Code review

For group 2 constructed by group 1 :)

Design

Insight: how easy the code was to read, and how to improve legibility but others than those working on the project.

You have added commentary to parts of your code, which is really nice because it makes it easier to understand the code, although some methods did not have any commentary. This was often not a problem, but for some parts the code was harder to read.
Another thing we noticed is that the variable names in ‘General_Form’ are quite similar, for example ‘SSM’ and ‘SMM’. We can imagine it would be quite easy to accidentally use the wrong variable. We would recommend changing the names so there is a smaller chance of a mixup to happen. For example: “Start_sub_menu SsubM;” and “Start_main_menu SmainM;”

Modularity: how well the code is split up in independent parts. Whether parts of the code can be reused.

Your code has been split up in a lot of different parts, which makes it really organized. The classes that belong together are easily viewable in the same folder. Most of the functions are standalone and could be repurposed for different additions. Even though some functions are intertwined it is done in a logical sense, only bringing together functions which are dependent on each other.

Extensibility: how well the interfaces are designed and documented, how easy it is to take existing components and add stuff onto them without breaking the program.*

Extension of the program was fairly straightforward, the UI allowed for the addition of multiple extensions. This without the breaking of the functionality as is. That said most of the UI has not been connected up with functionality just yet. So be sure to keep extensibility in mind whilst connecting the functionality with the UI.

Supporting Documentation: How well the design was captured in documentation, diagrams or other media, and how this can be made more clear.

The information given in the midterm report about the UI is very clear. It describes the program's functionality and how it’s meant to work very well. A complete/updated class diagram might be a great tool to capture the essence of the problem.

Tips:

- The comments that have been added are great, but some methods are missing the comments which makes it harder to read for people that have not helped with the project. for example:
  Next time make sure to have one codebase which has all the update sub categories in, instead of 4 different projects which each contain a part. Otherwise renaming the project corresponding to its content would be a nice idea as well.

Functions

Methods: How well the methods are structured and named.

The method names are very easy to understand and basically tell you what that part of the code is about. There is no problem with those names. The only thing to keep in mind would be to in some cases incorporate more dissimilarity between function names. As mentioned previously in Design. Whilst you are actively working on the project this might not pose a problem but instances like ElemSLM/SSM/SRM.Hide(). Might pose a threat of being easily mixed up. When returning to the project after a while.

One thing that we noticed is that some of your method names are in english, and some are in dutch. Make sure that all the method names are in the same language for consistency. There is the method ‘tekenAuto’ for example in the class ‘Vehicle’. This will probably not be part of your final version, but keep it in mind even whilst testing.

Construction of Methods: How well the methods are built/implemented.

The variables have good clear names.The methods are of a nice length, definitely not too long. Methods have been split up into multiple smaller methods, which makes adding alterations a lot more straightforward.

Cohesion: How well the methods work with each other.

Especially on the ‘General_form’ the methods work very nicely with each other. You’ve got the ‘Hide_all_menus’ method for example, which contains the code to hide all the UIs, which is then called in a different method. As stated in modularity: most of the standalone functions allow for repurposing for different additions. Connection of functions is done in a logical sense, only bringing together functions which are dependent on each other.

Error Handling: How well errors are handled in your code.

At the moment you do not have any noticeable error detection in your code.You make use of the already activated console for troubleshooting. Make sure that when you start adding functionality by merging the different versions you don't accidentally create errors. And possibly start thinking about program breaking scenarios like corrupt save files and how those problems could be resolved, to make the next iteration of the program more versatile.

Tips:

- Make sure to stay consistent in the language that you use for the method names.

Structure

Clarity: How well your project is organized into files, units, classed, projects, etc.

One of the things we did notice is that the project has been divided into 4 different folders. Those folders also have different visual studio projects with the same name "GreenLight".

This was not very convenient, because we had to switch between the visual studio projects everytime we wanted to see the newest version of a specific part. This also was a challenge because the projects had the same name. Keeping the main version of your program up to date could be a useful way to allow different parts, whilst keeping the main available to extend upon. This could also reduce the labor of merging bigger projects together. If you want to use different visual studio projects, it is maybe an option to give them different names.

Layout: How the code and the documentation is written.

The documentation is very well done. All the methods are written in a very clear way, and the spaces between the different methods make it pleasant to read through the code.

Tidiness: How consistent standards are applied and how little superfluous material is in the project.

There is no consistency in the amount of empty lines between new methods. We would recommend sticking to one or no empty lines between new methods for consistency. If you look at 'GridController' for example, there's a lot of empty lines between the method 'OnClick' and the method 'DrawGridPoints'. A tip would be to have a meeting to get every group member on the same page about structure.

Comments: How the comments of the code itself are handled.

There are comments above all the classes which is very nice! They give a clear description of what is in the class and its functionality. It might be a nice idea to add comments to some of the main methods in order to make your code easier to read.

In the class ‘Vehicle’ there are also some comments behind some of the constants. These are in dutch, whilst the rest of the program is in english. We’d recommend sticking with one language, just like the method names. That said, adding comments to important variables might be a great addition to make the code more readable.

Tips:

- Try to stay consistent in using empty lines between methods and the language for the comments.
- Add additional comments to important functions and variables.

Quality Assurance

Unit Testing: Whether there are tests being done and how they can be improved.

There are no unit tests as of this moment. The team has not planned to add these in the near future.

System Testing: How well builds are tested and how this is done.

They don’t use a special way to test their software. They just test it to see if it works, and try to work as neatly as possible.

CI/CD: Whether a Continuous integration/deployment pipeline is available and how this is handled.

The group uses git for CI. The git has multiple branches, on which progress on individual components is being performed. There are continuous merges with the main branch, which is a great thing to keep up. The merge comments could be more descriptive, in order to easier locate versions that might have caused potential problems.

Version Control

VCS: How well the version control system is used.

This team uses Git to submit their work to the other team members. The names of the branches and commits are very clean.

Issue Tracker: How well an issue tracker is used (such as Jira or Trello)

This team uses Trello for their ‘issues’. In Trello there is a clear overview of tasks that need to be done, which tasks they are working on and what has already been done. Furthermore, you can see the hours that they have spent on a task. The large tasks are split up into smaller tasks and every person is assigned to some tasks. Because every task is assigned to someone, it is easy to know who you need to speak to when you have a question or something.