eight hours. Details are provided in Section 4.

To create maximum contrast with the robots, we decided that the robots had to be white, while the table should be black. From previous experience we knew that a glare from the table would be disastrous for image processing. Therefore the table was given a matte finish.

### 3.3 Summary

---

The requirements were thought through very carefully. The planning was clear on when each requirement was due and which requirement required other requirements. However, even this clear description could not oversee the challenges we would encounter. Therefore, especially in the last two phases, many requirements were changed and rescheduled to create a fully functional, fun and informative game.

## 4 Project Design

---

In this section the hardware and platform choices are explained. The next section – Section 5 – contains the software architecture built on top of these hardware and platform choices.

### 4.1 Hardware

---

The exhibit will consist of several parts. The basis is a table with a border. Robots drive on top of this table forming a visual and tangible representation of the game. A webcam is mounted above the table to capture the table and the robots on it. The image from the webcam and the signals from a gamepad are processed by a computer. This computer sends signals to the robots to control them.

#### 4.1.1 Table

---

Since the robots are driving around on top of the table, it has to be large enough for the robots to navigate on. Therefore, from our point of view, bigger was better. The arrangement of the room however constrained the size of the table to about 1.6 metres long and 1 metre wide. Therefore these are the final dimensions of the table.

A white border and non-reflective black surface was necessary for a stable identification of the table by the webcam.

#### 4.1.2 Gamepad

---

To control the game, an input device was needed. The original idea of the Science Centre was to use a joystick; however we decided to use a gamepad. Most people are already familiar with a gamepad because they are used to control game consoles. We chose for the Xbox 360 controller. This controller was chosen for two reasons. First of all it has a USB connection, so it is easy to plug into the computer. Secondly, it is a very popular gamepad with many users. That way it is more likely to be well-supported and available in the future.

#### 4.1.3 Robots

---

The requirements state that the robot should be able to be controlled remotely and should be able to drive for minimum of four hours without running out of power. The Moway robots were the initial choice of the Science Centre. There was only one big issue: the robots could only last two hours before the battery would run out. For the exhibit, which has to run for a full day, this is not enough. The three options were finding a better robot, an automatic recharging robot or extending the battery capacity of the Moway robots. The only robots we found that were good enough required assembly, were too expensive, or both. Contacting Moway we found out that the battery capacity could be doubled with very little effort. As for controlling the robots remotely, Moway delivers radio frequency chips, so the ready-to-go Moway robots with extra battery capacity and RF-chips were chosen.

#### 4.1.4 Computer

---

Both processing the images from the webcam and drawing the GUI cost a lot of processing power. However, these tasks are not optimized for multiple processing cores. The best choice would therefore

be a fast dual core processor, rather than a triple or quad core. Intel's Core i3 530 proved to be good value for money. The graphics card needed to have two DVI ports in order to display data on two monitors. The remaining components were chosen with compatibility and low budget in mind.

#### 4.1.5 Webcam

---

The webcam needs to have a high resolution to capture enough details of the robots. The initial requirement was a resolution of 1920x1080. The images that are captured need to be of high quality for better image processing. After all, it is easier to get information from a good picture than from a bad picture with a lot of noise or blurs. The Microsoft Lifecam HD 5000 webcam met our standards, delivering high quality images to work on. Unfortunately the full resolution is not used; a resolution of 640x480 is the maximum resolution that is used. This is due to the software as can be read in the following section.

### 4.2 Software

---

There exists a large number of programming languages today, some more useful than others. When choosing in what language we would write the game, the choice quickly boiled down to two possibilities: C# and Java. These are two very similar languages. We eventually chose Java, because we have more experience with Java than with C#. Apart from that, Java is more widely used and it has been used longer. Therefore, there are more libraries available for Java than for C#.

To process the camera's images, the Java Media Framework is used (JMF). Due to this somewhat outdated software a maximum resolution of 640x480 is possible.

## 5 Architecture

In this section the architecture will be discussed and how it was designed. Designing the architecture has been an iterative process, improving and redesigning subsystems and even the coupling between subsystems. This process has led to a very clear and structured architecture.

### 5.1 Subsystems

The system is divided into six subsystems (Figure 5-1). The system is divided into loosely coupled subsystems, so one subsystem can easily be rebuilt without affecting other subsystems.

- The *Engine* is the central component of the system. It controls the execution of the program which will be explained later in more detail.
- The *Image Processing* subsystem is responsible for capturing webcam images and analysing them to extract the position of the robots on the table.
- The *Xbox Control* subsystem listens to the Xbox controller and fires events when values have changed.
- The *Robot Control* subsystem calculates and sends the robot control signals to the robots.
- The *Graphical User Interface (GUI)* is responsible for displaying information on the screen.
- The *Game* subsystem uses the information provided by the image processing and the Xbox control subsystems to execute its game rules. These game rules calculate commands to send to the robot control and what to present on the GUI.

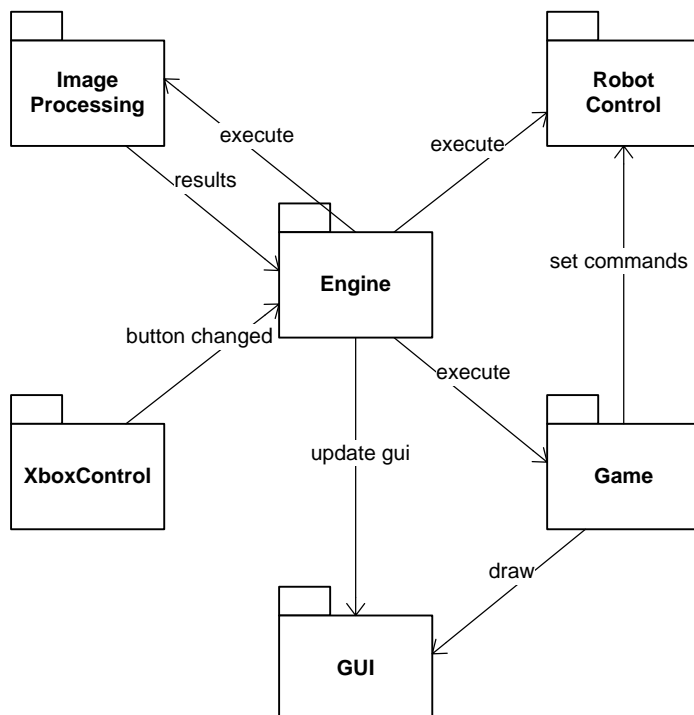


Figure 5-1: The Subsystem Diagram

## 5.2 The Main Control Loop

In the previous section composition of the subsystems is discussed. To explain how it works the main control loop needs to be discussed. The Engine controls the main control flow and controls all other subsystems.

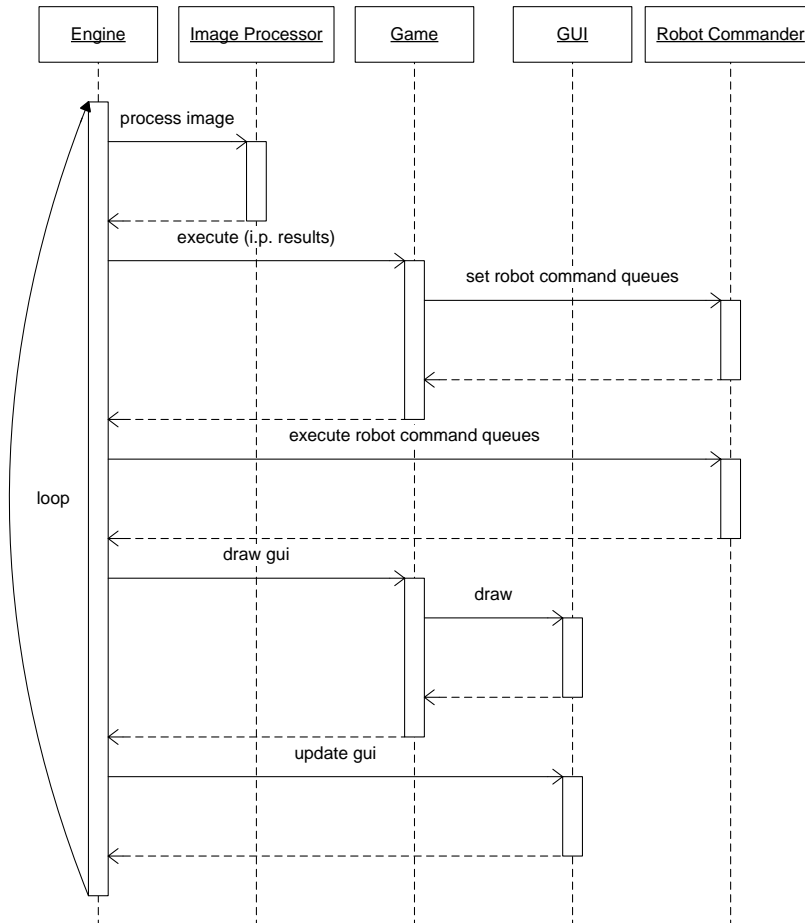


Figure 5-2: The Main Control Loop

In Figure 5-2 the Main Control Loop is shown. First of all, the Engine calls the Image Processor to capture an image and analyse the image. When the Image Processor completed its work successfully, the Game is called with the new information. The Game then uses its game rules to calculate the new game state. These rules dictate whether the level is completed, whether Pacman is still alive and where a certain robot should drive to. The robot commands are then sent to the Robot Commanders, each robot has his own commander. After the Game is done updating and calculating, the Robot Commanders are told to execute their commands. These two steps are separated to ensure that Engine is in control, making the subsystems as loosely coupled as possible. The last part of the Main Control Loop is drawing the GUI.

This is done through game using the double dispatch principle<sup>1</sup>. When drawing the new GUI frame is complete, the Engine signals the GUI to display all the new information.

### 5.3 The Xbox Control Loop

The Main Control Loop does not include the Xbox Controller signals. We have chosen to use polling in a separate loop instead of polling in the main loop. This is good for the response on the control signals. The Main Loop takes about a tenth of a second. Waiting one tenth of a second or longer before there is a response on your input will become annoying when playing a game.

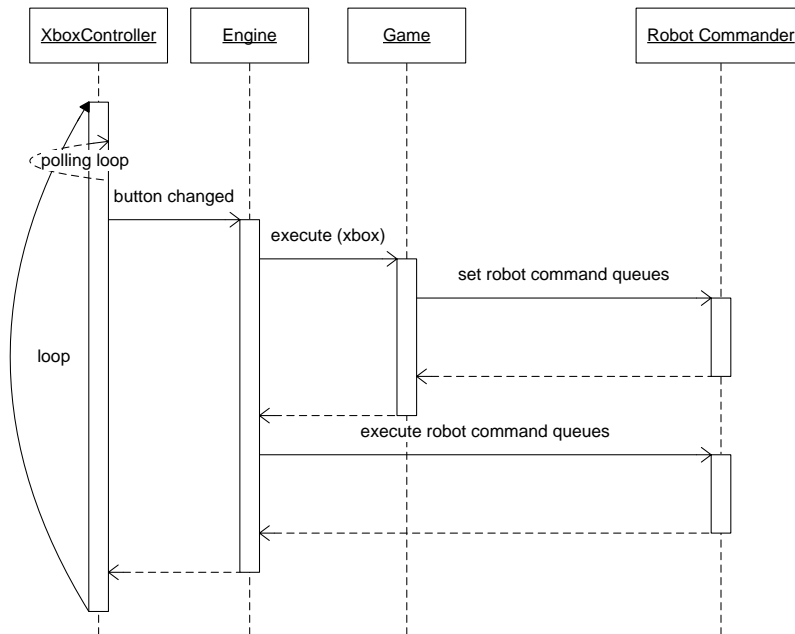


Figure 5-3: The Xbox Control Loop

The control loop driven by the Xbox controller is shorter than the main control loop. We have chosen not to draw the GUI again because that is a costly operation – two full HD windows have to be updated – and the Xbox controller can update up to a hundred times per second. The Loop starts with polling the Xbox Controller for changes. If there is a change, the Engine is notified that something changed. The Engine asks the game to calculate what should happen with the information and then tells the robots to execute.

### 5.4 Control Loop Cooperation

Two control loops have been explained in the two previous subsections. However, both loops use the Engine, the game and the Robot Commanders. Two threads interacting with the same objects typically result in concurrency problems. To solve concurrency, the execute methods in Engine are *mutually excluded*: they cannot be run at the same time. These methods include updating the Engine state and

<sup>1</sup> See [http://en.wikipedia.org/wiki/Double\\_dispatch](http://en.wikipedia.org/wiki/Double_dispatch)



telling the game to execute the game rules on the new information. The image processing and GUI drawing are the operations that cost the most time. These are not in the mutual exclusion because there is no concurrency possible, only the main loop calls them. Because the Xbox Control Loop does not have to wait on these heavy operations, the Xbox Control Loop still stays responsive.

Image processing results are needed in both control loops, but they originate from the Main Control Loop. To be able to use those results in the Xbox control loop – in a different thread than the main control loop – the results have to be saved until new data comes in. This is called *memoization*.

## 5.5 Summary

---

We have learned that a clear design makes the software a whole lot easier to comprehend and implement, on the level of subsystems and on the level of complete architecture. This will stay an iterative process and healthy software will have to be redesigned from time to time.

## 6 Image Processing

---

The biggest subsystem of our program is the image processing. This subsystem is needed for localising and recognising the table and the robots. The whole game depends on the quality of the image processing. This section contains the technical details of the image processor and explains two cases of problems we encountered in more detail.

### 6.1 Technical details

---

The system processes each captured frame step by step to be able to save the result of every step. We chose this structure because it is easy to visualise, which makes the system easier to debug.

The processing starts by fetching an image from the camera in a resolution of 640x480 and scaling it down to 320x240 pixels (Figure 6-1a). This makes the following steps faster for fewer calculations have to be done. Especially the first step, edge detection, requires a large number of calculations.

The image processing starts by finding the table and the objects on the table. The first step is extracting all edges (Figure 6-1b). This is done through a first order difference filter on the intensity of the pixels. This filter measures the difference between pixel intensities. An extra advantage is that because of using the first order difference the absolute intensity of the pixels in the image does not matter, so gradual illumination changes do not affect the edge detection.

To find relevant particles a system flood fills the image to the found edges. To find the correct point to start filling, nine fixed points on the image are checked. The algorithm starts filling at the first point. If none of the other fixed point is filled, the first point was placed on a robot instead of a point on the table. This process is restarted for all other points until it finds a correct point. When a correct fill is executed the difference between the original image and the filled image is calculated resulting in an image with one single object; the surface of the table with holes on the positions of the robots (Figure 6-1c).

With this image a number of points on the edge of the table surface are calculated. Points that differ too much from the other points are filtered out. The remaining points are used to calculate the position and angle of the borders of the table. The calculated lines are drawn on the image so that robots that are near the table border are not a gap on the outside of the table surface but a hole in the table surface (Figure 6-1d). This way they can be recognised as objects on the table.

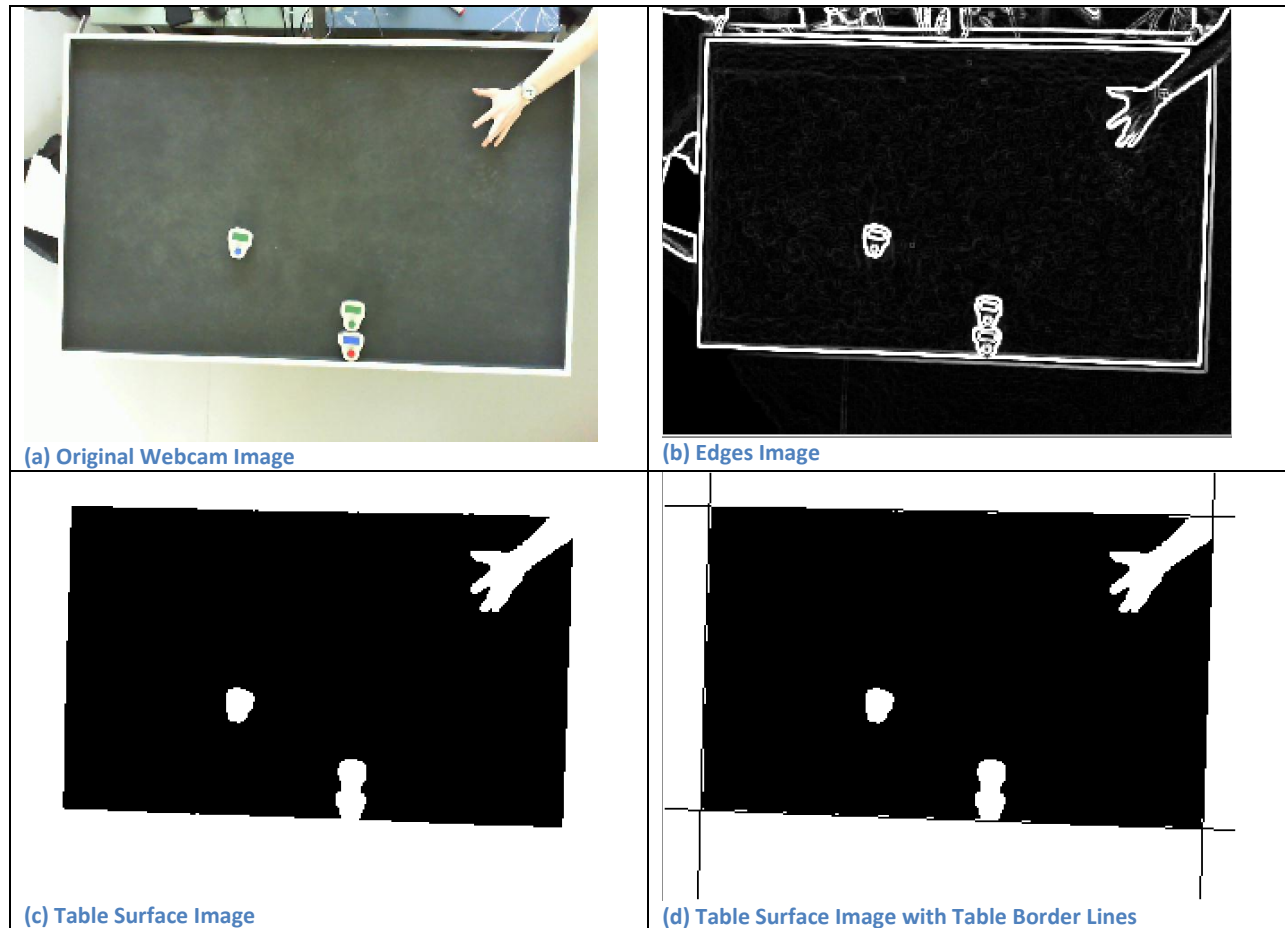


Figure 6-1: Image Processing: Analysing table

The next step is analysing the objects on the table. Of course a hand or an arm of someone positioned above the table will also be an object on the table. This is not a problem for the image processor, and the holes in the table surface created by hands and arms are used in the path finding algorithm to let the robots drive around them.

The objects on the table are copied from the original image (640x480) in high resolution and a difference filter is applied (Figure 6-2a and b). Then the particles (groups of connected pixels) are analysed and filtered on the size and aspect ratio (Figure 6-2c and d).

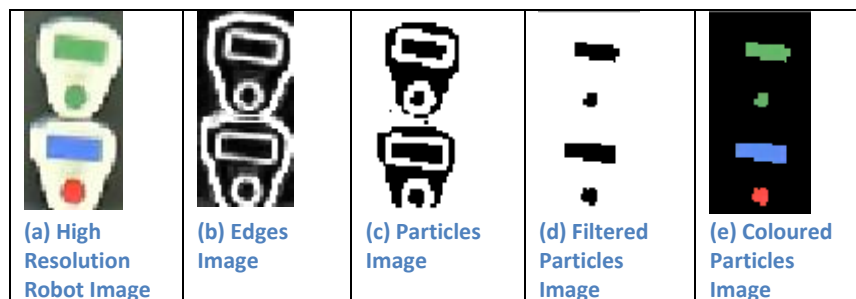


Figure 6-2: Image Processing: extracting particles

Then each particle (Figure 6-3a) is individually analysed for median colour using histograms<sup>2</sup> (Figure 6-3f and g), aspect ratio, position and orientation. The aspect ratio is used to determine if the particle is a rectangle or a circle, aspect ratio below 2.0 is considered a circle and the other particles are rectangles.



Figure 6-3: Image Processing: extracting colours

The positions and orientations are used to find matches for the rectangles with the circles. The circles should be perpendicular to the long axis of the rectangle (Figure 6-4). The matches of rectangles and circles identify the robots. The robot ids, positions and orientations are packaged as robots and this information is passed on to the engine.



Figure 6-4: Circles perpendicular on long axis of Rectangles

## 6.2 Modify the real world

In the previous subsection the inner workings of the image processing subsystem were explained. While creating the image processing subsystem we noticed that a lot of problems were easier to tackle in the real world than solving those using complex algorithms. The main result of this was a more robust system.

An example of our modifications to the real world is to use a black table with thick white borders. The big contrast between table and its borders makes it easier to distinguish them. The table also had to be matt to avoid reflections, which would distort the contrast, of the sun and other light sources.

<sup>2</sup> Each channel (red, green and blue) has its own histogram on pixel intensity. Those are drawn on top of each other in their own colour. For more information on histograms: <http://en.wikipedia.org/wiki/Histogram>.

Another example is using a cover plate with coloured shapes on top of the robots. The original idea was to put the shapes directly on top of the robots. However the image of the robot without a cover plate is very complex and noisy (Figure 6-5a, b and c). To solve this we mounted a flat cover plate with shapes on top of each robot (Figure 6-5c, d and e). Images c and d are clearly less noisy and because of that easier to process. By experimenting with different materials and colours we eventually chose the most suitable ones as cover plate.

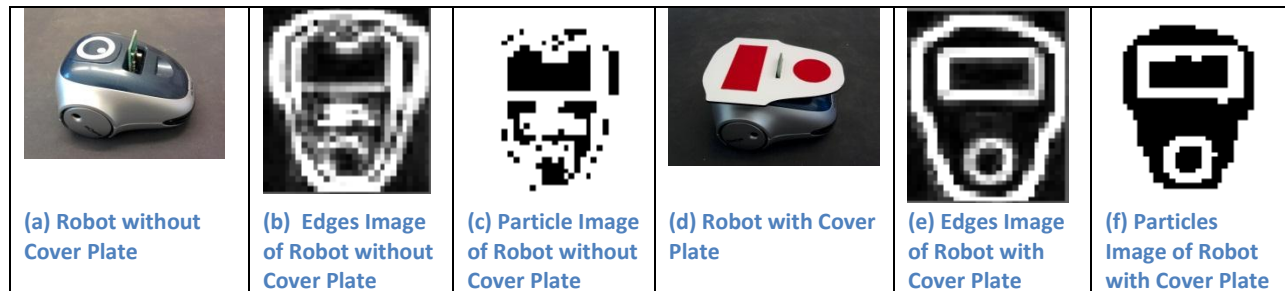


Figure 6-5: Cover Plates

### 6.3 Colours and Shapes

We started by using a fixed threshold on colour (hue value) to distinguish colours from the coloured particles image (Figure 6-2e). However this method was not stable, especially when the light source or the direction of the light changes. Later we looked at histograms of the colours and noticed that we could easily distinguish red, green and blue by simply looking at which of the channels (red, green and blue) had the largest intensity. Once again simplicity was the key to success.

Because it took us some time to distinguish the different colours we tried to find other solutions. It was an option to just use shapes instead a combination of shapes and colours. A larger number of different shapes or more and smaller shapes per robot would be needed. However the combination of low resolution and noisy edges made it impossible distinguish more complex shapes or smaller shapes.

After we found the solution for the colour classification, the robot extraction and identification algorithm was stable. Another option would be to only use colours. However this would require more different colours and the colours would be harder to distinguish. So we used the solution that has only two different shapes, and that has only three different colours.

## 7 Robot control

---

The robots form an important part of the game. How the robots are controlled from the game will be discussed in this section.

### 7.1 Robots

---

The robots are programmed using the Moway GUI that is supplied with the robots. It features a simple control loop that listens for data from the RF chip, decodes it and controls the motors.

### 7.2 Sensors

---

The robot is fitted with a variety of sensors, including four distance sensors. These sensors only have a range of three centimetres but can be very useful, for two reasons:

- *The information from the distance sensors is redundant.*  
The same information can be extracted from the image from the webcam. Redundancy is good for robustness, because unreliability in one source can be compensated with data from another source.
- *The information from the distance sensors is processed faster than the information from the webcam.*  
While it takes at least one tenth of a second to process the webcam information, it takes considerably less time to poll the distance sensors on the robot.

However, using the distance sensors brings a number of complications. We found three ways of using the sensors in the algorithms. However each option has its own complications.

#### 7.2.1 Robots avoid objects by themselves

---

The robots avoid objects by themselves. They use their own sensors and when one of their sensors pick up a signal, they turn 90 degrees to avoid a collision.

The downside of this method is that Pacman will be avoided as well. After all, the robots do not know what is in front of them: a wall, a “monster” robot or the “Pacman” robot. This will make the player invincible, causing the game to become very boring.

Besides that, the independent robot actions can conflict with the commands of the computer. The robot could turn away from an obstacle while the position the robot needs to drive to is on the obstacle (it could be a different robot). Then the robot would turn away, the computer would command it to turn back and the robot would turn away again, etc.

#### 7.2.2 Robots send their sensor data to computer

---

The robots send their sensor data back to the computer, which processes the data and takes it into account for the commands.

This would increase the number of RF messages and would change the RF traffic from one-way traffic to two-way traffic. The complexity of the RF traffic would increase and the total bandwidth in RF space would decrease. The robots need to receive new commands real time. Adding more traffic and more complex traffic would cause in less responsive robot control, resulting in less reactive robots.

The argument that the information from the distance sensors can be processed faster does not hold either. The information has to wait till the RF message can be sent, when the information has reached the program and is sent back it will be as old as the image processing information.

### 7.2.3 Computer anticipates possible robot sensor information

---

The computer anticipates on the robot actions and calculates what the first obstacle would be if the robot distance sensors would measure an obstacle. Based on what obstacle would be the first to encounter the computer tells the robot if it should stop or not when the sensors of the robot measure an obstacle. If the robot senses an obstacle it uses its 'slightly' outdated information on continuing or avoiding the obstacle to either continue or avoid the obstacle.

Basically this is an upgraded version of the independent robot solution (see Section 7.2.1). It negates the problem of driving around Pacman but it does not completely remove it. When a robot is close to more than one robot or close to another robot and the wall the 'slightly' outdated information can still result in wrong actions.

The conflict between commands from the computer and the independent actions from the robots is not solved in this solution. The robot can still turn away, turn back, turn away, etc. (see Section 7.2.1).

### 7.2.4 Conclusion on sensor usage

---

Because there is no solution without negative effects and the architecture gets more complicated using the sensors, we decided to keep the program simple and rely only on the image processing data.

## 7.3 Connection

---

Included with the robots is a radio frequency (RF) dongle that connects using USB. Though we were optimistic at first, it proved to be very hard, if not impossible, to control a USB device directly from Java. One of the main reasons for this is the way Java was designed: a high level language that is independent of the operating system. Because USB control is a very low-level task that is handled differently in each operating system, this is not included in Java. There are some libraries that claim to provide this support, but we did not get one of them to work.

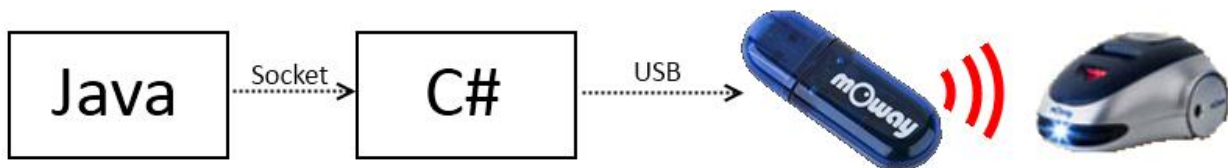


Figure 7-1: Connection between program and robots

We eventually contacted Moway for help with this, and they kindly provided us with a program with which we could send messages to the Moway robots. The downside of this program is that it is written in C#.NET, and not in Java. Communication between Java and the .NET framework is also not supported in Java, but we found a work-around by using network sockets. We modified the C# program to open a network socket and forward all incoming messages from that socket to the robots. The Java program sends all data that is destined for the robots to the opened socket.

## 7.4 Scheduling Algorithm

---

Having decided on the structure of the connection, the next step was to decide how the messages coming in from the network socket should be sent to the robots. Originally we used a queue, so all messages would be sent to the robots in the same order they came in. A problem with this approach is reliability: it is not sure whether or not a message was actually received. To solve this, the program waits for a response and, if necessary, retransmits the message. To avoid starvation, i.e. the problem that some messages never make it to the robots, the number of retries is limited. Despite this there still was starvation: when too many messages are sent to non-responding robots, the messages come in a lot faster than they are sent, causing delays of several seconds between receiving the message from the socket and sending it to the robots.

To be able to control the robots based on the information from the image processing subsystem real-time adjusting of speed is required. Without real-time control the robots would receive orders they needed to receive several seconds ago resulting in robots driving the wrong way for a second or two and robots driving into walls.

The solution was a new algorithm, which was derived from the round-robin scheduling algorithm. For every destination address the last message is stored, for only the last message is relevant. The program then iterates over the destination addresses, sending the last message per destination address. When a message has been sent 10 times, it is assumed that the message is received and it will not be sent again. This algorithm is more efficient than the queue-based algorithm in two points:

1. It does not wait for response; instead it just assumes that a message is received correctly. This does not guarantee that every message is received correctly, but the probability of failure proves to be very low. Moreover, the impact of one lost message is very small because a lot of different messages are sent to the robots.
2. It overwrites old messages with new ones, spending as little time as possible sending obsolete messages.

## 7.5 Commands

---

The commands that are sent to the robot are basic instructions like *'left engine forward, speed 53'*. In the program, higher level commands like *'turn left'* and *'drive to point (x, y)'* are used. These commands use the command pattern. This means that command objects are responsible for controlling the robot.



The advantage of the command pattern is that commands are very easy to queue: simply put them in a queue and call the first item. Besides that it is very easy for other parts of the system to drive the robots: simply enqueue a new command.

## 8 Artificial Intelligence

The main goal of our project was to give the users of the Pacman game an inside look into the methods behind the game and control of the robots. To do this the application shows the calculated paths of how the robots will drive. How these paths are constructed is explained in Subsection 8.1.

The robots are represented by units in the game. The different states and modes those units can have are displayed on the GUI as well. This is done in the form of a state diagram. The implementation of the control of the units is briefly described in Subsection 8.2.

### 8.1 Path finding

There is a number of algorithms that solve the shortest-path problem.<sup>3</sup> These path finding algorithms all need graphs, sometimes simulated by grids. The problem with those solutions is that our robots do not drive on a grid, they drive in the real world using commands like 'drive to point  $(x, y)$ ', as pointed out in Section 7.5. The binary image of the table could be considered a graph with every pixel a node connected to all neighbouring pixels. However this would result in an extremely large graph. Therefore we designed our own path finding algorithm.

Our algorithm is based on an image of the table in black and white. This image indicates that a point is either safe or blocked. A safe point is on the table and available to drive on. A blocked point indicates an obstacle (a robot or an arm, for example). Although the algorithm is not proven correct or optimal, in practice it works well when trying to avoid robots. Avoiding objects like arms is harder, but also possible. In case no path is found, the robot will just drive directly to its target, ignoring any obstacles.

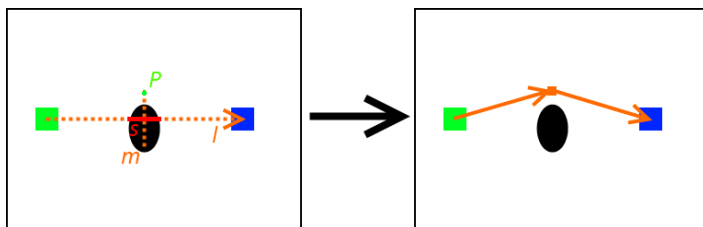


Figure 8-1: The Algorithm Illustrated

The path finding algorithm works as follows:

1. Draw a virtual line  $l$  between the start and the target, and traverse it.
2. If no obstacle is found, return;
3. If an obstacle is found:
  1. Create a new point  $P$ , distant from the obstacle. This is done in a few steps:
    - i. Determine the line segments where  $l$  crosses the obstacle  $s$ .
    - ii. Construct a line  $m$  perpendicular to line  $l$  through the centre of  $s$ .
    - iii. Explore  $m$  with increasing distance from  $l$  on both sides of  $l$  simultaneously.
      1. When a point on  $m$  is found that is safe, take that point as  $P$ .

<sup>3</sup> Source: [http://en.wikipedia.org/wiki/Shortest\\_path\\_problem](http://en.wikipedia.org/wiki/Shortest_path_problem)

2. When no point on  $m$  is safe, there is no path from start to target.
2. Execute (1.) for the pairs  $(start, P)$  and  $(P, target)$ .

This algorithm is optimized in two ways:

1. To avoid stack overflows, the maximum recursion depth is 30 calls. If after 30 calls no path can be found, it is very unlikely that a path can be found with this algorithm.
2. Because the algorithm is sensitive to the first encountered obstacle, the algorithm is run twice: once from start to target and once in reverse order. The shortest path 'wins'.

## 8.2 Unit control

---

The game contains three different AI units to represent the robots in the game: wandering enemy, following enemy and friendly unit. These units have different state diagrams, but they all have a common Search state. This state holds three modes:

- *Wander*, in which case the robot wanders randomly on the table;
- *Patrol* lets the unit patrol between two points;
- *Scan* is a state in which the unit turns from left to right and back to find Pacman.

When the unit is in its Search state, the mode determines how the unit acts. The unit sees the mode as an assignment that can be completed. When it is, it switches to another mode. The choice is based on a random and the unit could keep the same mode.

The wandering enemy only has this Search state. For this unit the GUI does not display the Search state as the state diagram, but it shows the modes the unit can be in. The active mode is highlighted.

The following enemy has two more states:

- *Chase*, in which the robot chases the Pacman robot;
- *Return*, a state in which the robot returns to the position on the table where it left when it started to chase the Pacman.

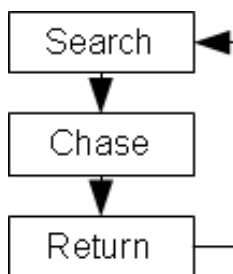


Figure 8-2: The state diagram of the following enemy

This robot leaves the Search state to enter the Chase state whenever it “sees” the Pacman. This means that the Pacman robot entered the view range of the following enemy. The view range is an arc of 99 degrees centred on the base rotation of the robot and has a radius of about a 6<sup>th</sup> of the length of table. While the unit is in the Chase state, the target it has to drive to is updated all of the time. When the

player manages to let the Pacman escape from the view of a unit, that unit will return to the last position on the table it was in when it left the Search state. This is called the Return state. After reaching that point the unit will continue in the Search state with the same mode it had before the chase.

The friendly unit represents a robot that tries to flee from the Pacman whenever it sees the Pacman. Its implementation does not differ much from that of the following enemy except for the Run state instead of the Chase state. The Run state makes the unit go to a random point on the table that is far from the Pacman. When it reaches that point, the unit watches if the Pacman is still in view. If it is, the unit will drive to another random point. Otherwise it will return to the position on the table where it was when it started to run from the Pacman.

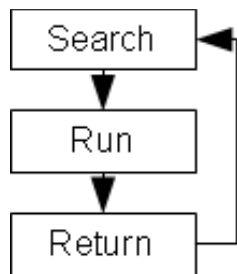


Figure 8-3: The state diagram of the friendly unit

## 9 Results

### 9.1 Game

The goal of the project was to create an exciting and interesting game that simultaneously gives an insight into technology. We believe we have accomplished that goal. We have created a playable game that does not only show a fancy interface, but also includes information about the internal systems such as artificial intelligence and image processing.



Figure 9-1: The left screen of the GUI showing the target, level, score and high scores

Figure 9-1 shows the left screen of the GUI. The assignment for the user is displayed at the top of the screen. On the right a short description of the level and the current score of the player is displayed. The high score list is also displayed on the right, with the current score of the player in it. The player will see his score ascend while he gets a higher score.

On the left screen the target for the player is displayed as a circle with arrows to point where Pacman has to drive to. In the case of Figure 9-1 the player has to drive to the target within a given number of seconds. This is displayed in the message at the top. All robots have an overlay to indicate what type of robot it is. Some robots react on the Pacman if it comes within their view range. The robot on the left is such a robot.



Figure 9-2: The right screen of the GUI showing the AI states, image processing, path finding and the quality of service

The top half of the right screen of the GUI - see Figure 9-2 - shows the states the robots are in. The speed of both wheels is displayed as well, next to the image of the robot. The bottom half of the screen displays the image processing pipeline. It shows how the original image is edited to extract information from it, see Section 5.

The path finding is displayed on the right of the image processing pipeline (the largest table image). The robots path is displayed as red line on the image the system uses to calculate the paths. Underneath the path finding there is a small box with statistics and graphs showing the quality of service. The quality of service is split into a graph displaying how well the borders of the table are detected (top graph) and the quality of recognising the objects on the table (bottom graph).

## 9.2 Discussion

One of the requirements at the start of the project was that the robots had to work together to chase down Pacman. Then the GUI would display what the robots planned to do and how they communicate and cooperate. We did not implement the AI to that level. All robots work independently. The game would get really hard if the robots would work together to get Pacman. Although the robots do have states that depend on the level the player is in and they will react on seeing Pacman. These are displayed on the right screen – see Figure 9-2.

We also decided to drop the idea of using fixed walls. Instead, the user can use his arms and hands to create walls the robots have to drive around. This makes the game more fun, intuitive and interactive. The user can use the right screen to see the effect of his arm on the path finding display because the paths the robots will use will change.

## 10 Conclusion

---

The goal of this project was to create a fun, interactive, Pacman-like game with robots and give an insight into the inner workings. The original plan was to solely display how the robots were controlled by artificial intelligence. Soon it became clear that displaying the robot recognition system, the image processing pipeline, was also a good feature to display on the screen. This gave a good insight look on how the system knows where which robot is on the table and how the system can be fooled by strange objects on the table.

A major issue we encountered during the project was how to properly structure the code to make it both readable and resilient to bugs. At the start of the project we did not have a clear view on how to structure the code and how to make the different subsystems interact with each other. During Phase II, when we already had a draft implementation, we decided to redesign and rewrite the whole engine and game properly. This taught us that designing code does not always come in one step, but rather as an iterative process.

We learned a lot from the test group that tested our game. We noticed that the game does not have to be complex to entertain the player. Instead, making the game more interactive was the key. This made us decide to drop the idea of creating fixed walls and rather invite the visitors to use their arms to create obstacles.

The project did not only cover designing a system and programming, but also team management, time management, seeking help, searching for hardware and ordering hardware. We had to search for good materials, make sure these were delivered on time and seek people for advice on how to improve our image processing and control the robots. This has taught us a lot and has helped us achieve this result. Not only we are happy with the results, also the coordinators from Tinker were quite excited and pleased. The game is interactive, stays within the time constraints and with robots which or not always easy to avoid, especially when in the chase state, great fun. The display contains more useful information than expected when we started the project and we delivered a working product on time.

## Appendices

---

- A. Project Introduction
- B. Original Plan
- C. Tech Tree
- D. Game Architecture
- E. Exhibit Information
- F. User Manual
- G. Maintenance Manual



## Bachelor-project EWI

### SCIENCE CENTRUM TU DELFT



Het Science Centrum Delft is de opvolger van het voormalige Techniekmuseum Delft, en zal begin september 2010 openen. Het science centrum wil dmv actieve opstellingen en projecten laten zien wat er nu gebeurt aan de TU Delft: de TU binnenste buiten! De opstellingen worden ism en door studenten & wetenschappers ontwikkeld en gebouwd. In het nieuwe science centrum van de TU Delft is speciale aandacht voor robotica. Doel van dit project is het uitvoeren van een klassiek Pacman-spel uitvoeren met echte Moway-robots.

#### The MoWay-Pacman game

The goal of this project is to develop an real-life game set-up using small robots. The Moway-robots are provided to you by your client (<http://www.moway-robot.com>). The game will be identical to Pacman, but in a 3D sense. The idea is that the player can steer one of the robots through a maze. Instead of the dots that the Pacman usually collects, the player should try to reach certain positions within the maze. The other robots, which are computer-controlled, try to catch the player by touching it - these robots should exhibit different levels of cooperation. The goal of playing is not winning, but understanding how social corporation between autonomous systems works.



#### The game that needs to be developed should meet the following requirements.

- it should be very cool and attractive to play! (fascinating game & technology)
- user interface is a joystick
- gameplay (details discussed with students)
- visualisation of gameplay (a screen is used to display the 'camera view' of all robots).

To let the audience know what the computer controlled robots are "thinking", there is a separate screen where messages can be displayed like, "Ghost 1 going left", "Ghost 2 target in sight", "Ghost 3 bumps into Ghost 1", "Target intercepted" and so on.

THE CLUE OF THIS GAME IS TO GIVE THE VISITOR AN ELEMENTARY INSIGHT IN CLASSICAL AI: KWOWLEDGE RULES, PRODUCTION RULES AND PERCEPTION.

Students are invited to develop the whole game, including hardware components, knowledge rules and showcontrol (including a logical start/stop routine, lowbattery indication en, if feasible, a high score). It is the deliberate intention of the client to exhibit the result of this work in the science center. So a robust working protocol and dito installation are critical.

**Vragen?** Voor algemene vragen én voor meer information over technische vereisten etc, mail ons!

**Werkplek.** Studenten krijgen tijdens het project een werkplek binnen het Science Centrum. Indien gewenst kunnen ook faciliteiten zoals computers beschikbaar gesteld worden.

**Begeleiding** tijdens het project wordt geleverd door:

Elsa van der Kooi, Science Centrum Delft (algemene begeleiding, [e.m.vanderkooi@tudelft.nl](mailto:e.m.vanderkooi@tudelft.nl))

Stan Boshouwers & Wiebe Berkelaar, Tinker Imagineers (technische begeleiding, bereikbaar via Elsa)

The page features three large, stylized blue circles of varying sizes, each composed of concentric rings of different shades of blue. These circles are arranged in a descending staircase pattern from the top right towards the bottom right. Two thin, light blue lines originate from the top left and extend diagonally across the page, framing the circles.

# Pacman

Bachelor Project TU Delft

**Bram Kol, Rick de Ridder, Daan Wilmer en Daco Harkes**  
**April 02, 2010**

**INHOUDSOPGAVE**

Inhoudsopgave.....	2
Plan Details.....	3
Robots & Artificial Intelligence.....	3
Beeldverwerking .....	5
Game .....	6
Interface .....	7
Fysieke opstelling .....	9
Versies.....	11
Versie 1.0.....	11
Versie 2.0.....	11
Versie 3.0.....	11
Versie 4.0.....	11
Planning.....	12
Benodigdheden .....	13
Tafel.....	13
Robots .....	13
Camera/webcam .....	13
Controller .....	13
Computer .....	13
Schermen .....	14
Belichting.....	14
Budget.....	15
Contactgegevens.....	16
Projectleden .....	16
Contact personen .....	16

## PLAN DETAILS

Dit is het volledige plan voor het Pacman Project bij het Science Centrum van de TUDelft. Het systeem is opgedeeld in vijf subsystemen (Robots & AI, Beeldverwerking, Game, Interface en de fysieke opstelling). Per subsysteem is er in incrementeel plan in milestones en overall zijn de afhankelijkheden van de vorige milestones aangegeven.

### ROBOTS & ARTIFICIAL INTELLIGENCE

Dit subsysteem is verantwoordelijk voor de robots aansturen en de artificial intelligence van de robots.

#### ROBOTS AANSTUREN VANAF PC (1)

Om een spel te kunnen maken waarbij robots aangestuurd worden door een computer moet deze aansturing wel mogelijk zijn. De robots zijn aan de computer verbonden door middel van radio frequency. De eerste uitdaging is om de robots aan het rijden te krijgen en ze aan te kunnen sturen. De robots moeten worden aangestuurd vanuit de Java-applicatie. Er moeten tegelijkertijd meerdere robots aangestuurd worden.

Afhankelijkheden:

- Robots aanwezig

Milestone:

- De robots kunnen nu aangestuurd worden door ertegen te vertellen hoe hard de wielen moeten draaien.

#### ROBOTS AANSTUREN VANAF PC (2)

De tweede laag in het robots aansturen is de robots naar een bepaald coördinaat op het speelveld rijden.

Afhankelijkheden:

- Robots aansturen vanaf PC (1)
- Beeldverwerking (1)

Milestone:

- Robots kunnen naar een bepaalde positie op het bord rijden

#### ROBOTS AANSTUREN VANAF PC (3): PATHFINDING

De laatste laag van aansturing van de robots is dat deze uit zichzelf om muren heen kunnen rijden. In Java wordt er een efficiënte route gezocht vanaf de positie van de robot tot de positie van het doel. Vervolgens wordt de robot via deze route naar het doel geleid met behulp van de webcam.

Afhankelijkheden:

- Robots aansturen vanaf PC (2)
- Beeldverwerking (2)

Milestone:

- Robots kunnen zelfstandig naar een positie op het speelveld rijden zonder daarbij tegen muurtjes of andere robots aan te rijden

---

### PACMAN AANSTUREN (1): ZELF STUREN (BASIC METHOD)

Pacman moet door de speler bestuurd worden. In deze eerste vorm van besturing is het zo dat naar boven betekent dat de robot vooruit of achteruit rijdt en dat als er opzij wordt gedrukt dat de robot draait. Afhankelijk van de gevoeligheid van de controller zijn er meerder gradaties in besturing mogelijk.

Afhankelijkheden:

- Robots aansturen vanaf PC (1)
- Controller aanwezig

Milestone:

- Pacman kan worden aangestuurd door direct de richting vanuit pacman gezien op te gaan (knopje naar boven is naar voren rijden)

---

### PACMAN AANSTUREN (2): RICHTING AANGEVEN (ADVANCED MODE)

De speler geeft instructies die aangeven waar hij wil dat de robot heen wil (noord, oost, zuid en west) en dit wordt bestuurd door het pathfinding gedeelte.

Afhankelijkheden:

- Robots aansturen vanaf PC (2)
- Controller

Milestone:

- Pacman kan worden aangestuurd door de richting van de speler aan te geven, waarna de robot in die richting gaat

---

### PACMAN AANSTUREN (3): EERSTVOLGENDE RICHTING AANGEVEN (CLASSIC MODE)

De speler geeft aan waar hij heen wil. De eerstvolgende mogelijkheid gaat de robot daarheen.

Afhankelijkheden:

- Robots aansturen vanaf PC (3)
- Controller

Milestone:

- Pacman kan worden aangestuurd door de richting van de speler aan te geven, waarna de robot bij de eerstvolgende mogelijkheid daarheen gaat.

---

### ARTIFICIAL INTELLIGENCE (1)

Robots lopen richting de speler met een willekeurige fout.

Afhankelijkheden:

- Beeldverwerking (1)
- Robots aansturen (2)

Milestone:

- Robots lopen constant ongeveer in de richting van de speler

---

## ARTIFICIAL INTELLIGENCE (2)

Een simpele AI met verschillende states. De AI zal niet veel meer zijn dan de speler zoeken en achtervolgen om proberen aan te tikken. Het lopen naar de speler kan met willekeurige waardes worden aangepast, zodat de speler nog wel speelruimte over houdt.

Afhankelijkheden:

- Beeldverwerking (2)
- Robots aansturen (2)

Milestone:

- Robots bevatten een AI met verschillende states
- De robots kunnen muurtjes en andere obstakels ontwijken

## BEELDVERWERKING

Dit subsysteem is verantwoordelijk voor de beeldverwerking. Het zal in steeds grotere mate robuust worden en met grote precisie de locatie en oriëntatie van alle spelobjecten kunnen vinden.

---

### BEELDVERWERKING (1): ROBOT-IDENTIFICATIE

Het systeem moet robots onderscheiden van de tafel en ze afzonderlijk identificeren. Ook de richting van de robots kan worden bepaald. Dit wordt gedaan door middel van hoog contrast tussen robots en tafel, en een uniek, niet-roteerbaar uiterlijk per robot.

Afhankelijkheden:

- Tafel met webcam

Milestones:

- De beelden van de camera kunnen worden ingelezen in de Java-applicatie
- De robots kunnen onderscheiden worden van de tafel
- De robots kunnen onderling onderscheiden worden
- De richting van de robots kan bepaald worden
- De randen van de tafel kunnen herkend worden

---

### BEELDVERWERKING (2): MUREN-IDENTIFICATIE

Als uitbreiding op de herkenning van robots, moet het systeem muren kunnen onderscheiden.

Afhankelijkheden:

- Beeldverwerking (1)

Milestone:

- Muren kunnen onderscheiden worden van de rest

---

### BEELDVERWERKING (3): ARMEN-IDENTIFICATIE

Een mogelijke laatste uitbreiding is het onderscheiden van andere objecten dan muren en robots. Hieronder vallen armen en handen. Dit is om het systeem nog robuuster te maken. Het systeem moet in ieder geval om kunnen gaan met een arm die in het beeld komt, de arm hoeft niet per se herkend te worden.

Afhankelijkheden:

- Beeldverwerking (1)

Milestones:

- Het systeem kan omgaan met een arm in het beeld
- Het systeem kan armen in het beeld herkennen

## GAME

Dit subsysteem bevat alle spellogica. Het spel zal incrementeel opgebouwd en verbeterd worden.

---

### GAME (1): IN LEVEN BLIJVEN

De basis van het spel bestaat uit een engine die de states bijhoudt en de interactie regelt bij het in de buurt komen van andere robots. Bij een reset of game over krijgen de robots opdrachten om naar de begin punt te rijden.

Afhankelijkheden:

- Beeldherkenning (1)
- Robots aansturen (1)

Milestones:

- Engine met states als: playing, game over, returning to start position (reset)
- Als monster te dicht in de buurt van pacman is, is het spel over

---

### GAME (2): OPDRACHTEN

Het spel geeft de speler telkens opdrachten om hem bezig te houden. Dit kan bijvoorbeeld het rijden naar een punt op het speelveld zijn. Deze opdrachten worden door de GUI weergegeven met een icoontje in het speelveld.

Afhankelijkheden:

- GUI (1)

Milestones:

Er worden opdrachten aangemaakt en weergegeven in de GUI

---

### GAME (3): SCORE

Om de speler een idee te geven hoeveel opdrachten hij heeft gehaald, wordt er een score bijgehouden en weergegeven.

Afhankelijkheden:

- Game (2)
- GUI (1)

Milestones:

- Er wordt een score bijgehouden

---

### GAME (4): HIGHSCORE

Om spelers elkaars score te laten vergelijken wordt er een highscorelijst bijgehouden.

Afhankelijkheden:

- Game (3)
- GUI (2)

Milestones:

- Er wordt een highscorelijst bijgehouden
- De speler kan zijn naam opgeven via de controller

---

### CHEATS / EASTER EGGS

Om het debuggen gemakkelijker te maken en om de speler extra functies te geven, voegen we ook een aantal cheats en easter eggs toe. Een voorbeeld hiervan is met de X-toets kan je je pacman weer tot leven wekken of je stopt alle robots behalve de pacman met de Y-toets. Deze cheats zullen voor de uiteindelijke versie uit het spel gehaald worden, of in ieder geval ververstopt. Easter eggs die de gameplay niet makkelijker maken kunnen er echter in blijven.

Afhankelijkheden:

- Pacman aansturen (1)

Milestones:

- Knoppen op de controller linken naar speciale acties bij de robots en de spelstatus

---

### INTERFACE

Het Interface subsysteem is verantwoordelijk voor het weergeven van informatie aan de gebruiker door beeld en geluid.



---

### GUI (1): SPEELVELD OOK OP SCHERM WEERGEVEN

In onze applicatie gaan we uit van twee beeldschermen om de bezoekers te laten zien hoe het spel verloopt. Op het linker beeldscherm wordt het speelveld weergegeven met daarop de opdracht voor de speler aangegeven. Ook zie je icoon om aan te geven waar de robots zich bevinden. Deze stap is belangrijk voor het debuggen van de applicatie.

Afhankelijkheden:

- Beeldverwerking (1)

Milestones:

- Het speelveld met bijbehorende spelers wordt weergegeven
- De richting van de robots worden in het speelveld weergegeven

---

### GUI (2): CAMERABEELD EN ROBOT STATES

In de tweede versie van de GUI zal de kaart op het linker beeldscherm het beeld van de camera weergeven. Op het rechter beeldscherm wordt bovenaan groot de naam van het spel weergegeven. Het onderste gedeelte heeft een state diagram weer waarop gevisualiseerd wordt hoe de AI van de monsters in elkaar zit en in welke state de monsters zich op dat moment bevinden.

Afhankelijkheden:

- GUI (1)
- Artificial Intelligence (2)

Milestones:

- Het camerabeeld wordt weergegeven achter de kaart
- De states van de robots worden fancy weergegeven

---

### GUI (3): ACTIES EN PLANNEN VAN ROBOT OP SPEELVELD WEERGEVEN

De laatste versie van de GUI geeft in het speelveld ook aan hoe de robots denken te gaan rijden waarbij verschillende kleuren worden gebruikt om aan te geven wat voor actie het is. Bijvoorbeeld rood voor aanvallen. Op het rechter beeldscherm, bovenste gedeelte, staan behalve de naam van het spel meldingen zoals 'gewonnen', 'dood', 'verwijder item uit beeld'. Aan het einde van het spel kan de speler zijn naam opgeven met de controller.

Afhankelijkheden:

- GUI (2)
- Artificial Intelligence (2)

Milestones:

- De acties van de robots worden weergegeven in het speelveld
- Spelmeldingen worden weergegeven

---

## GELUID (1)

Een mogelijke toevoeging om het spel interessanter te maken is geluid. Dit zal vooral als extra feedback naar de speler dienen. Dit zou in eerste instantie het regelen van het afspelen van geluid via de audio-output van de computer en het aansturen van de robots om geluid af te spelen via hun interne speaker. Hierbij moet opgepast worden dat het geen druk gekwetter wordt, omdat nog andere bezoekers zijn.

Afhankelijkheden:

- Hardware aanwezig
- Robots aansturen vanaf pc

Milestones:

- Geluid kan afgespeeld worden via de robotjes
- Geluid kan afgespeeld worden via speakers in de tafel

---

## GELUID (2)

Zodra het mogelijk is geluid af te spelen, kunnen we het systeem uitbreiden om muziek af te spelen die afhankelijk is van het spelverloop. Daarnaast kunnen ook de robots apart ook geluid maken afhankelijk van hun state.

Afhankelijkheden:

- Geluid (1)

Milestones:

- Er wordt muziek afgespeeld
- De muziek hangt af van de state van het spel
- De geluiden van de robots hangen af van hun state

## FYSIEKE OPSTELLING

Het laatste subsysteem is de fysieke opstelling. Dit subsysteem wordt door het Science Center gerealiseerd.

---

## TAFEL MET WEBCAM EN VERLICHTING

Een matzwarte tafel met opstaande witte randen waar aan drie kanten het spel goed te overzien is en waterpas staat. De randen en muurtjes zouden van doorzichtig materiaal gemaakt kunnen worden, maar moeten nog wel een duidelijke witte bovenkant. Deze moet minimaal een centimeter breed zijn om goed waar te kunnen nemen met de camera.

De camera moet op een hoogte van ongeveer 1 meter midden boven de tafel hangen. Deze kan aan de tafel worden bevestigd als deze ook trillings-vrij is.

Er moet een afsluitbare kast onder de tafel staan voor de computer.

De tafel moet voldoende verlicht zijn. Mocht de standaardverlichting niet voldoende zijn, dan moeten lampen boven de tafel komen te hangen.

Milestones:

- De tafel staat
- De camera hangt boven de tafel

---

## ROBOTS

Zes robots als basis voor het spel moeten op de tafel rond rijden. Deze hebben op de bovenkant een plaatje zitten dat goed zichtbaar is voor de webcam. Deze plaatjes zorgen ervoor dat de beeldverwerking het verschil kan bepalen tussen de verschillende robots en de oriëntatie van de robots kan bepalen.

Milestone:

- De robots zijn aanwezig

---

## COMPUTER EN TWEE SCHERMEN

Er is een computer met 2 externe schermen. De computer zal plaats vinden onder het bord in de afgesloten kast. Hier zal ook de keyboard en muis plaats vinden die voor het gebruik van het spel niet nodig zijn. De beeldschermen zullen aan een muur worden bevestigd die aan een kant van de tafel staat. Mocht de opstelling niet naast een bestaande muur staan, dan moet de beeldschermen aan nog te bouwen muur worden gehangen op oog hoogte.

Milestone:

- De computer is aanwezig en aangesloten met beeldschermen

---

## CONTROLLER

De controller voor aansturen van de pacman zal aan het voorkant van de tafel liggen (tegenover de beeldschermen).

Milestone:

- Controller aanwezig en aangesloten voor spel bediening

---

## MUURTJES OP DE TAFEL

Als er muurtjes op het speelveld geplaatst worden, moeten ze bijna even hoog als de robotjes worden (ongeveer 3 cm hoog en 1 of 2 cm breed) en bovenop wit en aan de zijkant matzwart geverfd zodat het goed afsteekt van de rest van het speelveld.

De muurtjes moeten niet gelijk vastgelijmd worden, eerst zal er getest moeten worden met beeldverwerking.

Milestone:

- Tafel met muurtjes gereed

## VERSIES

De software is naast subsystem ook onderverdeeld in versies op basis van prioriteit en afhankelijkheid van bepaalde subsystemen.

### VERSIE 1.0

Robots aansturing PC 1 & 2

- Pacman aansturing 1 & 2
- AI 1
- Beeldverwerking 1
- Game 1
- GUI 1
- Fysieke opstelling helemaal zonder muurtjes

### VERSIE 2.0

- AI 2
- GUI 2
- Game 2 & 3

### VERSIE 3.0

- Fysieke muren
- Beeldverwerking 2 (Muren)
- Robots aansturing 3 (Pathfinding)
- Pacman aansturing 3
- GUI 3

### VERSIE 4.0

- Game 4 (Highscore)
- Beeldverwerking 3 (Armen)
- Geluid 1 & 2

**PLANNING**

De planning is op basis van de versies van de software.

Week	Werkzaamheden
16	<ul style="list-style-type: none"><li>• Onderzoek naar softwarepakketten.</li><li>• Experimenteren met hardware.</li><li>• Versie 1</li></ul>
17	<ul style="list-style-type: none"><li>• Versie 1</li></ul>
18	<ul style="list-style-type: none"><li>• Versie 1</li></ul>
19	<ul style="list-style-type: none"><li>• Versie 1</li></ul>
20	<ul style="list-style-type: none"><li>• Versie 2</li></ul>
21	<ul style="list-style-type: none"><li>• Versie 2</li></ul>
22	<ul style="list-style-type: none"><li>• Versie 3</li></ul>
23	<ul style="list-style-type: none"><li>• Versie 3</li></ul>
24	<ul style="list-style-type: none"><li>• Uitloop</li><li>• (Versie 4)</li></ul>
25	<ul style="list-style-type: none"><li>• Uitloop</li><li>• Eindverslag af</li><li>• Presentatie</li></ul>

## BENODIGDHEDEN

De spullen die nodig zijn om het project te kunnen realiseren.

### TAFEL

Het speelveld is een tafel van 160 x 160 cm die van boven mat zwart geleverd is. In de eerste versie nog zonder een vast doolhof, wel met een witte opstaande rand rondom. Onder de tafel zouden we graag de computer kwijt willen, maar afgeschermd van de spelers.

In latere versie komen er misschien witte muurtjes op de tafel bij om zo een doolhof te vormen.

### ROBOTS

moway 6 stuks

De beplakking hiervan is belangrijk voor beeldverwerking: kleuren/stippen/codes voor identificatie

3x deluxe kit van 2 robots, 2 rf robot modules en 1 rf usbstick voor in computer.

[http://www.minirobots.es/index.php?option=com\\_content&task=view&id=141&Itemid=190](http://www.minirobots.es/index.php?option=com_content&task=view&id=141&Itemid=190)

3\*300 euro excl btw.

### CAMERA/WEBCAM

Moet boven de tafel gehangen worden en aangesloten aan de pc. De professor van beeldverwerking op de TU geeft aan dat een webcam hoogst waarschijnlijk voldoet, er wordt ook bij practica gewoon een webcam gebruikt.

Microsoft lifecam HD5000, 720p resolutie, 40 euro

<http://tweakers.net/pricewatch/254791/microsoft-lifecam-hd-5000.html#tab:prices>

### CONTROLLER

Een die heel blijft met kinderen.

In eerste instantie gaan we uit van een enkele joystick. We houden echter nog wel een multiplayerfunctie in gedachten waar dan een extra joystick voor nodig zou zijn.

Na overleg hebben we gekozen voor een xbox360 controller. Deze wordt al veel gebruikt dus is waarschijnlijk beter ondersteund in de software. Het is een universele controller die waarschijnlijk nog lang in omloop blijft, dus hij is vervangbaar mocht deze stuk gaan.

<http://tweakers.net/pricewatch/128964/microsoft-xbox-360-controller-for-windows.html>

30/35 euro

Wellicht kan er nog een houder bijkomen, om het snoer te beschermen en ontlasten. Het is niet bekend waar zoiets verkrijgbaar is of tegen welke prijs.

### COMPUTER

Een computer die 2 schermen met hoge resolutie kan aansturen (geen onboard videokaart!) + toetsenbord en muis. De pc moet krachtig genoeg zijn om beeldverwerking te kunnen doen en 2 grote schermen aan te sturen.

+ - 500 euro

videokaart: 2x DVI aansluiting

Bijvoorbeeld de 500e versie van: (maar wel met 2xDVI kaart)

<http://tweakers.net/reviews/1583/2/tweakers-punt-net-best-buy-guide-editie-maart-2010-budget-en-basissysteem.html>

## SCHERMEN

2 grote schermen

23 inch, full HD, dvi (scherm) 2x175 euro

<http://tweakers.net/pricewatch/236702/samsung-syncmaster-p2370.html>

32 inch, full HD, vga/hdmi (tv) 2x400 euro

<http://tweakers.net/pricewatch/247861/akai-al3280fbk.html>

## BELICHTING

De verlichting moet vrij uniform zijn over de tafel (in ieder geval vanuit het oogpunt van de camera), en de hele tafel moet goed verlicht zijn.

**BUDGET**

Een overzicht van de kosten van de benodigheden.

Onderdeel	Aantal	Prijs ex. btw per stuk	Totaal	Opmerkingen
<b>Beeldscherm 23"</b>	2	145	290	Er kan gekozen worden voor twee LCD tv's 330,- euro
<b>Computer</b>	1	500	500	Prijs is een schatting, ligt aan configuratie
<b>Webcam</b>	1	32	32	Full HD
<b>Robot deluxe kit</b>	3	300	900	Moway robots
<b>Ganepad</b>	1	30	30	Prijs is een schatting
<b>Materieel tafel</b>	1		?	
		<b>Totaal:</b>	<b>1752</b>	



**CONTACTGEGEVENS****PROJECTLEDEN**

Rick de Ridder

Studienummer: 1308254

Email: [acidkurck@gmail.com](mailto:acidkurck@gmail.com)

Tel.: 06 10752810

Bram Kol

Studienummer: 1192701

Email: [bram.kol@planet.nl](mailto:bram.kol@planet.nl)

Tel.: 06 28180843

Daan Wilmer

Studienummer: 1358510

Email: [daan.wilmer@gmail.com](mailto:daan.wilmer@gmail.com)

Tel.: 06 4489 6059

Daco Harkes

Studienummer: 1361902

Email: [dc.harkes@gmail.com](mailto:dc.harkes@gmail.com)

Tel.: 06 12 100 358

**CONTACT PERSONEN**

Drs. Elsa M. van der Kooi

(Aanspreek punt praktische zaken in science center)

Email: [e.m.vanderkooi@tudelft.nl](mailto:e.m.vanderkooi@tudelft.nl)

Tel.: +31 (0)15 27 83937

Prof. dr. K.G. Langendoen

(Begeleider project vanuit EWI)

Email: [K.G.Langendoen@tudelft.nl](mailto:K.G.Langendoen@tudelft.nl)

Tel.: +31 (0)15 27 87666

Roel Bolhuis

(Projectleider van Tinker imagineers BV.)

Email: [roel@tinker.nl](mailto:roel@tinker.nl)

Tel.: +31 (0)30 230 04 05

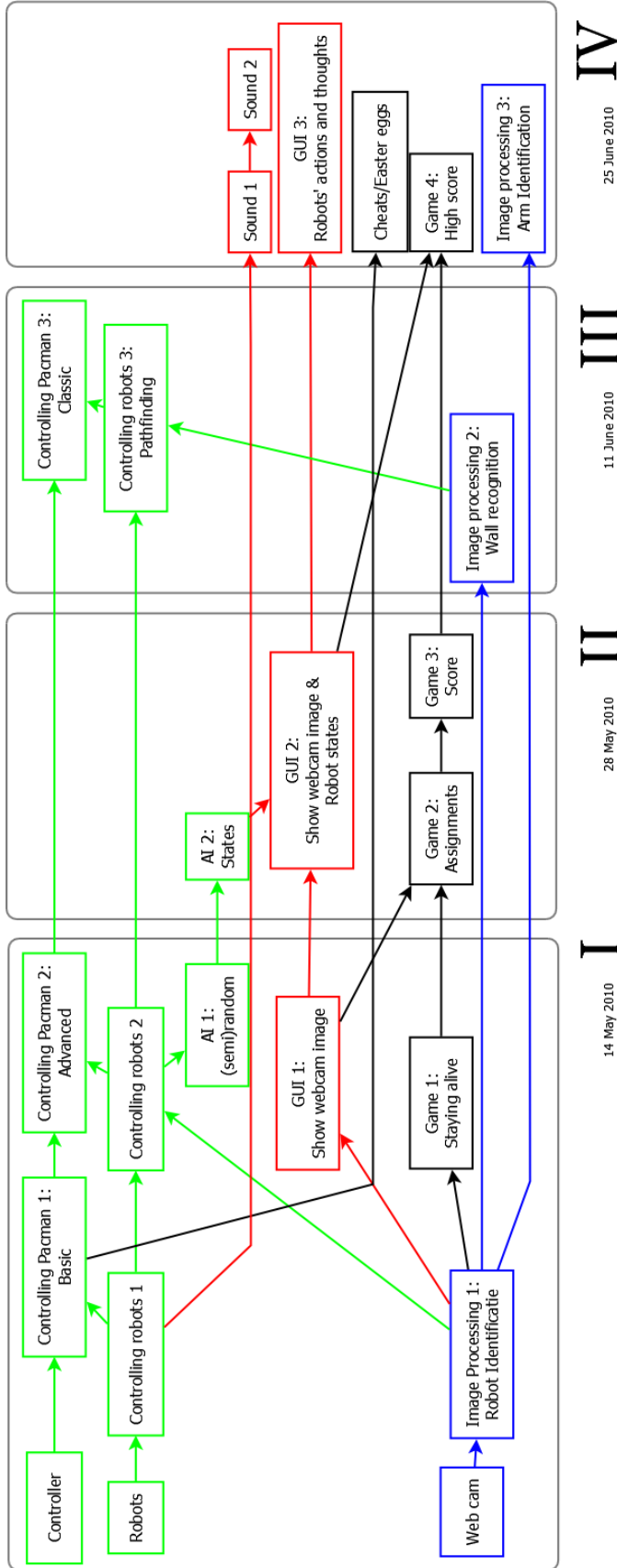
Mob.: +31 (0)6 16 11 87 19

Stan Boshouwers

(Oprichter en directeur van Tinker imagineers BV.)

Email: [stan@tinker.nl](mailto:stan@tinker.nl)

# Appendix C - Tech Tree



IV

III

II

I

# Game Architecture

---

All subsystems except the Game subsystem are thoroughly explained in the report. Since the Game subsystem was not explained in the report the architecture will be explained in this document.

## 1 Game

---

Game manages the levels and the map of robots and highly depends on updates from the Engine. The Engine controls the Game and the Game may only update when the Engine calls it.

On the start of Game it creates a queue of levels, starting by a Reset level (Subsection 2.2), followed by two StaticTarget levels (2.3), a JumpyTarget level (2.4), and a FriendlyTarget level (2.6). The list continues with another Reset level to make sure that the Pacman will not collide with an enemy robot which was a friendly robot in the previous level. The second Reset level is followed by three TimedTarget levels (2.5) and an Endurance level (Subsection 2.7).

After the initialisation of the levels the Game starts executing the first level in the queue and switches to the next level once the level has been completed. If the completed level was a Reset level, the Game notifies the Engine that the Game should be paused. The Engine is also notified when the player has failed in completing a level and the Engine should ask for a name to enter the score in the highscores list.

## 2 Levels

---

The Game was build up with five different levels the player should complete to obtain a score. In each level the player has to complete one assignment. This can be reaching a point on the table, sometimes within a time limit, or chasing and reaching another robot that tries to flee from the Pacman. The levels have been built to be as stateless as possible to avoid programming errors.

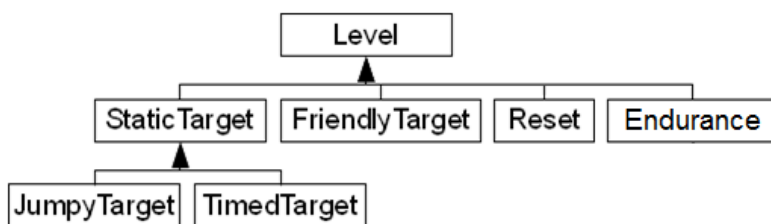


Figure 1: The extension tree of the levels

### 2.1 Level

---

The Level class is an abstract representation of a level. It contains a basic score for the level, functions to update the map of robots and units and it contains the execute function that calls basic level parts. The update of the unit map consists of assigning specific units to the given robots when the unit map does not hold a unit for the robot. The implementations of this abstract class are StaticTarget, FriendlyTarget, Reset and Endurance, as shown in Figure 1.

### 2.2 Reset

---

The Game starts by resetting the robots. All robots on the table are guided to a fixed location on the table. This Level does not give the player any possibility to control robots via the Xbox controller. When all robots have reached their target on the table, this Level is completed.

### 2.3 StaticTarget

---

The Game continues with two StaticTargets with increasing rewards. The StaticTarget level contains a single target, randomly placed on the table, but far from the Pacman. The player should guide the Pacman towards the target without getting killed by the other robots. This Level has no time limit and will stay in place until it has been reached.

This Level requires at least one robot on the table. The first robot will be Pacman. Other robots will become wandering enemies.

### 2.4 JumpyTarget

---

This target inherits all functions from StaticTarget but it has a different implementation of the setUnitCommands-method. This method is used to check whether the current target is within the table and to replace it if it is not within the boundaries. The method is also used for setting new commands in general for all units before the units receive an execute-call.

The JumpyTarget not only replaces the target when it is out of the bounds of the table, but also whenever an enemy robot reaches it. This means that the player of the Game should reach the target before another robot does, but the robots will not drive to that target on purpose.

The Game contains just one JumpyTarget and comes right after the two StaticTargets. It has a higher reward than both of them.

### 2.5 TimedTarget

---

Just like the JumpyTarget the TimedTarget extends the StaticTarget. The Game contains three of these targets in which the player has to reach the target within the given time limits (30, 23 and 15 seconds). The reward depends on the time left when reaching the target. When the player does not reach the target within the given time limit, a new random target is created, but the player may fail only one time. To make this level even more challenging, one of the enemy robots is a following enemy.

### 2.6 FriendlyTarget

---

The last level in the Game is tagging a special robot on the field. One robot is assigned as the Pacman, one as the friendly robot and all others as following enemies. The friendly robot will try to run from the Pacman whenever it "sees" it, while the enemies try to chase Pacman whenever they "see" it. The player should try to reach the friendly robots before the enemies reach Pacman. When the player reaches it, the Game is over.

### 2.7 Endurance

---

The last level in the Game is the Endurance level forces the user to run and hide from the enemy robots. All robots on the table, except for the Pacman, are following enemies. This makes it very difficult for the player to drive on the table with Pacman without getting killed (touched by the enemy robots). For every ten milliseconds the user gets a point. The Endurance level cannot be completed. The Pacman can only die.

### 3 Units

---

Each Level contains a map of units. The Level determines which units are contained in that map. The units in the map do not have to be of the same class. Every unit has an execute-method so that the Level does not have to pay attention to what kind of unit it is.

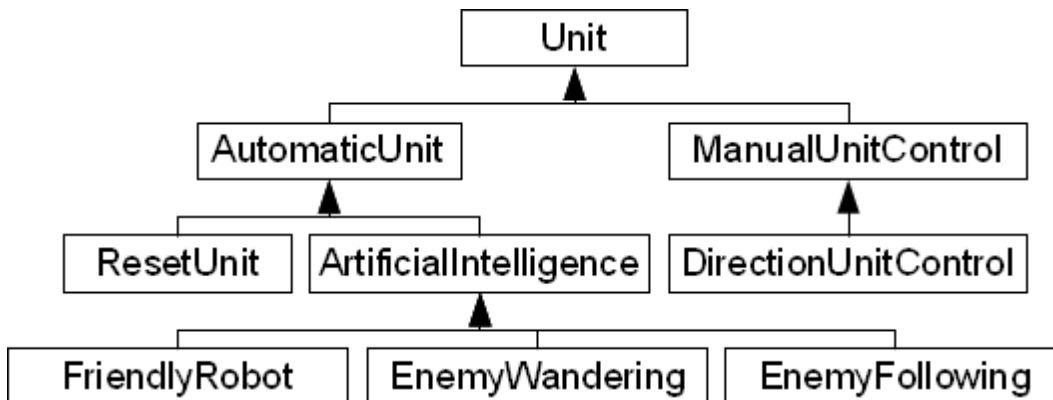


Figure 2: The unit extension tree

#### 3.1 Unit

---

The abstract class unit, see Figure 2, contains the declaration of the execute-method which saves the position and rotation of the linked robot. These values are used by the draw-methods of subclasses to draw the robots on the GUI, but are kept private so subclasses can only retrieve them via getters and cannot overwrite them.

#### 3.2 AutomaticUnit

---

This abstract class extends the Unit class and contains extra methods for checking whether the unit has completed its current assignment. It also holds the modifiers for the list of assignments the unit has.

#### 3.3 ResetUnit

---

The ResetUnit is an automatic unit that just drives to a target on the table given by the Reset level. It does not modify the list of assignment, which just holds a single ReturnAssignment.

#### 3.4 ArtificialIntelligence

---

The abstract ArtificialIntelligence class defines methods for getting new random search assignment (scanning, patrolling or wandering) and checking whether the Pacman is in view of the AI unit.

#### 3.5 FriendlyRobot

---

The friendly robot runs from the Pacman whenever it spots it comes in the view of the friendly robot. When the friendly robot does not see the Pacman, it will execute a search assignment. If the search assignment has been completed, a random search assignment is generated and executed. It also calls the GUI for drawing the unit onto the screen with its view range displayed as well.

#### 3.6 EnemyWandering

---

The wandering enemy is nothing more than switcher between search assignments. It has the same functionality as the FriendlyRobot but does not flee from the Pacman.

### 3.7 EnemyFollowing

---

The following enemy does the same switching between search assignments as the friendly robot and the wandering enemy. The only difference is that the following enemy chases the Pacman whenever it comes within the view of the unit.

### 3.8 ManualUnitControl

---

The abstract ManualUnitControl class enables the levels to have a robot controlled by the Xbox controller. It also supplies a check for the death of Pacman. The original plan stated an option for having three different ways of controlling the Pacman (freely, direction, direction and walls), but only the direction control was left.

### 3.9 DirectionUnitControl

---

DirectionUnitControl implements the ManualUnitControl by using Xbox controller input to move the unit in just four directions on the table: north, east, south and west. The actual code for achieving this is implemented in the robot control layer.

## 4 Assignments

---

Every AutomaticUnit contains a queue with assignments. These assignments order the corresponding RobotCommander to let the robot drive to a point or change the rotation of the robot. It checks for the completion of itself.

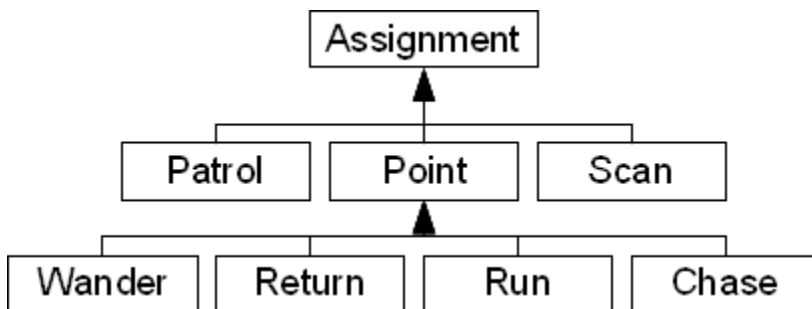


Figure 3: The assignment extension tree

### 4.1 Assignment

---

The main class in the assignment extension tree is Assignment, as shown in Figure 3. This abstract class defines an execute function that should be implemented by all subclasses. It also keeps a boolean that tells the system whether the assignment has been completed or not.

## 4.2 Patrol

---

The PatrolAssignment makes the unit drive from one point to another and back. The assignment is initiated with two random points on the table and a random number of times it has to visit a point. This is depicted in Figure 4. The number of visits is at least four and never more than eight. The assignment keeps track of the number of visits it has left. The assignment has been completed when the visits left counter reached zero.

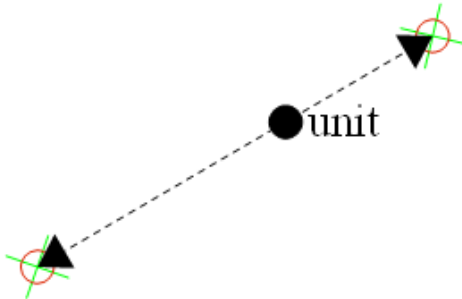


Figure 4: The unit patrols between two points

## 4.3 Scan

---

The ScanAssignment was created to let the robot scan its surrounding area for the Pacman. The assignment starts by setting a random target rotation. A second target rotation is set 90 degrees from the first rotation, see Figure 5. Then a random number of swings is set. When the assignment is called to execute, it rotates to the left target rotation. When the rotation is reached, it swings back to the other rotation. The assignment is completed when it has reached the set number of swings.

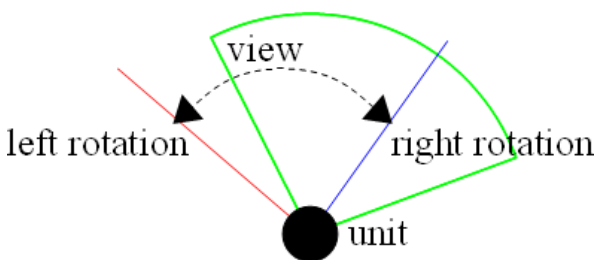


Figure 5: The unit turns from left to right and back

## 4.4 Point

---

The abstract class PointAssignment implements the execute function for point assignments. It makes the robot drive to a predefined point on the board and sets the completed boolean to true when it has reached that target. When the target was not set or when the parent robot was lost, the assignment is set as completed as well. When this happens, the unit holding the assignment can create a new assignment.

- The PointAssignment has four different implementations (see Figure 3):
- WanderAssignment creates a random point on the table to drive to;
- ReturnAssignment needs to get a point from the unit holding it;
- RunAssignment creates a random point far from Pacman;
- ChaseAssignment creates a point on Pacman and this assignment is replaced every frame by the FollowingEnemy that holds this assignment.

## Pacman Project

### Wat zie je hier en hoe werkt het?

De Pacman spel is een klassiek computerspel dat iedereen wel kent. Studenten aan de TU Delft kregen op de opdracht om dit spel met echte robots naar eigen inzicht te reproduceren.

Deze opstelling bestaat uit twee beeldschermen en een tafel waarop de robots staan. Boven de tafel hangt een camera. Het beeld van de camera wordt geanalyseerd. Hieruit worden de tafel en de robots geïdentificeerd. Bij de robots wordt vervolgens gekeken naar de gekleurde vlakken bovenop de robot. De verschillende kleuren zorgen ervoor dat robots uit elkaar gehaald kunnen worden. Het rondje en rechthoek worden nog eens gebruikt om de richting van de robot te bepalen.

Op het linker beeldscherm staan bovenaan belangrijke spelmeldingen zoals de huidige opdracht. Daaronder is een de camera-afbeelding te zien. Hierop wordt de locatie van de robots en de opdracht weergegeven. Aan de rechter zijde staat informatie over: het huidige level, de huidige score en de high scores.

Op het rechter beeldscherm is bovenaan informatie over de robots te zien. Bij drie robots worden de huidige status en snelheid aangegeven. Onder de kop 'Routes' worden de routes van de robots weergegeven. Het onderste gedeelte van het beeldscherm geeft de werking van de beeldverwerking weer van tafel tot aan de herkenning van elke robot. Rechts onderin het beeldscherm staan nog een aantal statistieken.

### Het project

Vier studenten van de afdeling Technische Informatica hebben dit project gerealiseerd. Het grootste onderdeel van het project is de beeldverwerking. Er wordt op de monitoren visueel weergegeven hoe de beeldverwerking te werk gaat van herkenning van de tafel tot het identificeren van een robot. Er wordt gebruik gemaakt van verschillende technieken, zoals het vergelijken van de intensiteit van pixels of verhoudingen van verschillende deeltjes.

Beeldverwerking wordt steeds belangrijker en vaker gebruikt. Sommige camera's hebben bijvoorbeeld 'gezichtsherkenning' of een 'business card reader' functionaliteit. Hiervoor is allemaal geavanceerde beeldverwerking voor nodig.

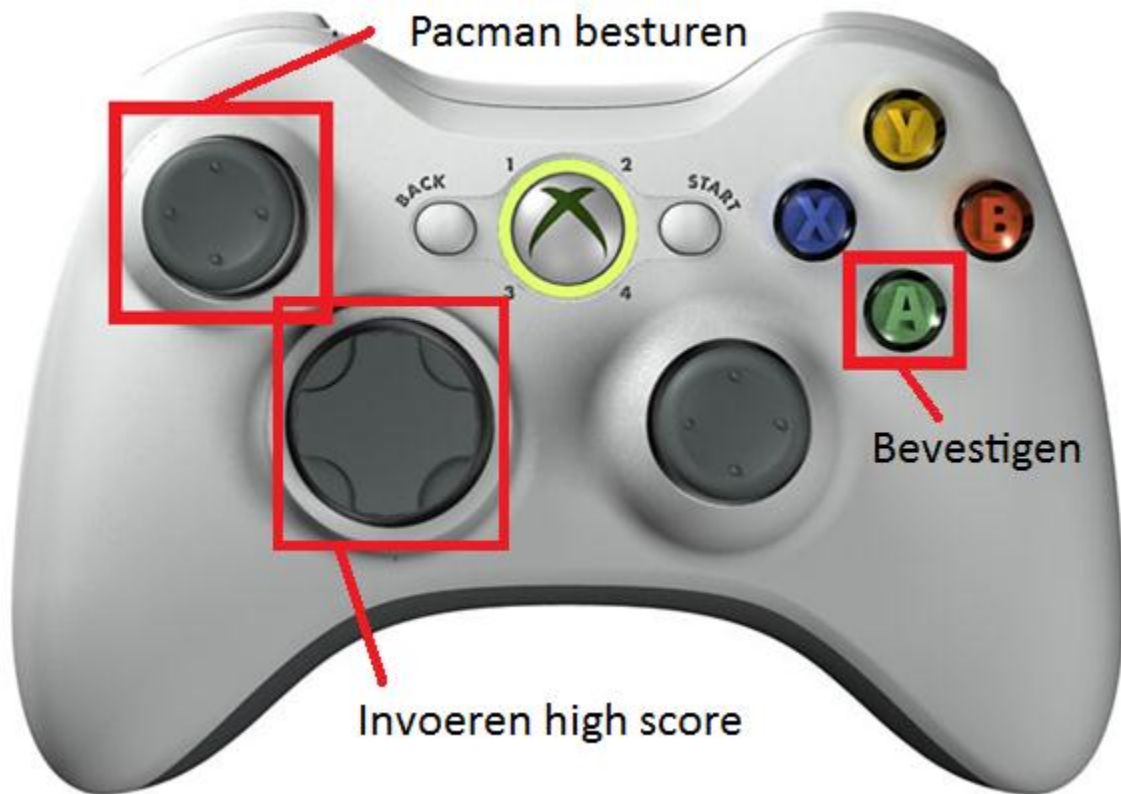
### Gebruiksaanwijzing

- Let op: Alle spelmeldingen staan bovenaan in het linker beeldscherm.
- Als er op een knop van de controller wordt gedrukt dan start het spel.
- Door middel van de linker joystick op de controller (zie figuur 1) wordt de middelste robot bestuurd. Deze is Pacman en is op het linker beeldscherm te herkennen aan het gele Pacman logo. De besturing werkt als volgt: Links ingedrukt houden om naar links te blijven rijden. Rechts ingedrukt houden om naar rechts te rijden, etc.
- Met Pacman moeten een aantal opdrachten worden uitgevoerd. Deze staan bovenaan het scherm vermeld en afgebeeld op de camera afbeelding.
- De overige robots moeten worden vermeden. Deze zijn op het beeldschermen te herkennen aan de zwarte spookjes. Als een spookje Pacman pakt dan is het Game Over en kan de high score ingevuld worden.



## Appendix E - Exhibit Information


- Er zijn in totaal 8 levels met verschillende opdrachten. Als deze allemaal behaald zijn, of als Pacman gepakt is door een spookje, kan de high score ingevuld worden.
- De high score moet als volgt worden ingevuld: Beweeg de D-pad op en neer voor de gewenste letter. Beweeg de D-pad naar rechts om de volgende letter in te vullen. Druk op de groene knop om te bevestigen.

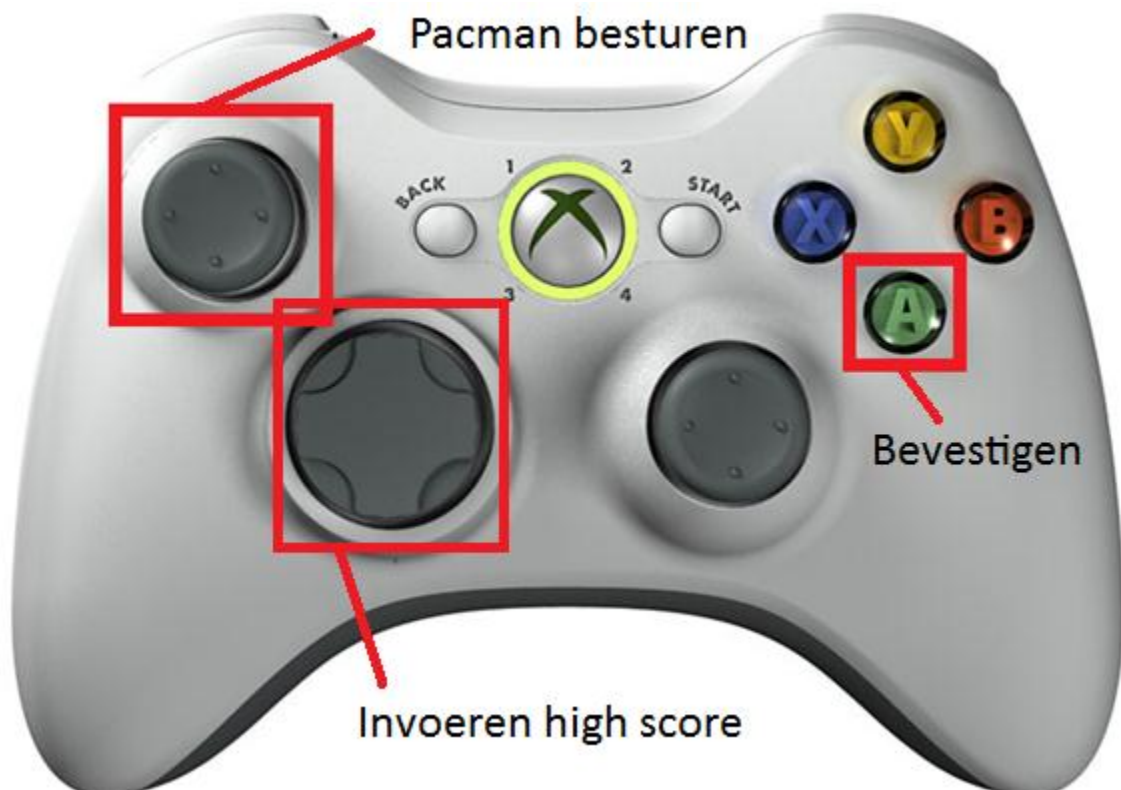


Figuur 1: Het rode vierkant geeft aan welke knopen er gebruikt worden.

# Gebruiksaanwijzing

## Uitleg spel: - Voer de opdrachten uit.

- Druk op een knop van de controller om te beginnen.
- De robot in het midden van de tafel is Pacman (jouw robot). Deze is te besturen met de linker joystick op de controller.
- Voer met Pacman de opdrachten uit. Deze opdrachten staan in het linker beeldscherm (bovenaan). Kijk uit voor de overige robots, als deze spookjes Pacman pakken, dan is het Game Over.
- Game Over? – Vul je naam in voor de high score. Met de D-pad zijn letters en plaats te bepalen. Druk op  als je klaar bent.



Figuur 1: Het rode vierkant geeft aan welke knoppen er gebruikt kunnen worden.

## **Uitleg beeldschermen:**

### **Links**

- Op het linker beeldscherm staat bovenaan de opdracht. Daaronder is een afbeelding van de tafel weergegeven. Hierop is de locatie van de robots en de opdracht te zien. Rechts staan het huidige level, de huidige score en de high scores vermeld.

### **Rechts**

- Op het bovenste deel van het rechter beeldscherm staat informatie over de robots met aan de rechterzijde informatie over de route van de robots.
- Op het onderste deel van het rechter beeldscherm is informatie te zien over de beeldherkenning.
- Rechtsonder staan statistieken en twee grafieken over de kwaliteit van de beeldverwerking.

## Maintenance Manual

### The Program

#### *Start-up*

When the computer is switched on Windows 7 32 bit will load. When this is finished the Pacman program will load automatically; this may take a few seconds. A GUI will fill up both monitors and in the background a robot control program will be running.

To start the program from the windows desktop click on the shortcut: "RUN PACMAN".

#### *Shut-down*

To close the program alt-F4 may be used. The standard windows desktop is now visible.

#### *Files*

All files are located on the c drive in the folder: "pacman".

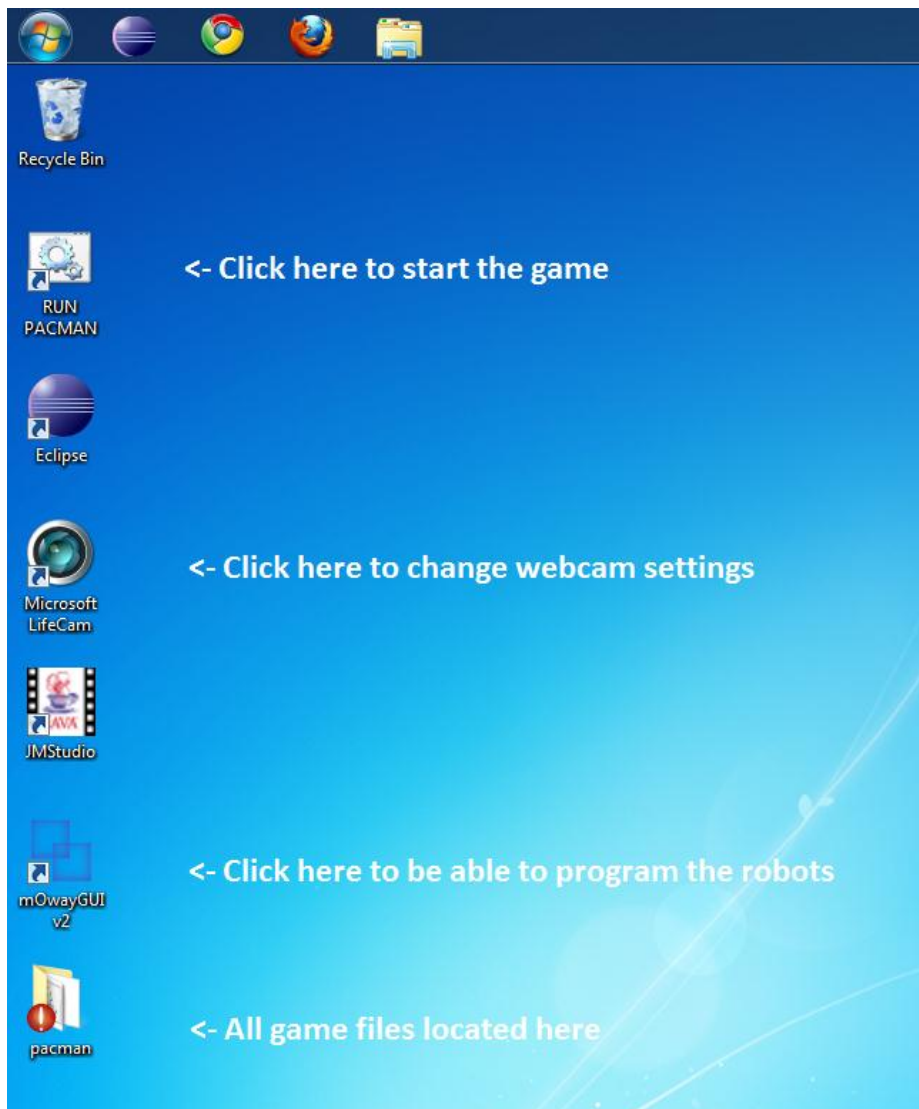


Figure 1: Desktop with explanations of icons

## Webcam

If the camera is removed or replaced simply make sure that the whole table is in view of the camera as can be seen in figure 2.

### Known Issue

The webcam loses its settings when cut off from power. The auto-focus function will therefore be reinstated. This feature can be annoying during game play. To switch off the auto focus the following steps must be undertaken:

- See figure 2: Open the option menu to the right (1), and click on the setting button (2). Click on properties (3).
- In the properties menu, click on the right tab “Camera control” and make sure that the Focus is set to 5 and that there is no check mark for “Auto”.

It may occur that the auto exposure feature of the webcam is insufficient resulting in table image which either too light or too dark. Unfortunately this setting is also reset when power is cut off to the webcam. Symptoms are robots with erratic driving behaviour and bad image recognition. The bad image recognition can be seen in on the right GUI; the names and colours of the robots should be stable. The auto-exposure can be changed as follows:

- See figure 2: Open the option menu to the right (1), and click on the setting button (2). Click on properties (3).
- In the properties menu, click on the right tab “Camera control” and make sure that the Exposure is set to a value so that the outer perimeter of the table is clearly white, while the table is a deep black. Figure 2 can be used as an indication.

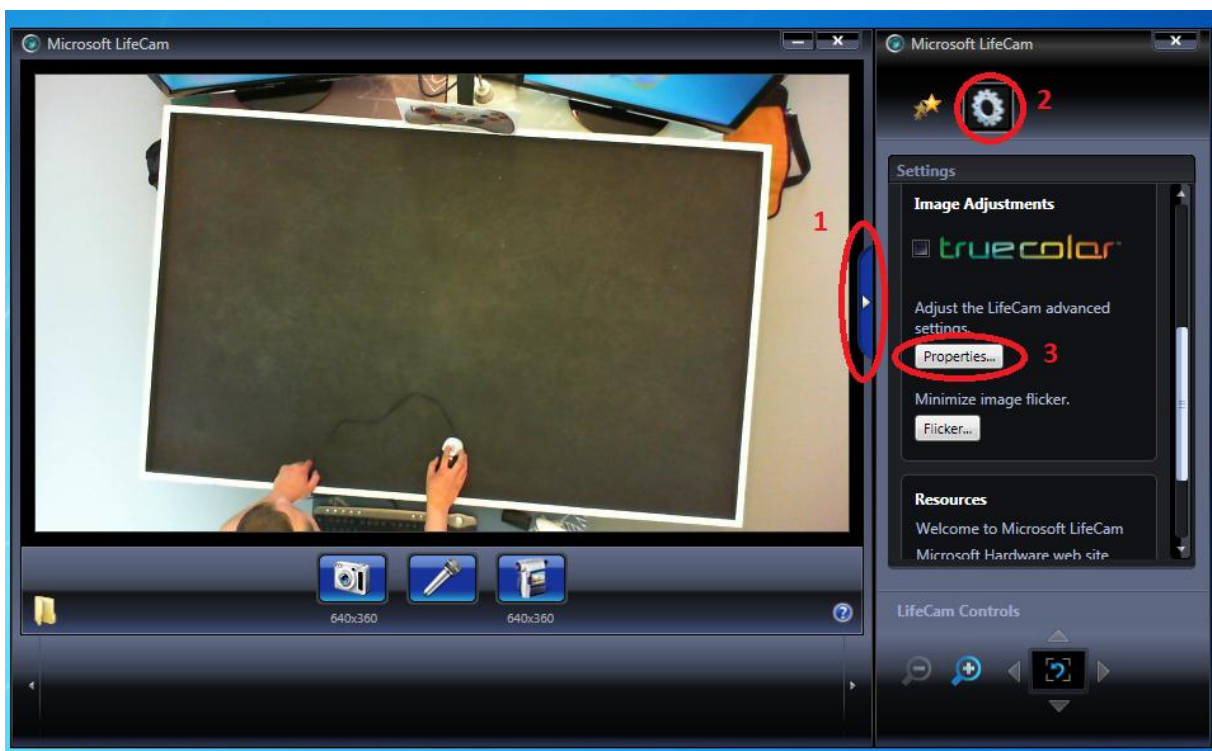


Figure 2: The Microsoft LifeCam software

## The Moways/Robots

The robots should be recharged every night. Make sure the robot is switched off. The switch can be found on the bottom of the robot. There are 6 robots available. Although al 6 robots can be used at the same time, we advise to use 3. If a robot no longer works, probably due to empty batteries, it may be replaced by any other robot.

**NOTE: When the robots are removed from the table they should be switched off.**

If a mOway robot breaks down please contact the mOway group ([www.moway-robot.com](http://www.moway-robot.com)). Our contact person at the time of writing was Elena Merino. Her e-mail address is: [e.merino@minirobots.es](mailto:e.merino@minirobots.es)

The program loaded on the robot is different for each robot. Via the mOway GUI the program can be loaded onto the robots. This program has a shortcut on the desktop. The robots' programs can be found in the folder <pacman/svn/installs>. Each robot has a corresponding file <moway#.mwp>. The # corresponds to the address (single digit) of the robot and can be found on the bottom of the robot or in the following table.

Robot number	Name	Rectangle Colour	Circle Colour
2	Daco	Blue	Red
3	Bram	Blue	Blue
4	Rick	Red	Red
5	Daan	Green	Green
6	Stan	Green	Blue
7	Roel	Red	Green

## Cover Plates

The cards on top of the mOways are replaceable. They were created by the Appel Reklame company in Delft. The plaatje.pdf file was used as the blueprint.

## Computer

The computer has been assembled by the Pacman group themselves. The following parts were used:

Motherboard	MSI H55-E33 motherboard
Processor	Intel i3 530
Power supply	Be quiet! Pure Power BQT L7-430W
Hard drive	Samsung Spinpoint F3 Desktop Class HD502HJ
RAM	Kingston ValueRAM 2GB DIMM-240pins1333MHz/PC3-10600
DVD drive	Sony Optiarc AD-5240S
Case	Cooler Master Elite 335
Graphics card	Sapphire RADEON HD 5750
Monitor	2x Samsung SyncMaster P2450H

## Controller

An Xbox controller with USB is used for this project.

## Software

Software used during this project:

- Windows 7 32 bit
- Java Media Framework
- Moway
- Eclipse
- Visual Studio
- Microsoft LifeCam

## Programmers

Contact information with the creators of the project is displayed in the table below. Contact via e-mail is preferred.

Name	E-mail Address	Telephone Number
<b>Daco Harkes</b>	dc.harkes@gmail.com	0612100358
<b>Bram Kol</b>	bram.kol@planet.nl	0628180843
<b>Rick de Ridder</b>	acidkurck@gmail.com	0610752810
<b>Daan Wilmer</b>	daan.wilmer@gmail.com	0644896059