Final report – System Design Project

Juozas Kaziukėnas – s0820151

Initial work

I started my contributions in my team by working in robot construction team in initial stages of robot development. However without having much time to have any substantial influence on robot design, apart from expressing the idea of having a lightweight fast robot, I moved to movement and strategy group quite quickly after realising that we had lack of people with good Java experience. Obviously the first task was to familiarize with the code base which was already developed and also quickly figure out problems like Bluetooth connection between controlling computer and the robot itself.

Just after a week weeks I noticed that we had problems I had experienced numerous times before: chaotic nature of development when without any supervision. Communication between team members was close to not existing and project was moving to a direction which I knew would fail eventually. It seemed logical to step up and take on the managing role which is where I later contributed most. Similarly to coming from construction to movement teams, at that time I also needed to review what has already been done with managing and make some adjustments.

Movement system

My job in movement was to work in system design, performance problems and integration with vision and leave smaller parts assigned to others. Our entire code base for movement and strategy system was written using Java and surprisingly a lot of what we had done initially ended up staying. For example idea of having client the robot and server the computer which had data coming in from vision.

I worked on a lot of different code parts so I'm not going to mention all of them here, just the ones I believe were we most important. My first task was to get integration between vision and movement systems and design basic movement system. For integration I had chosen to use standard I/O with movement application spawning a vision instance and reading data from output it would be writing to. This was implemented quickly and worked fine whole project time. And for movement system design I had chosen to separate concerns into strategies, executors and processors (vision program, local prepared file and simulator) with runner part controlling the instantiation. All of this worked really well and had allowed others to develop and iterate quickly.

State machine-based strategy was something I'm most proud in our system design. I wanted to have a strategy which would be easy to work with and when something would stop working it would be easy enough to figure out why. The strategies we had before were a

mess of maths formula, which we supposed to be working by wiki saying, but didn‘t. Together with state machine I also made strategies layered – lowest layer handles how to move, middle layer what points move to and top level is the state machine handling how to react to different states. After I had implemented this as a wireframe, my team stepped in and developed all states-specific code and maths required to calculate the required points.

To make things easy to control I had created a GUI which had all the controls needed – choosing strategies, executors (Bluetooth or simulator), goal to attack, colour of the plate, pausing/resuming robot movements etc. What is more, after the simulator was developed it‘s frame of pitch visualisation was integrated to whole system to also visualise real pitch. This development made it easier to calibrate various parameters and also to see what state is stored inside computer. For example for some of the strategies we had visualisation of points we want to move to or boundaries we want to avoid, which proved to be very useful to improve strategies.

Managing

We had setup a wiki site to log our activities, work on documents together and share ideas. The problem with this was that it wasn‘t as active as it needed to be, which made it quite pointless. Asking and then reminding everyone every single day that writing there is a requirement, in the end produced some good results. For example quite a few people have logged their activity every single day (*this was a requirement for everyone, but ...*) or then I remade the TODO pages to make it easier for tasks to be assigned and updates.

Making sure everyone knows what they need to do and deliver it on time was my main job while on this project. This is an essential part of any team project and I needed to fix this ASAP. Firstly, I setup team meetings to discuss the problems we are having and how to achieve milestone goals we had imposed. Once this was done the next logical step was to figure out the atomic elements of those goals and assigning them to team members which would be most capable in doing them. I believe this has worked really well as we quickly became one of the leading teams in milestones 2 and 3 while being the worst in milestone 1 – all because we were very effective in pulling all *pieces* together.

From a start we had an idea of having a small, light and fast robot which winning strategy would be to execute fast and precise. I had this in my mind throughout the development period and made sure robot design and code architecture doesn’t change this. At one point we had problems with robot going straight which were caused by wheels and motors imperfections – I put testing this as top priority and in no time we had this issue documented and fixed that our initial idea would still hold.

Being in this position required making decisions when it was needed. For example when we experienced serious problems with SVN server we used for storing our source code, I moved whole repository to a new server which was offsite university network and had proved to

work without any problems. And obviously when it came to assigning tasks, arranging meetings and making sure that work gets done, I was the one in charge of this.

What I have learned and what I think about my performance

When I proposed I'd become a manager I remember joking „I'll be the one everybody hates". I believe this had happened and I liked it. From my years of experience in professional software development I have learned that developers are very passionate and think about their code as something very personal. Thus trying to steer them into different direction is a hard work and they (usually) hate managers for that. I think I succeeded in this for the most part and in the end we had finished all what we wanted to finish.

My goal was to make our team work efficiently and allow them to do what they know best. For the later one I needed to create a wireframe structure of strategies and movement application plus GUI because it was quite visible that others were lacking experience to make this happen. After this was done development pace increased a lot and we had managed to create working strategy for final match and more.

Final report – System Design Project

Juozas Kaziukėnas – s0820151

Initial work

I started my contributions in my team by working in robot construction team in initial stages of robot development. However without having much time to have any substantial influence on robot design, apart from expressing the idea of having a lightweight fast robot, I moved to movement and strategy group quite quickly after realising that we had lack of people with good Java experience. Obviously the first task was to familiarize with the code base which was already developed and also quickly figure out problems like Bluetooth connection between controlling computer and the robot itself.

Just after a week weeks I noticed that we had problems I had experienced numerous times before: chaotic nature of development when without any supervision. Communication between team members was close to not existing and project was moving to a direction which I knew would fail eventually. It seemed logical to step up and take on the managing role which is where I later contributed most. Similarly to coming from construction to movement teams, at that time I also needed to review what has already been done with managing and make some adjustments.

Movement system

My job in movement was to work in system design, performance problems and integration with vision and leave smaller parts assigned to others. Our entire code base for movement and strategy system was written using Java and surprisingly a lot of what we had done initially ended up staying. For example idea of having client the robot and server the computer which had data coming in from vision.

I worked on a lot of different code parts so I'm not going to mention all of them here, just the ones I believe were we most important. My first task was to get integration between vision and movement systems and design basic movement system. For integration I had chosen to use standard I/O with movement application spawning a vision instance and reading data from output it would be writing to. This was implemented quickly and worked fine whole project time. And for movement system design I had chosen to separate concerns into strategies, executors and processors (vision program, local prepared file and simulator) with runner part controlling the instantiation. All of this worked really well and had allowed others to develop and iterate quickly.

State machine-based strategy was something I'm most proud in our system design. I wanted to have a strategy which would be easy to work with and when something would stop working it would be easy enough to figure out why. The strategies we had before were a

mess of maths formula, which we supposed to be working by wiki saying, but didn‘t. Together with state machine I also made strategies layered – lowest layer handles how to move, middle layer what points move to and top level is the state machine handling how to react to different states. After I had implemented this as a wireframe, my team stepped in and developed all states-specific code and maths required to calculate the required points.

To make things easy to control I had created a GUI which had all the controls needed – choosing strategies, executors (Bluetooth or simulator), goal to attack, colour of the plate, pausing/resuming robot movements etc. What is more, after the simulator was developed it‘s frame of pitch visualisation was integrated to whole system to also visualise real pitch. This development made it easier to calibrate various parameters and also to see what state is stored inside computer. For example for some of the strategies we had visualisation of points we want to move to or boundaries we want to avoid, which proved to be very useful to improve strategies.

Managing

We had setup a wiki site to log our activities, work on documents together and share ideas. The problem with this was that it wasn‘t as active as it needed to be, which made it quite pointless. Asking and then reminding everyone every single day that writing there is a requirement, in the end produced some good results. For example quite a few people have logged their activity every single day (*this was a requirement for everyone, but ...*) or then I remade the TODO pages to make it easier for tasks to be assigned and updates.

Making sure everyone knows what they need to do and deliver it on time was my main job while on this project. This is an essential part of any team project and I needed to fix this ASAP. Firstly, I setup team meetings to discuss the problems we are having and how to achieve milestone goals we had imposed. Once this was done the next logical step was to figure out the atomic elements of those goals and assigning them to team members which would be most capable in doing them. I believe this has worked really well as we quickly became one of the leading teams in milestones 2 and 3 while being the worst in milestone 1 – all because we were very effective in pulling all *pieces* together.

From a start we had an idea of having a small, light and fast robot which winning strategy would be to execute fast and precise. I had this in my mind throughout the development period and made sure robot design and code architecture doesn’t change this. At one point we had problems with robot going straight which were caused by wheels and motors imperfections – I put testing this as top priority and in no time we had this issue documented and fixed that our initial idea would still hold.

Being in this position required making decisions when it was needed. For example when we experienced serious problems with SVN server we used for storing our source code, I moved whole repository to a new server which was offsite university network and had proved to

work without any problems. And obviously when it came to assigning tasks, arranging meetings and making sure that work gets done, I was the one in charge of this.

What I have learned and what I think about my performance

When I proposed I'd become a manager I remember joking „I'll be the one everybody hates". I believe this had happened and I liked it. From my years of experience in professional software development I have learned that developers are very passionate and think about their code as something very personal. Thus trying to steer them into different direction is a hard work and they (usually) hate managers for that. I think I succeeded in this for the most part and in the end we had finished all what we wanted to finish.

My goal was to make our team work efficiently and allow them to do what they know best. For the later one I needed to create a wireframe structure of strategies and movement application plus GUI because it was quite visible that others were lacking experience to make this happen. After this was done development pace increased a lot and we had managed to create working strategy for final match and more.

Final report – System Design Project

Juozas Kaziukėnas – s0820151

Initial work

I started my contributions in my team by working in robot construction team in initial stages of robot development. However without having much time to have any substantial influence on robot design, apart from expressing the idea of having a lightweight fast robot, I moved to movement and strategy group quite quickly after realising that we had lack of people with good Java experience. Obviously the first task was to familiarize with the code base which was already developed and also quickly figure out problems like Bluetooth connection between controlling computer and the robot itself.

Just after a week weeks I noticed that we had problems I had experienced numerous times before: chaotic nature of development when without any supervision. Communication between team members was close to not existing and project was moving to a direction which I knew would fail eventually. It seemed logical to step up and take on the managing role which is where I later contributed most. Similarly to coming from construction to movement teams, at that time I also needed to review what has already been done with managing and make some adjustments.

Movement system

My job in movement was to work in system design, performance problems and integration with vision and leave smaller parts assigned to others. Our entire code base for movement and strategy system was written using Java and surprisingly a lot of what we had done initially ended up staying. For example idea of having client the robot and server the computer which had data coming in from vision.

I worked on a lot of different code parts so I'm not going to mention all of them here, just the ones I believe were we most important. My first task was to get integration between vision and movement systems and design basic movement system. For integration I had chosen to use standard I/O with movement application spawning a vision instance and reading data from output it would be writing to. This was implemented quickly and worked fine whole project time. And for movement system design I had chosen to separate concerns into strategies, executors and processors (vision program, local prepared file and simulator) with runner part controlling the instantiation. All of this worked really well and had allowed others to develop and iterate quickly.

State machine-based strategy was something I'm most proud in our system design. I wanted to have a strategy which would be easy to work with and when something would stop working it would be easy enough to figure out why. The strategies we had before were a

mess of maths formula, which we supposed to be working by wiki saying, but didn‘t. Together with state machine I also made strategies layered – lowest layer handles how to move, middle layer what points move to and top level is the state machine handling how to react to different states. After I had implemented this as a wireframe, my team stepped in and developed all states-specific code and maths required to calculate the required points.

To make things easy to control I had created a GUI which had all the controls needed – choosing strategies, executors (Bluetooth or simulator), goal to attack, colour of the plate, pausing/resuming robot movements etc. What is more, after the simulator was developed it‘s frame of pitch visualisation was integrated to whole system to also visualise real pitch. This development made it easier to calibrate various parameters and also to see what state is stored inside computer. For example for some of the strategies we had visualisation of points we want to move to or boundaries we want to avoid, which proved to be very useful to improve strategies.

Managing

We had setup a wiki site to log our activities, work on documents together and share ideas. The problem with this was that it wasn‘t as active as it needed to be, which made it quite pointless. Asking and then reminding everyone every single day that writing there is a requirement, in the end produced some good results. For example quite a few people have logged their activity every single day (*this was a requirement for everyone, but ...*) or then I remade the TODO pages to make it easier for tasks to be assigned and updates.

Making sure everyone knows what they need to do and deliver it on time was my main job while on this project. This is an essential part of any team project and I needed to fix this ASAP. Firstly, I setup team meetings to discuss the problems we are having and how to achieve milestone goals we had imposed. Once this was done the next logical step was to figure out the atomic elements of those goals and assigning them to team members which would be most capable in doing them. I believe this has worked really well as we quickly became one of the leading teams in milestones 2 and 3 while being the worst in milestone 1 – all because we were very effective in pulling all *pieces* together.

From a start we had an idea of having a small, light and fast robot which winning strategy would be to execute fast and precise. I had this in my mind throughout the development period and made sure robot design and code architecture doesn’t change this. At one point we had problems with robot going straight which were caused by wheels and motors imperfections – I put testing this as top priority and in no time we had this issue documented and fixed that our initial idea would still hold.

Being in this position required making decisions when it was needed. For example when we experienced serious problems with SVN server we used for storing our source code, I moved whole repository to a new server which was offsite university network and had proved to

work without any problems. And obviously when it came to assigning tasks, arranging meetings and making sure that work gets done, I was the one in charge of this.

What I have learned and what I think about my performance

When I proposed I'd become a manager I remember joking „I'll be the one everybody hates". I believe this had happened and I liked it. From my years of experience in professional software development I have learned that developers are very passionate and think about their code as something very personal. Thus trying to steer them into different direction is a hard work and they (usually) hate managers for that. I think I succeeded in this for the most part and in the end we had finished all what we wanted to finish.

My goal was to make our team work efficiently and allow them to do what they know best. For the later one I needed to create a wireframe structure of strategies and movement application plus GUI because it was quite visible that others were lacking experience to make this happen. After this was done development pace increased a lot and we had managed to create working strategy for final match and more.