Project Plan

This is the project plan for Project 4: the CoPilot app. In this document we will define the client, the team, and the plan we have developed to create this project from this moment forward.

Client

We have been assigned to work with a project from Volvo Construction Equipment. In the words of our contact in the company, the scope of the project is “to implement a G-code program system for autonomous constructions equipment and propose possibility and its suitability to use that system. The evaluation should be carried out by implementing a system in a CoPilot and test the program on a simulated machine”.

In short, we need to create an Android app for the workers to plan the autonomous machines’ paths and to export it to the CoPilot tablet (used in the vehicles), for the operator to program the machines for work.

Team

Our team is formed by six people with the following roles.

- Project manager:
  - Joaquín García Benítez
- Client contact:
  - Clara Torre García-Barredo
- Android app developer:
  - Iván Muñiz
  - Tommy Ernsund
  - Mathias Svensson Karlsson
- Supporting:
  - Viking Forsman

Those are our assigned roles, but apart from those specific activities, we will all participate in all stages of development in the project. Each of us will be assigned different tasks to complete based on their specific skills.

To organize the work, we will meet a minimum of two times a week; Mondays from 13:30h to 15h, right before our weekly presentation with the steering group, and before our weekly Skype call with our client. Additionally, more meetings can be added if needed. In those meetings, we will revise the work each of us has done since the last meeting, and the remaining work. We will also prepare for upcoming presentations or meetings, either with the steering group or the client, and assign new tasks for everybody to complete before the next meeting.

Our routines regarding time reporting and configuration management are as follows: Regarding time reporting; from now on we have decided to write updates in our communication group on Slack with the number of minutes we have been working every time we do something related to this project. That way, our time spent is most accountable. Once a day, those times will be added and updated in the weekly presentation slides.

Regarding configuration management, we will use GitHub. To do so, we have arranged our Android Studios to work with it, so we can push our changes to the repository directly from the app.

The structure of the Git repository folder is as follows: We have divided the first folder into two subfolders: “CoPilotApp”, in which all of our source code will be kept, and “Documents”, where we will keep the documentation deliverables (under the folder “Deliverables”), the PDF versions of the slides used in the weekly meetings with the steering group (under the folder “Weekly Presentations”) and also the PDF versions of the group presentations slides (under the folder “Group Presentations”). We also have a Wiki inside GitHub for further explanation on every topic, in which we will add information as we develop the product. The README.md document in the repository explains all of this structure as well, for easy access.

After the project is finished, we will add a section to the Wiki that explains how the application can be properly set up for use.

We will ensure the high quality of our end product by testing our code thoroughly, following standards to maintain a consistent quality level throughout the entire development of the project. We will try to get regular feedback from the client, as that is the best way to get a sense of what needs to be improved for the client to be satisfied.

To ensure the quality, we have established the following routines:

- Peer review. We will run our own code through the rest of our teammates in order for the code to be further reviewed and approved, apart from it being correctly tested.

- Continuous contact with the client. We have established weekly meetings with the client to ensure that they are properly kept up to date with the development of the project. Any further contact will be made between the client and our client contact, Clara, via email. This way, we can make sure that the steps taken in regards to development of the app are in the correct direction, and not something that has to be undone later on.
- Clear documentation. We have organized so all members of the team collaborate on working with documentation deliverables, as well as reviewing them. Alongside this, we review the times reported by each member every day to ensure that nobody spends too much time working on something with a low priority and/or low quality.
- Functionality testing. Regularly, during the development of the application, we will test its functionality to ensure it's in line with the requirements obtained from the client, and, as stated above, we will also check with the client to make sure their requirements stay consistent.
- Acceptance testing. At the end of the project, we will run the application once again with the client to make sure the quality is as expected. If not, we will improve our product until it satisfies the client.

Time plan

The overall time plan is as follows:

- Project plan: We want to have it finished by November 19, so if there is anything that needs to be changed, we will have enough time to correct it before it is due.
- Detailed design description - preliminary: We have decided to try to have the first version ready for December 3. This way we will still have time to make any necessary adjustments.
- Product - functional: If the client doesn't demand any other deadline prior to the course one, which he hasn't yet, we will have the first version done by December 3, like the detailed design description.
- Detailed design description - final: We have decided to have the final version of this ready by January 13, to have time for adjustments.
- Product - final: We want to have the final version done by January 13, to have time to fix things if they are not working as intended.
- Project report: We also plan on having this done by January 13, to have a small gap between the day we finish and its due date.

Development

Currently the client is developing autonomous vehicles for carrying loads and working along a predetermined path. For this purpose the client requires an Android app designed for both mobiles and CoPilot tablets, which should be able to create these predetermined paths and allow the operators to use them. Through elicitation with the client, since when we started with our first meeting the client only had a high level idea of the product, we've determined the specific requirements described below.

Functional requirements

The client has expressed a specific need for the system to be able to handle G-Code for the pathing system, which is an old, very basic programming language for CNC machines. Furthermore, there is a requirement for accuracy beyond that of regular GPS technologies. The overall functionality required can be divided into the following:

- Login to the system, only a authenticated user is allowed access to the functionality.
- Logout from the system.
- Create a path with real world coordinates that represent the path the autonomous vehicles are supposed to follow. A path is created by walking to a location in the real world, and marking the location in the app by selecting what type of path it is. effectively creating a point on the path. The following path types are available.
    - Line, a simple straight line between the new and previous path.
    - Curve, a curve between the between the new and previous path.
    - Loading area, designates an area where the autonomous vehicles get their load.
    - Deposit area, designates an area where the autonomous vehicles deposit their load.
- Edit an existing path. This should allow for individual points on the path to be edited, that is, to either move them, change the path type or simply remove it.
- Remove an existing path.
- Export a path. This functionality is required by the operator to export/upload a pre-existing path to the autonomous vehicles.
- View the path as G-code, a path should be able to be displayed as pure G-code instead of using the app interface.
- Edit the path as G-code, similarly the path should also be editable through the viewable G-code.

- Reserve machines. To avoid concurrency issues, the autonomous vehicles need to be reservable for a single operator.
- Release machines, effectively removing the reservation.
- Execute a path on reserved vehicles. Start running a path on autonomous vehicles that has been previously exported.

A use case diagram with its definitions was created to define the functionality (see appendix). Beyond these requirements, there is also a need for the system to be able to validate these paths, essentially requiring a built-in path simulator.

Initially the prototype should not account for real-world terrain, but rather a simple 2-dimensional representation, which should be extended upon, to account for gradients and real terrain in later iterations.

More detailed requirements are detailed in the appendix 3 below.

External system requirements

The existing systems being interacted with are the CoPilot interface of various types of construction site machines, which is an in-house version of Android. There are two different actors in this system and those are the planner and the operator. The planner runs the app on a mobile device, while the operator runs it on a tablet device, mounted within the machines. The difference between these two is the ability for the operator to reserve ownership of a list of autonomous vehicles, hence avoiding concurrency issues, and to actually execute a path on these vehicles. We've also determined that there will be a need for an external database system for storing the paths in a centralized manner.

Development constraints

There exists very specific restraints for the development due to the use of G-code, which is originally designed for fixed CNC machines, not moving autonomous vehicles. G-code supports simple movement along a line or curve, as well as various CNC tool operations, such as rotating an actuator arm.

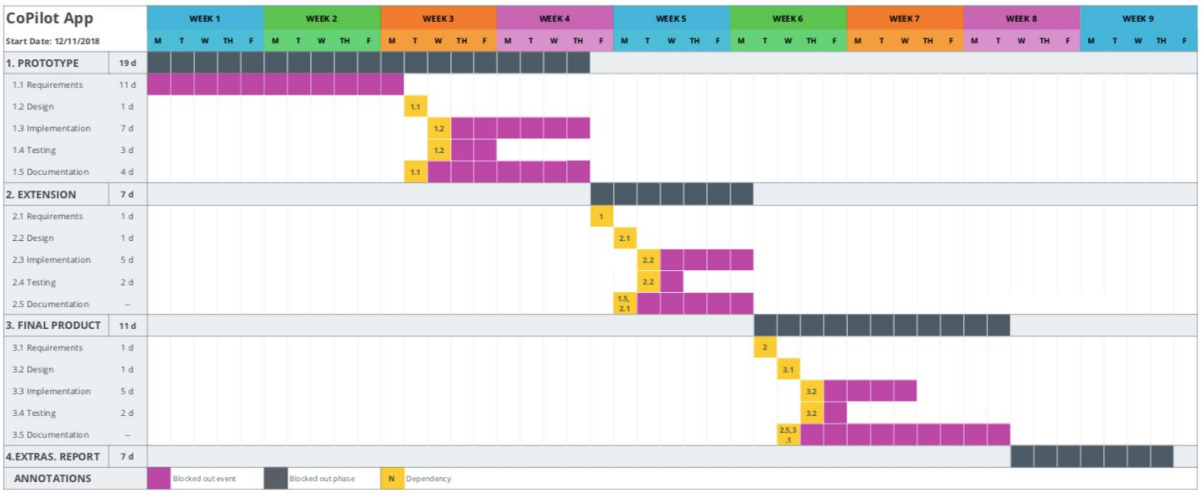
Backlog

The initial backlog contains items that are required to satisfy all product requirements. Each item in our backlog has a priority score from 1 to 10. In this scoring system, the value 1 represents the lowest priority and 10 represents the highest priority. More items might be added to the project backlog in GitHub Projects as a response to development changes or feedback from client.

The initial backlog:

- (Priority: 6) Create a “Wizard of Oz” representation of the app.
- (Priority: 6) Create use-case diagram for the system requirements.
- (Priority: 6) Create activity diagram for each use case.
- (Priority: 8) Install Android Studio (and java if necessary).
- (Priority: 8) Set up a GitHub repository for Android Studio app.
- (Priority: 4) Set up a kanban board on GitHub (for synchronisation).
- (Priority: 2) Set up a wiki page for the project on GitHub.
- (Priority: 8) Set up a Slack group for communication within the team.
- (Priority: 7) Write project plan in accordance to the course description.
- (Priority: 6) Write the first draft of the Detailed Design Description.
- (Priority: 6) Write the Detailed Design Description.
- (Priority: 7) Estimate effort and create Gantt chart.
- (Priority: 4) Implement the GUI design for the app (colors, font, logo, etc).
- (Priority: 9) Implement a prototype version of the app (only basic functionality).
    - Add all activities (login screen, list screen, display screen, edit screen).
    - Add navigation between activities.
    - Add functionality to login to the app.
    - Add functionality to logout from the app.
    - Add functionality to display created paths (list view).
    - Add functionality to create paths.
- (Priority: 9) Implement extended functionality prototype (refine the functionality)
    - Add functionality to edit paths (change or remove point).
    - Add functionality to remove paths.
    - Add functionality of path validation, only allow realistic turns.
    - Add functionality to toggle view mode (visual representation or G-code).
    - Add functionality to export path to other users.
- (Priority: 10) Implement the final product.
    - Add more G-code instruction for onloading and offloading.
    - Perform quality assurance methods, remove bugs.
    - Perform acceptance testing (client present or send result to client)
- (Priority: 8) Write the project report.
    - Write a brief introduction about the client and the project.
    - Write section about the planning phase.
    - Write section about the design phase.
    - Write section about the implementation phase.
    - Write section about project result and conclusion.
    - Write about the result of acceptance testing and how it was implemented.

Gantt chart



The Gantt chart represents how we plan on dividing our time during this course, and also the dependencies we have that exist between the different phases of our development.

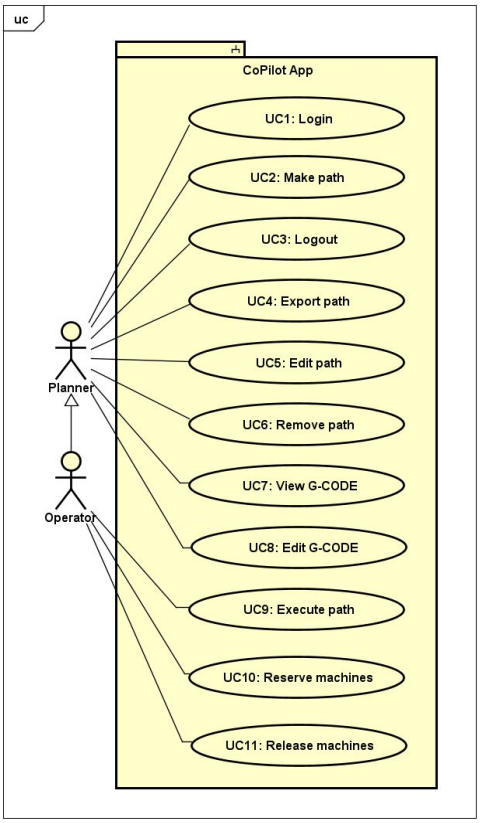
The first major phase is the prototype, which is the first version of the product we are going to develop. It has to be done by the 6th of December, and we have spent two weeks getting to meet the client, the team, the project, and gathering the requirements, so we have planned to complete the first version of the product in a short time to be able to meet the deadline. However, that gives us more time to fix problems that may come up with this first version later on.

The second major phase is the extension. The client has asked for us to consider only one operator and one autonomous machine in the prototype, so the extension is the time in which we have to improve the app in order to have multiple operators and multiple autonomous machines.

The third major phase is the final product. In this, we have more requirements from the client to tackle only if we have enough time to do so. We want to forget about the flat earth principle in this version of the product, and measure soil consistency and inclination.

After those three main phases are done, we have some extra time to prepare the project report and the presentation that goes along, and make any improvements the client considers we need for the product to be completed.

Appendix 1: Use Case Diagram



Use case list:

- UC1 - Login to the system
- UC2 - Make new paths
- UC3 - Logout
- UC4 - Export
- UC5 - Edit path
- UC6 - Remove path
- UC7 - View G-code
- UC8 - Edit G-code
- UC9 - Execute path
- UC10 - Reserve machines
- UC11 - Release machines

Actor list:

- Planner - Creates or edits path's for autonomous vehicles using the mobile app, can also view the corresponding G-code and edit it.
- Operator - The operator controls which G-code instructions (path) will be used by the control unit in the autonomous vehicles. Operator has access to all functionality a planner has (generalization).

ID:	UC1
Title:	Login to the system
Priority:	High
Description:	Users will be prompted to login with their Volvo account information before they can access the system.
Primary actor:	Planner
Pre-conditions:	1. The user has a Volvo account. 2. The user is trying to login to the CoPilot App. 3. The user is not already logged in.
Post-conditions:	1. The user logged in to the system. 2. The user can access the system functions.
Main success scenario:	1. The user types his username and password. 2. The system verifies his credentials. 3. The user gains access to the system functionalities.
Alternative flow:	2. Invalid username or password.
Frequency of use:	Every time the user wants to access the system functionalities.
Created by:	Iván Muñiz
Date created:	2018/11/14
Last updated by:	Iván Muñiz
Last updated:	2018/11/14

ID:	UC2
Title:	Make new paths
Priority:	High
Description:	Let the user create new paths for the autonomous machines.
Primary actor:	Planner
Pre-conditions:	1. User is logged in
Post-conditions:	1. A new path is created and saved.
Main success scenario:	1. User clicks the create new path button. 2. User creates the path. 3. User clicks save path. 4. System validates the path. 5. System saves the path.
Alternative flow:	4. System deems the path invalid - present details to user. 5. System error - cannot save path.
Frequency of use:	Every time the user wants to create a new path.
Created by:	Iván Muñiz
Date created:	2018/11/14
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

ID:	UC3
Title:	Logout
Priority:	Medium
Description:	Let the user terminate his session to the system.
Primary actor:	Planner
Pre-conditions:	1. User has an active session in the system.
Post-conditions:	1. User is logged out of the system.
Main success scenario:	1. User clicks on the logout button. 2. The system logs the user out.
Alternative flow:	2. System error - failed to logout.
Frequency of use:	Whenever the user wants to logout.
Created by:	Iván Muñiz
Date created:	2018/11/14
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

ID:	UC4
Title:	Export
Priority:	Medium
Description:	Let the user export a path to another device
Primary actor:	Planner
Pre-conditions:	1. User should be logged in. 2. A path should be available.
Post-conditions:	1. Path is exported successfully to the destination device.
Main success scenario:	1. User selects a path. 2. User presses the export button. 3. User selects a destination device. 4. System exports path to the destination device.
Alternative flow:	3. No destination devices available. 4. System error - transfer is interrupted.
Frequency of use:	Every time an export (update) is to be made to another device
Created by:	Mathias Svensson Karlsson
Date created:	2018/11/14
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/14

ID:	UC5
Title:	Edit path
Priority:	Medium
Description:	Let a already existing path be modified
Primary actor:	Planner
Pre-conditions:	1. User is logged in. 2. An existing path is available.
Post-conditions:	1. Existing path is replaced by the edited version.
Main success scenario:	1. User selects a path. 2. User presses the edit button. 3. User selects a path node. 4. User presses edit. 5. User updates new coordinates. 6. System validates the path. 7. System commits the updated to the existing path.
Alternative flow:	2. User presses cancel. 4. User presses delete. 5. Coordinates are invalid. 6. System deems the path invalid - present details to user. 7. System error - cannot update path.
Frequency of use:	Every time an edit is to be made to an existing path
Created by:	Mathias Svensson Karlsson
Date created:	2018/11/14
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

ID:	UC6
Title:	remove path
Priority:	Low
Description:	Let an already existing path be deleted
Primary actor:	Planner
Pre-conditions:	1. User is logged in. 2. An existing path is available.
Post-conditions:	1. Existing path is removed.
Main success scenario:	1. User selects a path. 2. User presses the remove button. 3. User presses the yes button on confirmation to remove. 4. System removes the path.
Alternative flow:	3. User presses no. 4. System error - path could not be removed.
Frequency of use:	Every time an delete is to be made to an existing path
Created by:	Joaquín García Benítez
Date created:	2018/11/15
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

ID:	UC7
Title:	View G-code
Priority:	Medium
Description:	View an existing path in G-code format
Primary actor:	Planner
Pre-conditions:	1. User is logged in 2. An existing path is selected
Post-conditions:	1. Display existing path in G-code format.
Main success scenario:	1. The user clicks on the change view button. 2. The system change display mode
Alternative flow:	
Frequency of use:	Every time the user wants to view the path in G-code format
Created by:	Viking forsman
Date created:	2018/11/15
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

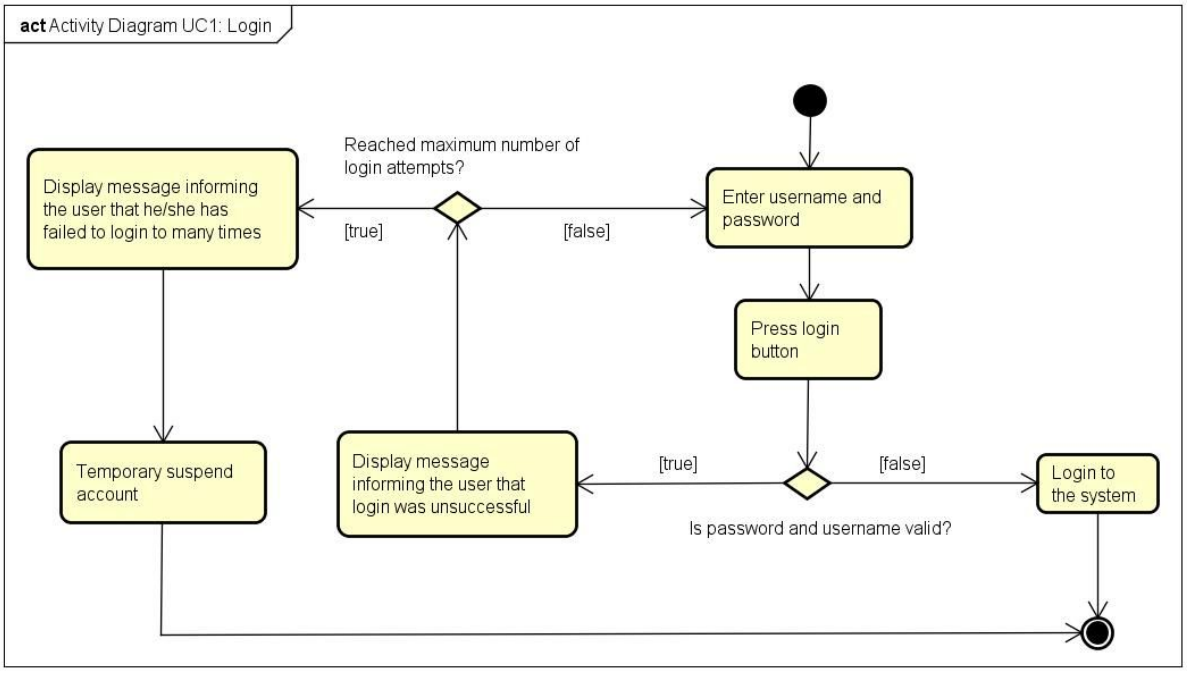
ID:	UC8
Title:	Edit G-code
Priority:	Low
Description:	Edit an existing path in G-code format
Primary actor:	Planner
Pre-conditions:	1. User is logged in 2. An existing path is available 3. The user view the path in G-code format
Post-conditions:	1. Changed existing G-code path.
Main success scenario:	1. User selects a path 2. User presses the edit button 3. User selects a G-code instruction 4. User updates the selected instruction 5. System validates the instruction 6. The update is committed to the existing path
Alternative flow:	5. System deems the path not valid. 6. Internal error - cannot write update
Frequency of use:	Every time the user wants to edit a path in G-code format
Created by:	Viking forsman
Date created:	2018/11/15
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

ID:	UC9
Title:	Execute path
Priority:	Medium
Description:	Let an operator run the path on a number of autonomous vehicles.
Primary actor:	Operator
Pre-conditions:	1. User is logged in 2. Machines are reserved 3. An existing path is available
Post-conditions:	1. Autonomous vehicles are executing the path
Main success scenario:	1. User selects a path 2. User presses the execute button 3. User presses the yes button on the confirmation to execute 4. System exports path to autonomous vehicles 5. System executes the path on the vehicles
Alternative flow:	1. User presses the cancel button 3. User presses the no button 4. System error - failed to export path 5. System error - failed to execute path
Frequency of use:	Every time an operator start executing a path.
Created by:	Mathias Svensson Karlsson
Date created:	2018/11/20
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

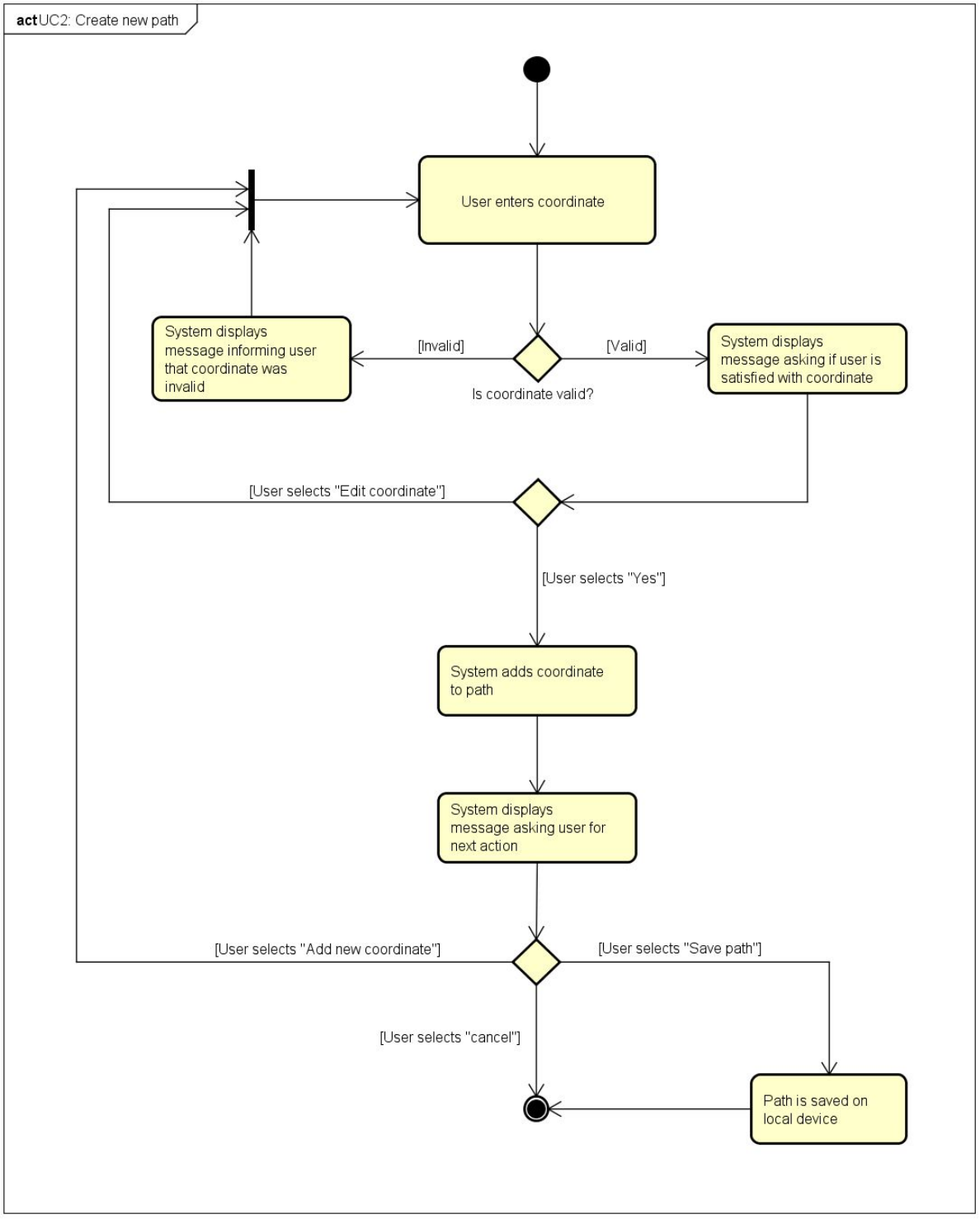
ID:	UC10
Title:	Reserve machines
Priority:	Medium
Description:	Let an operator reserve a number of autonomous vehicles
Primary actor:	Operator
Pre-conditions:	1. User is logged in 2. Existing machines are available 3. Machines are released
Post-conditions:	1. Machines are reserved
Main success scenario:	1. User clicks reserve machines 2. User selects machines from list 3. User clicks reserve 4. User clicks the yes button on the confirmation to reserve 5. System reserves the machines
Alternative flow:	1. User clicks the cancel button 2. User clicks the cancel button 4. User clicks the no button 5. System error - cannot reserve machines
Frequency of use:	Every time the autonomous vehicles are changed.
Created by:	Mathias Svensson Karlsson
Date created:	2018/11/20
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

ID:	UC11
Title:	Release machines
Priority:	Medium
Description:	Let an operator release reserved autonomous vehicles.
Primary actor:	Operator
Pre-conditions:	1. User is logged in 2. Machines are reserved 3. Path is not being executed
Post-conditions:	1. Machines are released
Main success scenario:	1. - 2. User selects the yes button on the confirmation to release 3. System releases the machines
Alternative flow:	2. User selects the no button 3. System error - cannot release reservation
Frequency of use:	Everytime the autonomous vehicles are to be changed.
Created by:	Mathias Svensson Karlsson
Date created:	2018/11/20
Last updated by:	Mathias Svensson Karlsson
Last updated:	2018/11/20

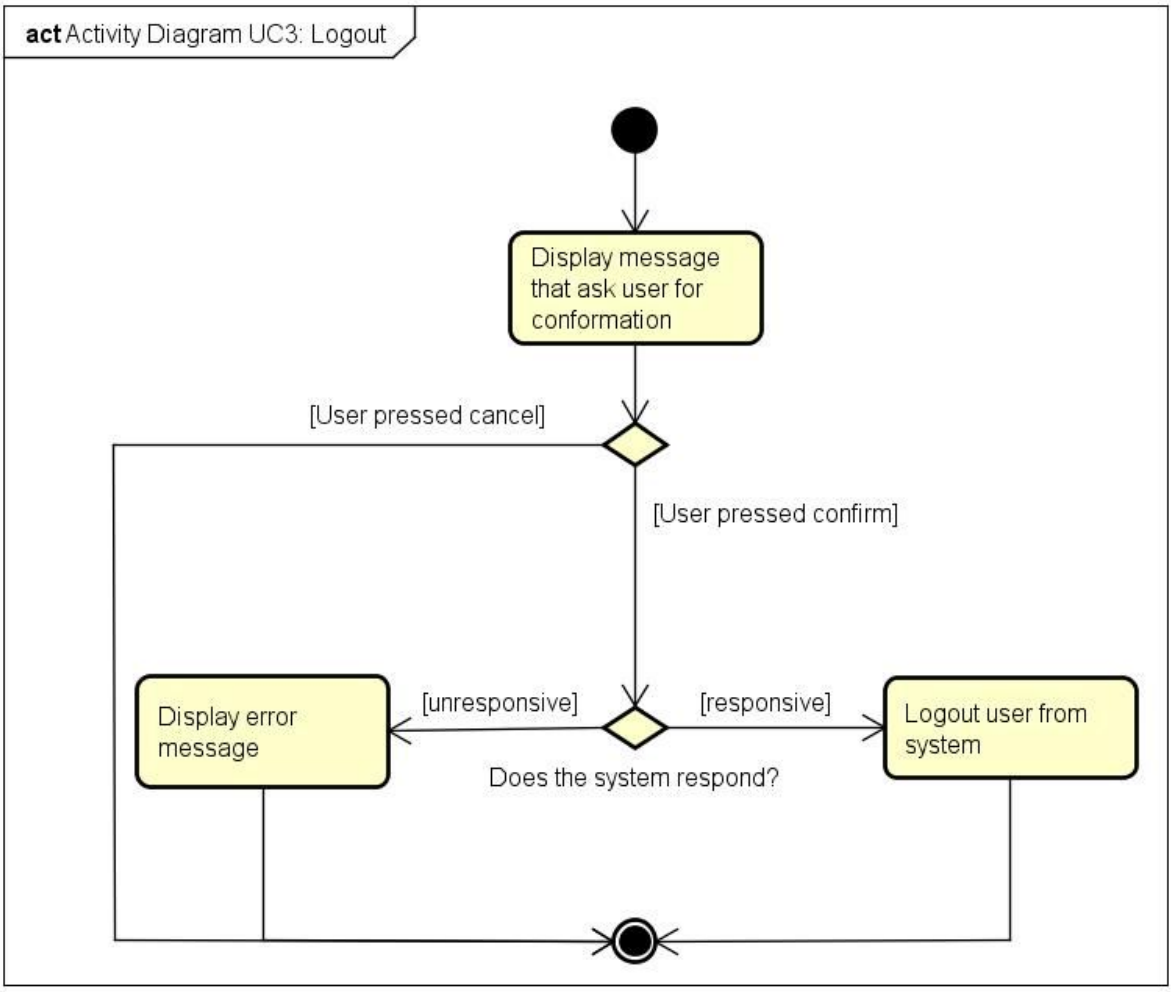
Appendix 2: Activity Diagrams



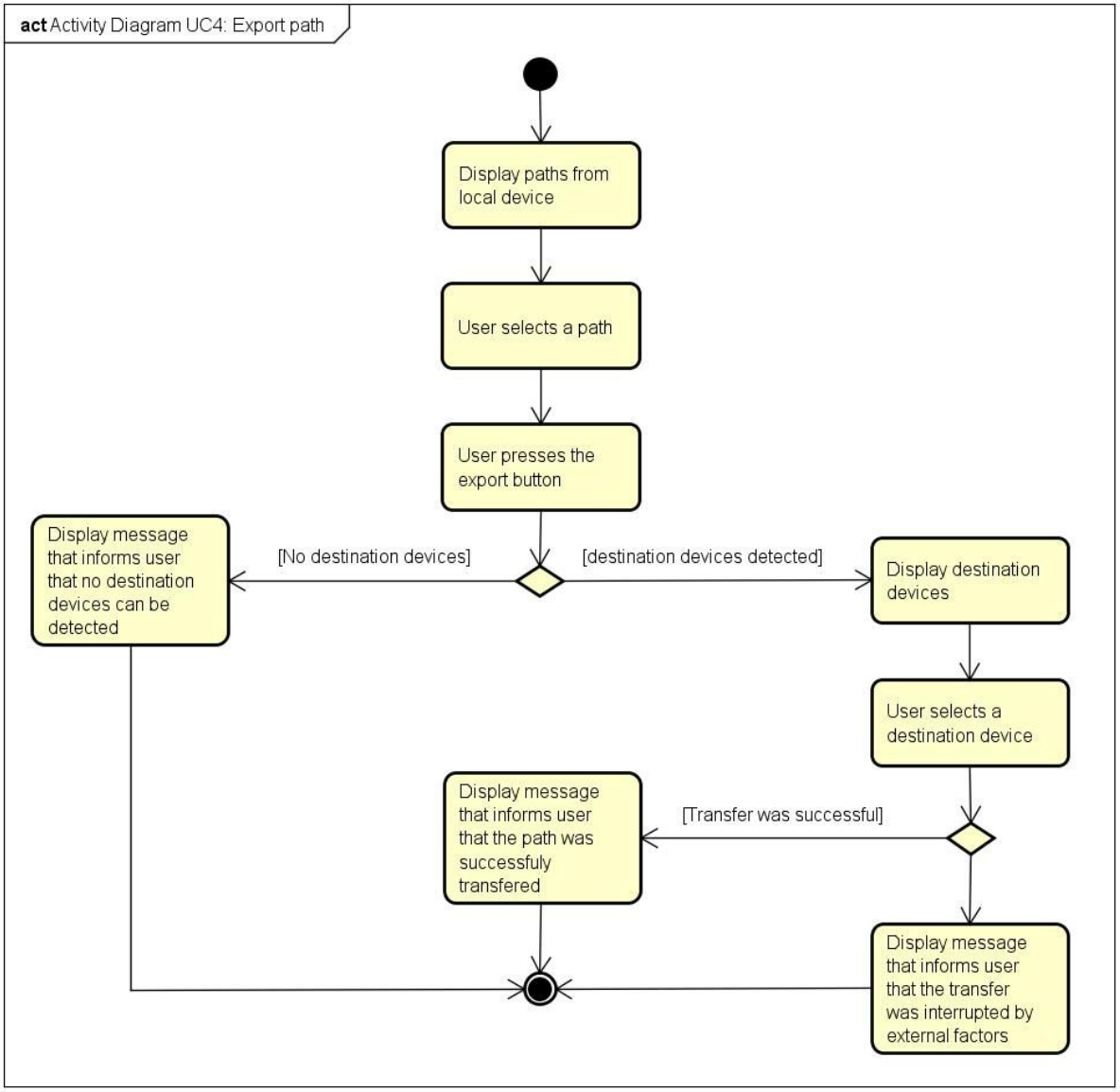
The planner and the operator have to been logged in the app, in order to know their working options, and to check that they are allowed to work with the machines. The workers must enter a username, or email, and a password.If the attempt to input the data is unsuccessful, a message will appear, letting the user know about the failed attempt. If the user has failed to enter too many times (a specific amount), the user's account will be temporarily suspended. If the login is successful, then they get access to the mainboard.



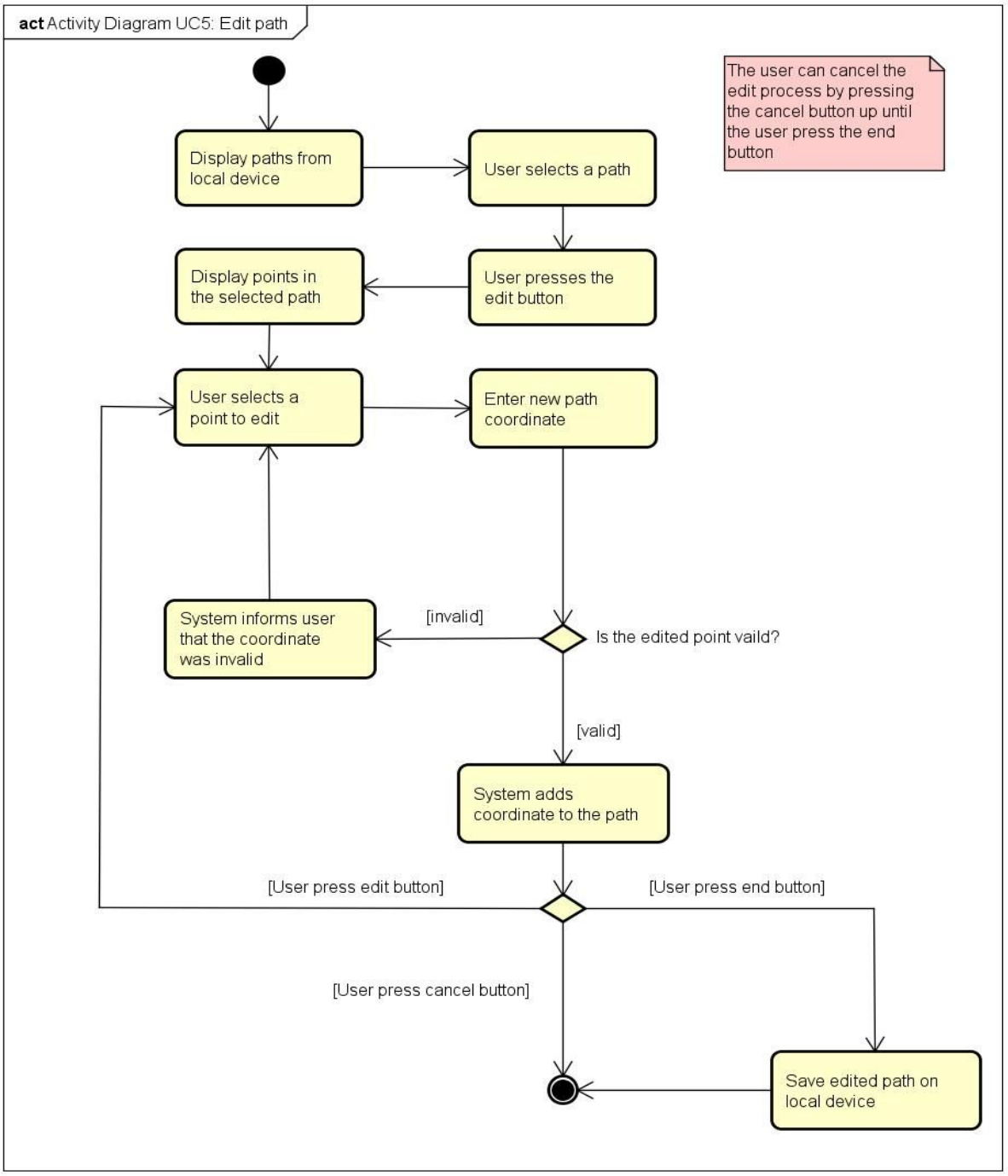
Only logged users can create and use paths. A path is the way that the machine should follow to make the work required. The logged user must enter coordinates in a map to make the path. The program should check if the coordinates are valid; if they are not, the system would alert to the user; if, on the other hand, the coordinates are valid, then the system should ask for the next action of the path, until the user selects “save path”.



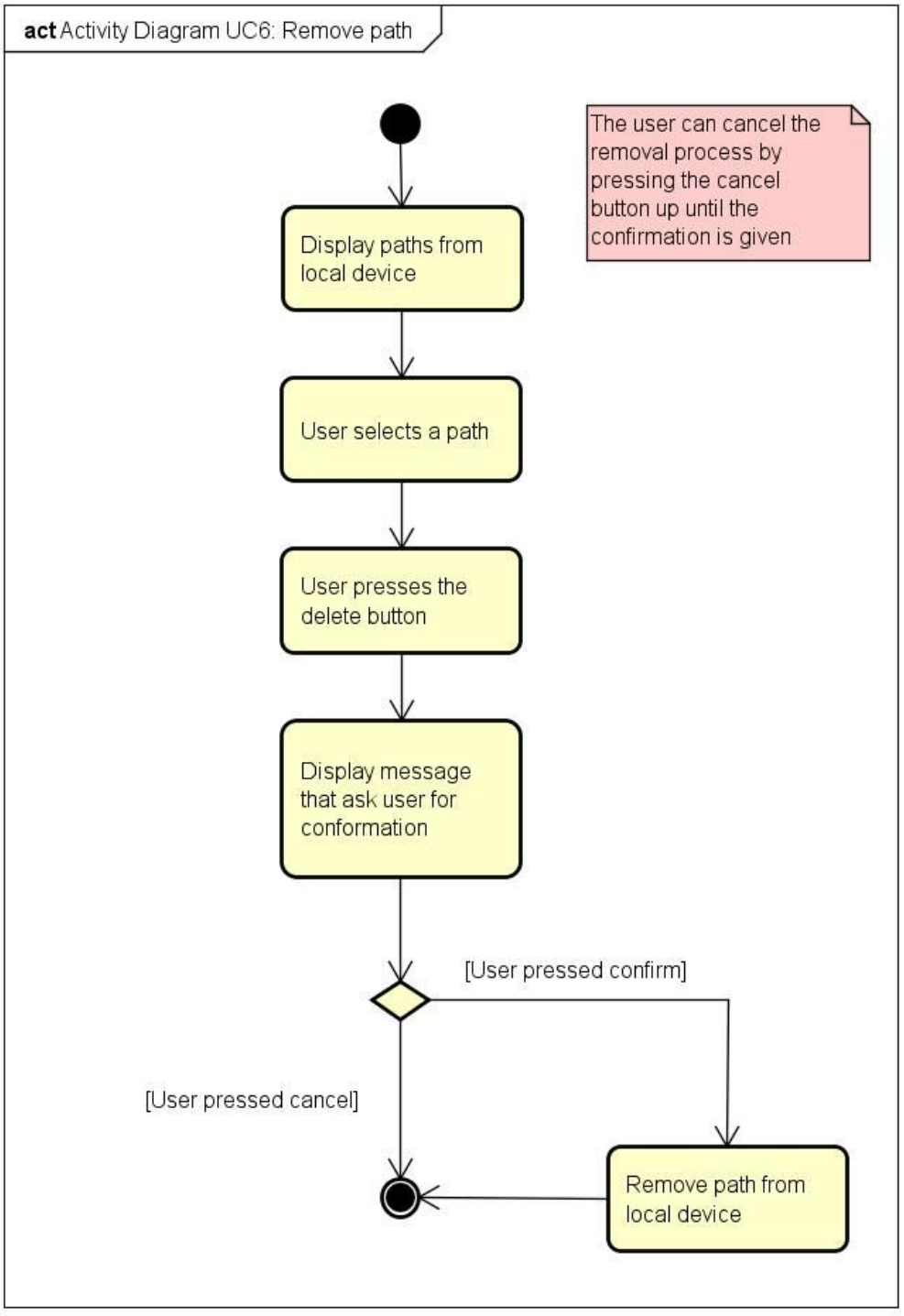
The logged users shall be able to logout when they want. To make that possible, when logout is selected, the system should confirm if the user wants to log out. If the user cancels the operation or does not answer in a period of time, then the system returns to the mainboard. If the user confirms, then the login screen is shown and the user is not logged in anymore.



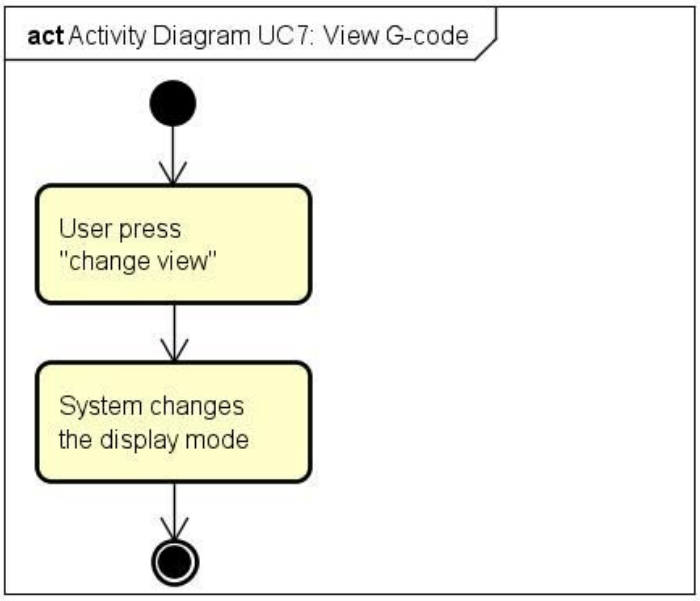
One of the functionalities the of the copilot app is that user should be able to export paths to other devices. This activity requires the users to be logged in to the system, and that the user have at least one path is stored on the local device. The user should first select the path and then press the “export button” to confirm this action. Once this happens the system should check if it can find any destination devices can be detected. If the system found destination devices it should display these, otherwise it should inform the user that no devices were detected. In the first case the user should then select the destination device, and the system should try to export the path to that device.



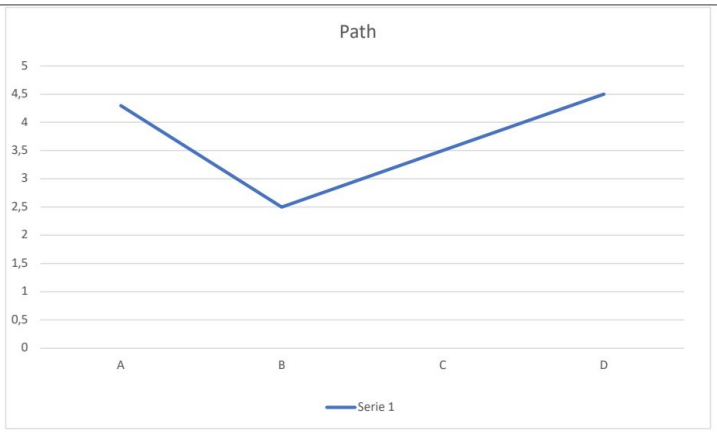
To edit a path, the user must be logged in and the path must be created. The functionality then is that the user chooses a path to edit, select “edit”, and then they can select a point to be edited. If the point edited is valid then is saved by the system, and if it is invalid, the system must inform the user that the point is not valid.

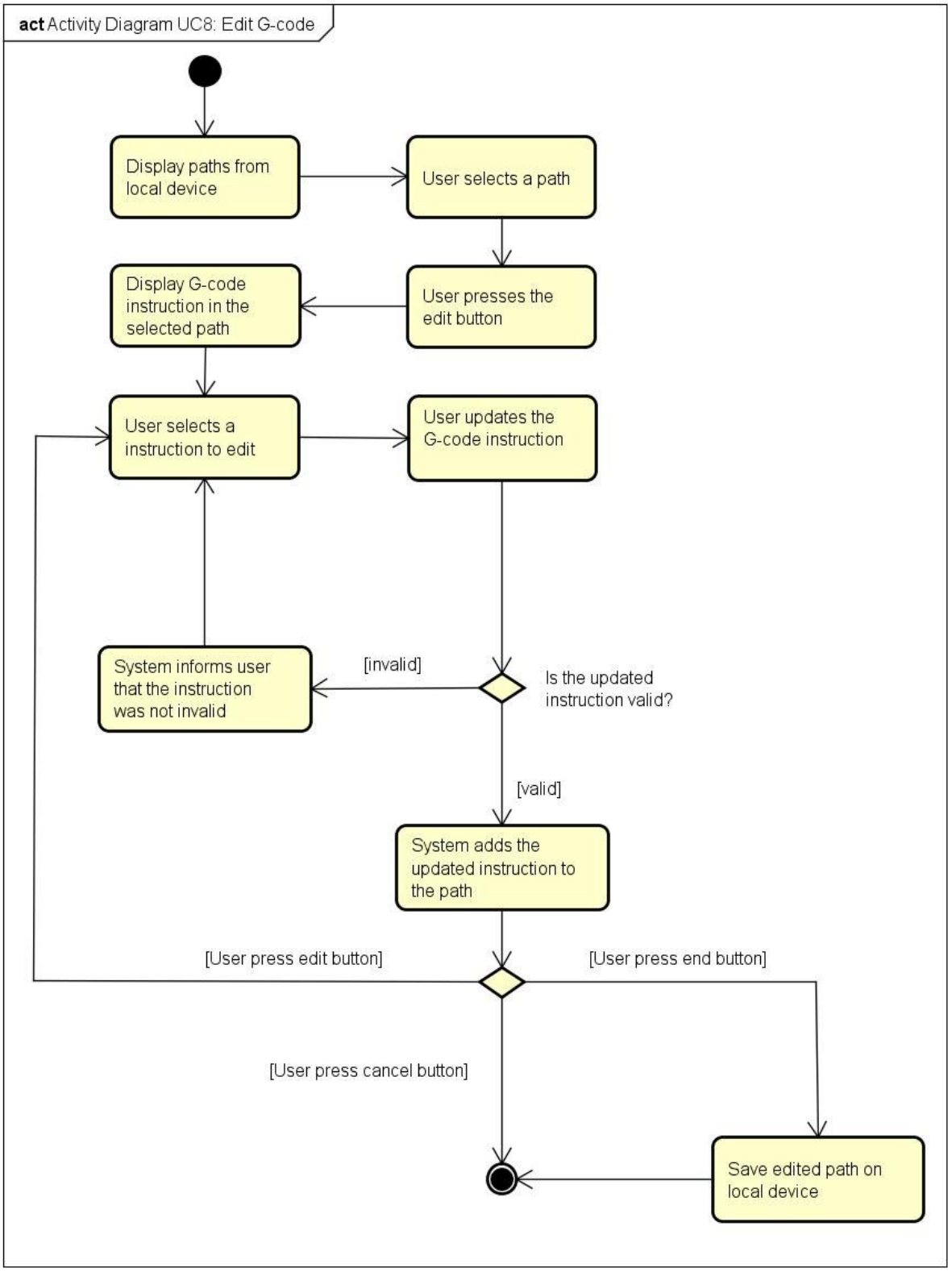


To remove a path a logged user should select a path to be removed. Then, they should select “delete”, and confirm that the path should be in fact deleted.

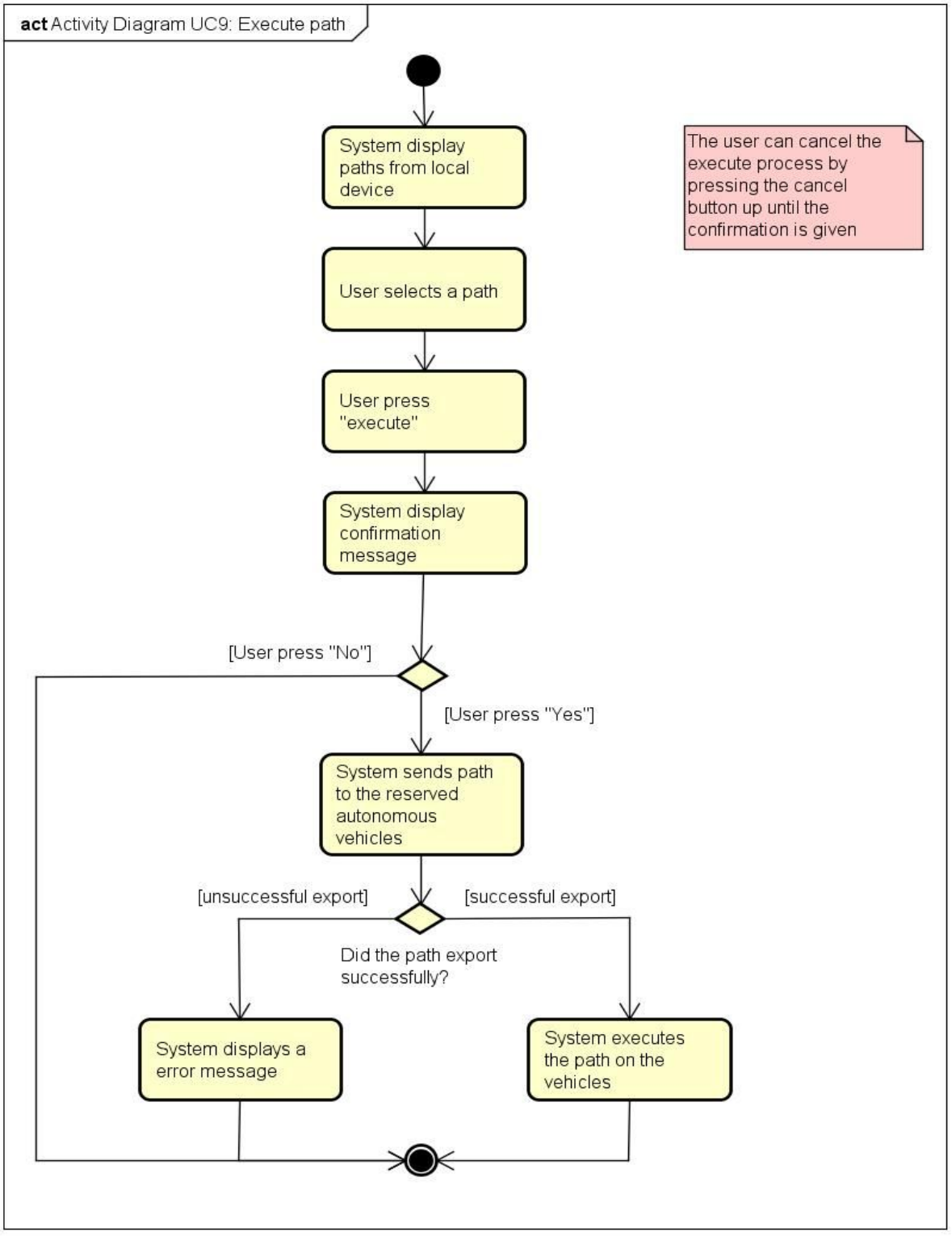


An option in the app for logged users is the ability to show a path as a G-Code; this is useful for the operator to confirm that the path is right in every aspect possible.If they select this option, the system should show a table similar to this, showing information about the path.

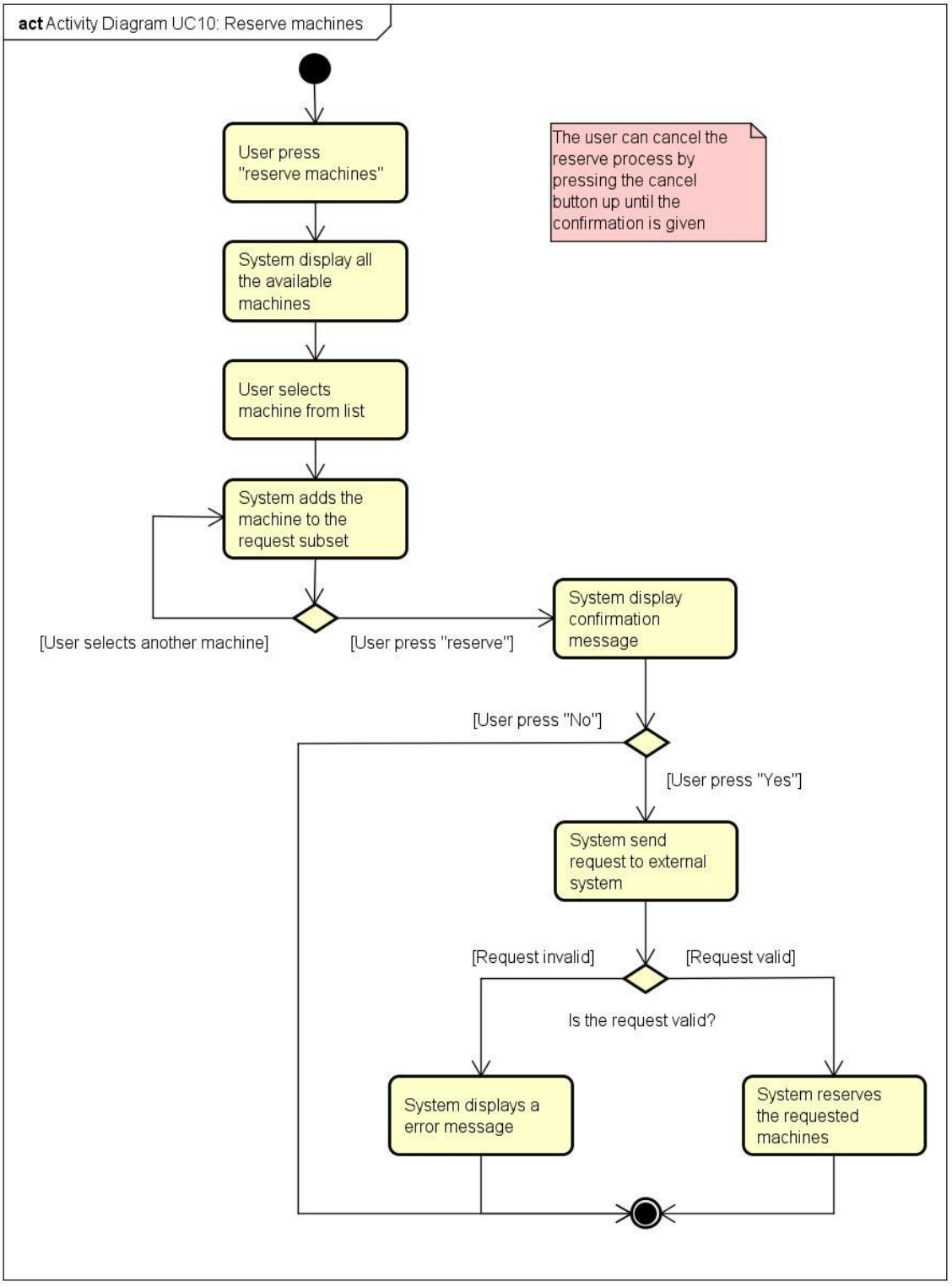
Company									Date	
Work site									Form filled in by	
Total excaveted volume, bm³ or BCY									Body volume m³ or CY	
Material									Body volume m³ or BCY	
Excavation class									Body volume m³ or LCY	
Density kg/bm³ or lb/BCY									Load mass tonnes or tons	
Swell factor									Working shifts per year	
Loading equipment									Operating hours per working shift	
Bucket volume lm³ or LCY									Productive time min/h	
Cycle time of loading equipment										
Road stretch	Length m(yd)	Gradient %	Rolling Resistance	Coefficient of traction	Curve radius m(yd)	Ground structure class	Note	Travelling time minutes		Loading
								laden	unladen	Travelling laden
A-B										Manouverung for tipping
B-C										Planned activities



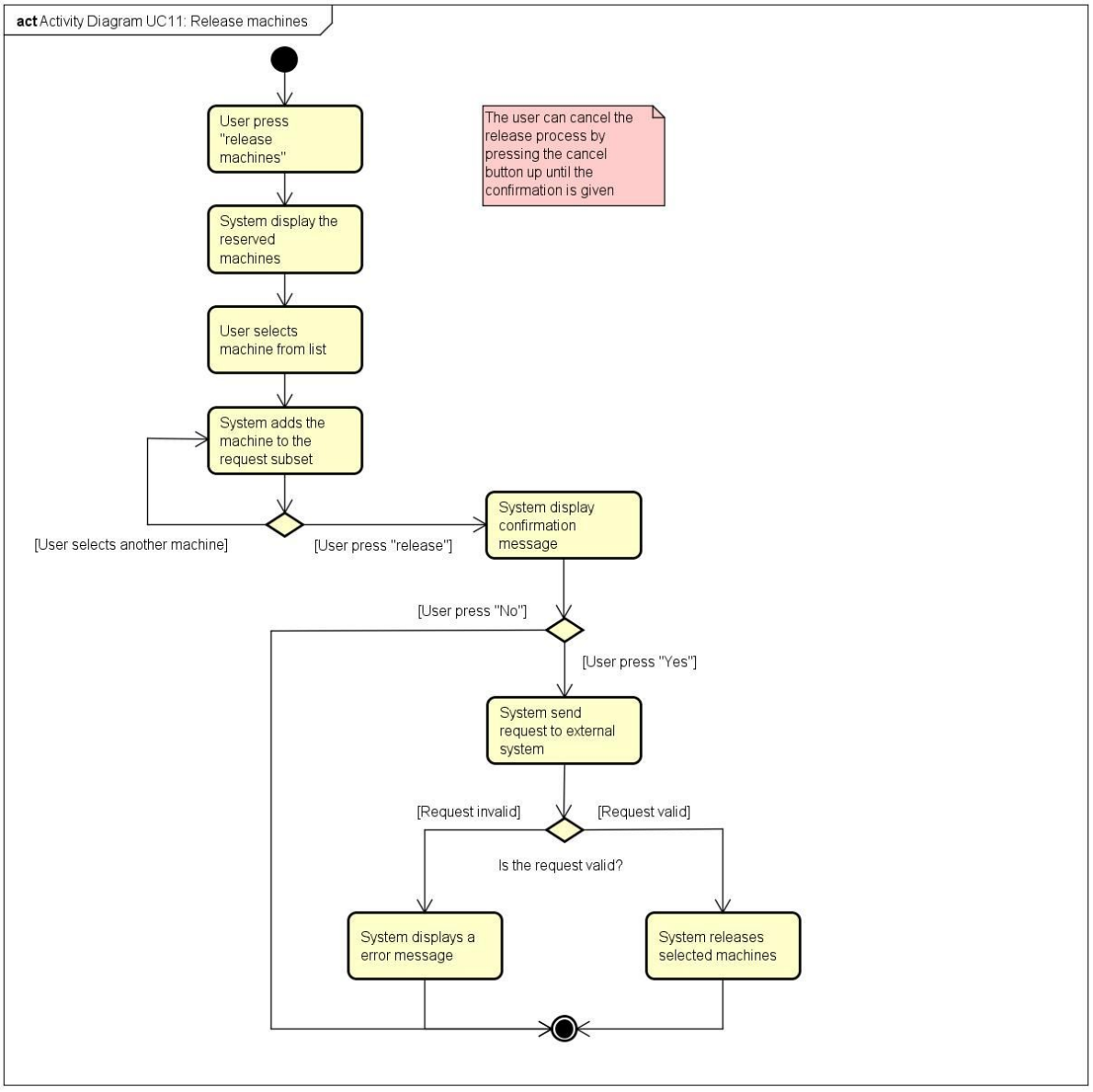
Just as edit path, this activity has the same behaviour, and the same requirements (user logged and path created), but here the user will be able to edit the G-Code.



The execute path activity, basically will make that machine start executing the G-Code so it will start working. When a user selects a path to execute, the system will ask to the client to confirm the execution, if the user cancel then the system goes to the mainboard, if accept then the system send the path to the machine, the system should return is the path has been successful export or not.



The logged users should be able to reserve machines. That means that, when the user selects “reserve machines”, the system should show all the available machines. The user should be able to select which machines they want to use to make the path, and if they are available, the system assigns the machines to the user. If one of them is not, then a error message is displayed.



The “release machines” activity is the opposite of “reserve machines”. In this case, the user wants to free the machines that they reserved before. In order to do it, the logged user must select, “release machines”; then the system should show a list of the reserved machines, the user should select which machines are going to be released.

The user should confirm about the release of the machines and if is confirmed, the system should check if it is possible, and then inform to the user if it is possible to make the operation or not.

Appendix 3: Requirements document.

This is the Project Requirements document for the CoPilot app that Team 4 was assigned to create for the Software Engineering 2: Project Teamwork course. The task is to develop an application that lets its user create paths and send them to autonomous machines for them to follow.

Client

The client is Volvo Construction Equipment. They need this application as a prototype to see if the idea they have is “doable”, as the client contact explained. If they are satisfied with the final product delivered, they will consider developing it further.

During the first meeting, the client had prepared a document with more information about what they had thought the project should involve. That document is attached to the Detailed Design Description document as an appendix.

Interface requirements

The client wanted the application to be easy to use. They emphasized the fact that the potential users will not be familiar with most of the process, so they wanted the app to be as intuitive as possible.

They wanted the paths to be made in two different ways:

- They wanted a visual interactive map in which the user could choose points in the map and create lines, curves, waiting points, pick up and unload points, etc. The selection of points could be done by physically walking the path or by interaction with the map.
- They wanted a G-code version of the map, for the workers to fill more accurately. The G-code version is written in a CNC language recognised and used by all workers, and should also be intuitive, since the original G-code table the client supplied us with was not fit for a mobile device.

The client specified that there were two main types of users:

- The planner. The planner should be able to create and edit paths, and send them to the CoPilot tablet on the control cabin so the operator can send these paths to the autonomous machines.
- The operator. The operator has every possibility the planner has, and they can also book and release machines, and send those machines the path they are supposed to follow.

There is a special user defined by the client, called “admin”, who is most probably external to the actual system and just has the possibility to delete paths from the database.

The client wants a list of paths to be the first view the user gets of the application. In this list, there should be an option to change the name and description of each path, and the option to see a preview of the path it contains. There should be buttons that lead to different activities, like edit the path, edit the G-code or delete the path.

In the list view there should also be a button to create a new path from scratch.

In the top action bar, there is a menu with three options: settings, the machines view for the operator, and the logout button.