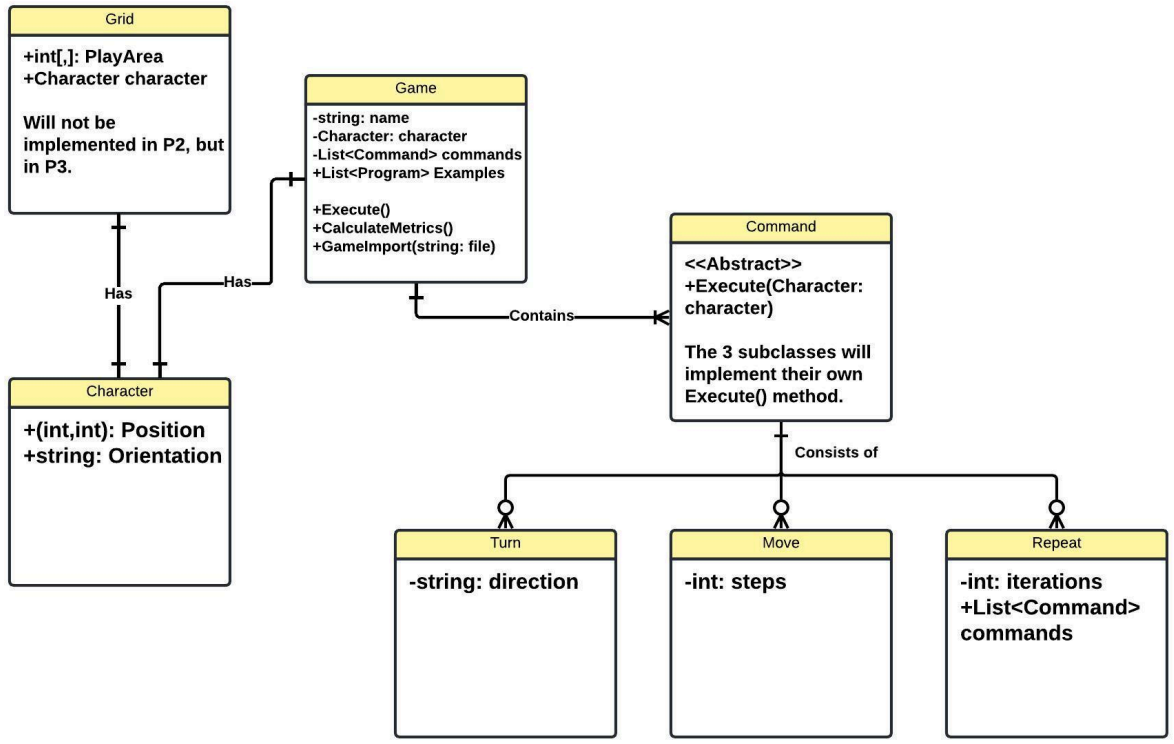
Taha Charef - 8947597
Vic ten Bokum - 1801473
Modelleren en Systeemontwikkeling
Universiteit Utrecht

Programming Learning App: Iteration 1

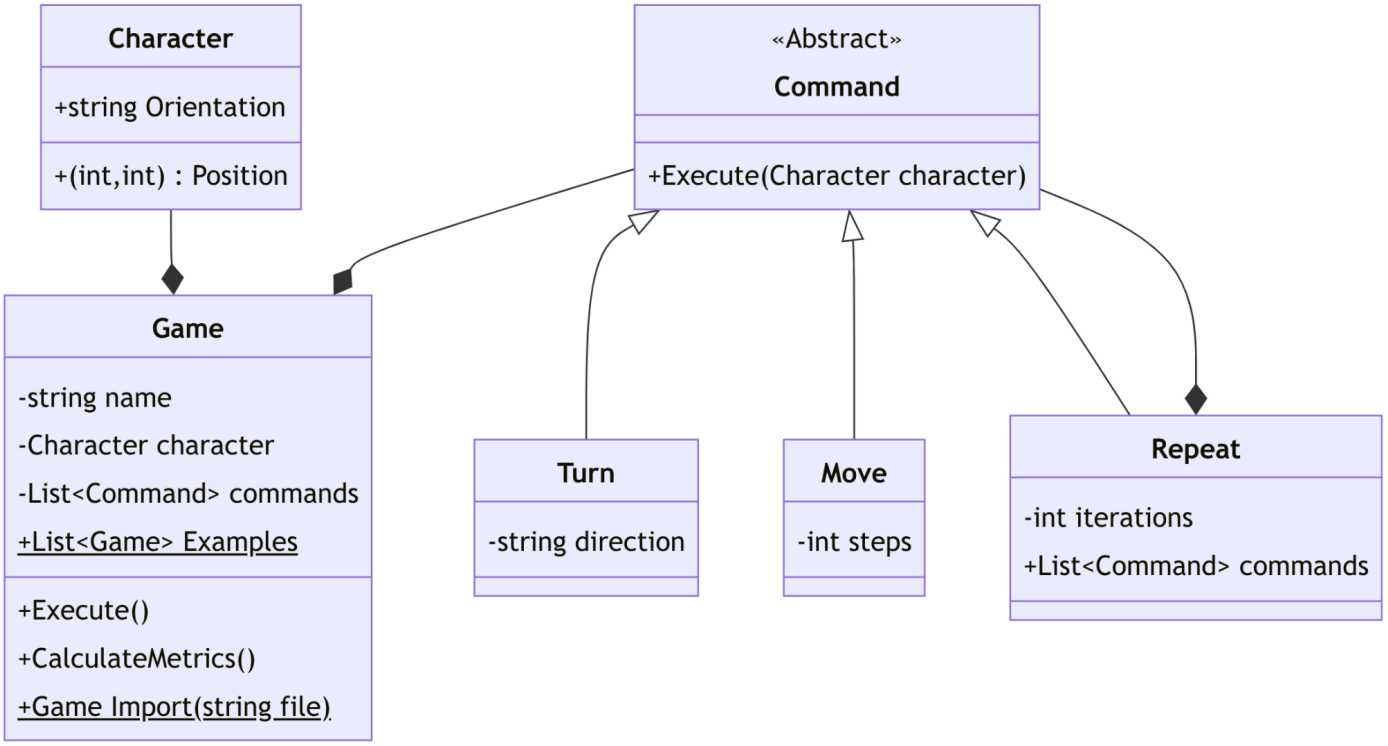
Domain model

Clarifications:

- The Grid class will not be implemented in this iteration of the project. This class will contain the elements that are necessary to render the program to a graphical user interface, which we will not implement in this iteration.



Class Diagram



Note: Some helper methods that are used in other methods are left out. This is done to prevent clutter in the class diagram.

Design evaluation:

We kept the design of the program as simple and minimal as possible, while still delivering all the necessary functionality. The variation that we encountered during the design phase of the program was very little. We experimented with different design choices, like adding a File and User class. In addition, we changed the way commands handle their execution, which was first done in specific methods for each Command, and now with the Execute() method. The result of this exploration of options made it so that the design ended up being extremely simple and elegant, with very few classes and specific methods.

An interesting design pattern that we used is the composite pattern. Using this pattern, we defined Repeat (Composite) as both a subclass of Command (Component) and as a composition of Commands, like Move and Turn (Leafs) or Repeat itself. This makes it possible to treat a Repeat command the same way as a Move and Turn command, which makes performing commands consistent and easy, regardless of the type of command.

Because this design is so simple, it is very capable of handling changing requirements. Some likely future changes to the specification are:

1. The output of the program should come in the form of a graphical user interface.
2. The format of the files that can be imported to form a program should be different.
3. Users should be able to generate random programs that can be executed.

How our design will handle these changes:

1. By simply adding a new class called Grid, which has the Character as property, we can handle all the rendering of the character on the given grid (playboard) through the use of the Character's location and its orientation. This way, the program is easily scalable and has a clear separation of logic and rendering.
2. Because all the logic of importing a file is contained in the Import() method of the Game class, only this method should change to account for the change in file format.
3. A new method like Generate() should be implemented in the Game class, that contains logic for generating a program, based on a random seed.

Changes that would be difficult to incorporate into the current design consist, for example, of fundamental changes in the way commands should work. If a command should perform some other side effects than changing the position or orientation of a character, not only the design of the Command classes should change, but probably also the Game class, and maybe even the Character class. This would require a complete revision of the implementation of the program.

The current program is designed with high cohesion and low coupling in mind. For example, the Game class might look quite large, but is in fact very cohesive. Everything that the Program class should do is implemented in that class, and nothing more, nor less. The same goes for all the other classes in the program.
In regards to coupling, we kept it as minimal as possible. The important coupled components of the design are Character, Game, and Command. These components are linked with each other at most with one property.
The use of high cohesion and low coupling keeps this design extremely open for changing requirements and scalability.

Implementation & testing

The implementation of the program, along with unit tests, can be found on https://github.com/xxheyhey/MSO-Project-UU.

Work distribution & reflection

Vic	Taha
Domain model research	Domain model creation
Clarifications of domain model	Class diagram
Design evaluation	C# implementation (together)
C# implementation (together)	Unit tests

Table 1 - Work distribution among both authors.

Reflection on our collaboration

Taha:
I had a very good experience working together with Vic once again. We complement each other and that is a good way of learning in my opinion. Especially the pair programming session allowed me to learn more while writing some of the base classes. Proper feedback from Vic and we could easily share ideas as well. There was good communication and we could help each other out whenever needed. Dividing the tasks went well too and the workload was balanced. The C# implementation took more time than we thought. Next time, we will have to plan this a bit earlier to keep time for testing and debugging. Aside from that, it was a pleasure to work with Vic on this project and I look forward to the next assignment.

Vic:
Working on this assignment with Taha went just like working together on the previous assignment: exceptionally well. We both did our parts in a timely fashion and Taha delivered high quality work. During the process, we were able to contact each other continuously, which has helped the progression of the project a lot.
We started quite late with the C# implementation of the project, which is why we finished the assignment quite late before the deadline. This is not anyone's fault, and Taha and I worked hard in the end to finish everything up. For the next time, we should think a bit longer about how much time certain parts of the assignment will take to complete. This will make it so that we can plan ahead more easily. All in all, I sincerely thank Taha for his hard work and effort.