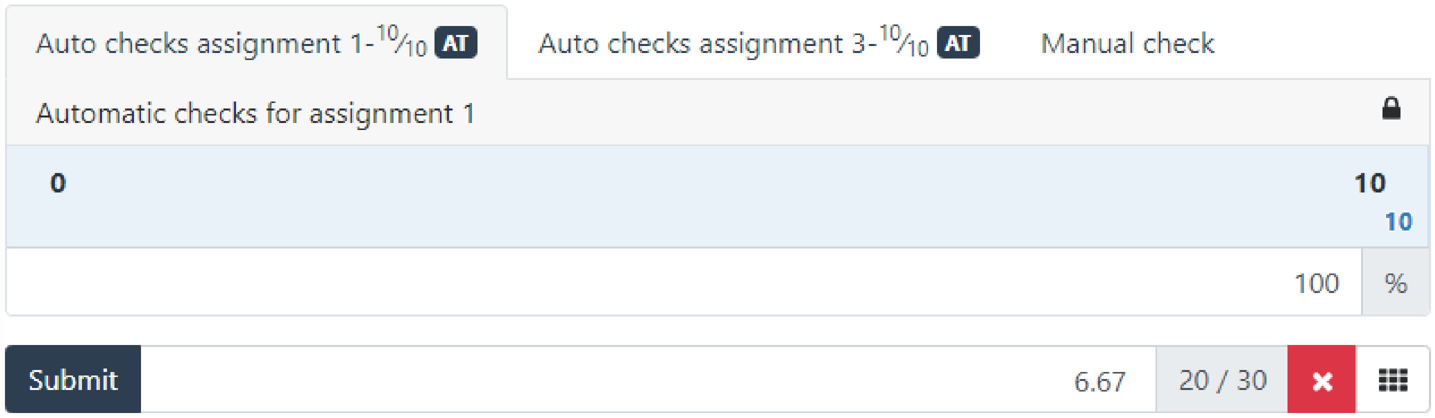
Week 4 assignments

When creating the program code, you must apply the following basic principles:

- create a separate project for each assignment;
- use name ‘assignment1’, ‘assignment2’, etcetera for the projects;
- create one solution for each week containing the projects for that week;
- make sure the output of your programs are the same as the given screenshots;

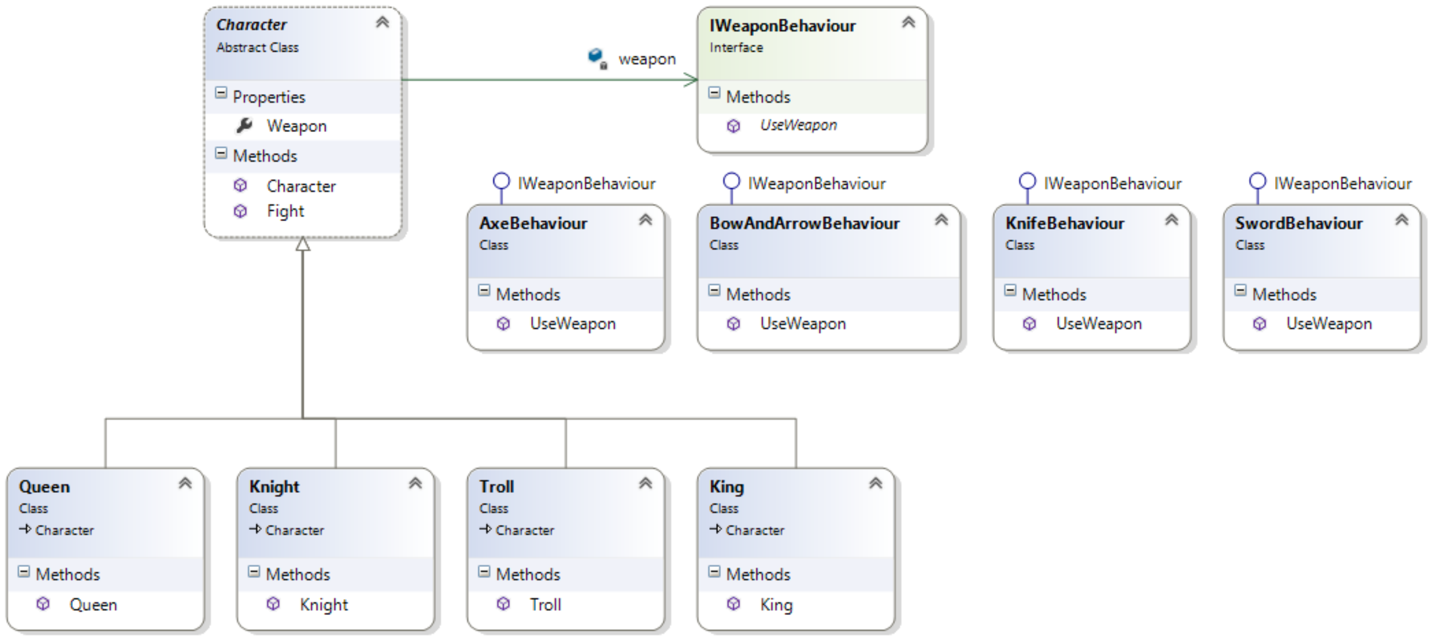
CodeGrade auto checks

Make sure all CodeGrade auto checks pass (10/10) for your assignments. The manual check will be done by the practical teacher.


Auto checks assignment 1-10/10 AT
Auto checks assignment 3-10/10 AT
Manual check
Automatic checks for assignment 1
0
10
10
100
%
Submit
6.67
20 / 30

Assignment 1 ('Strategy Pattern')

The folowing characters are present in a game: Queen, Knight, Troll and King. They all inherit from abstract base class 'Character'. Each character has a weapon to fight with. Create an interface IWeaponBehaviour and implement 4 different kind of weapons: Axe, BowAndArrow, Knife and Sword (all implementing interface IWeaponBehaviour). The class diagram below shows alle classes/interfaces.

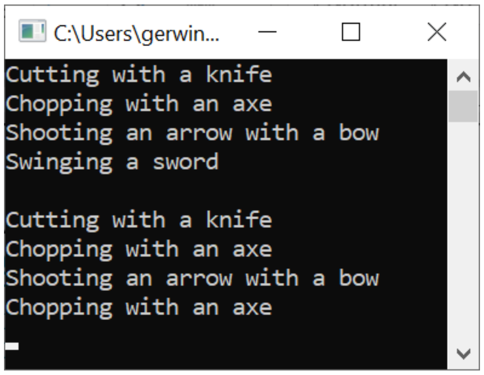


Each character has a default weapon, but this can change during the game (to another weapon).

Implement the classes/interfaces shown, and use the following main program to test it:

```
void Start()
{
  List<Character> characters = new List<Character>();
  characters.Add(new Queen());
  characters.Add(new Troll());
  characters.Add(new King());
  characters.Add(new Knight());

  foreach (Character character in characters)
    character.Fight();
  Console.WriteLine();

  // change weapon of knight to axe
  characters[3].Weapon = new AxeBehaviour();

  foreach (Character character in characters)
    character.Fight();
}
```

This code has the following output →

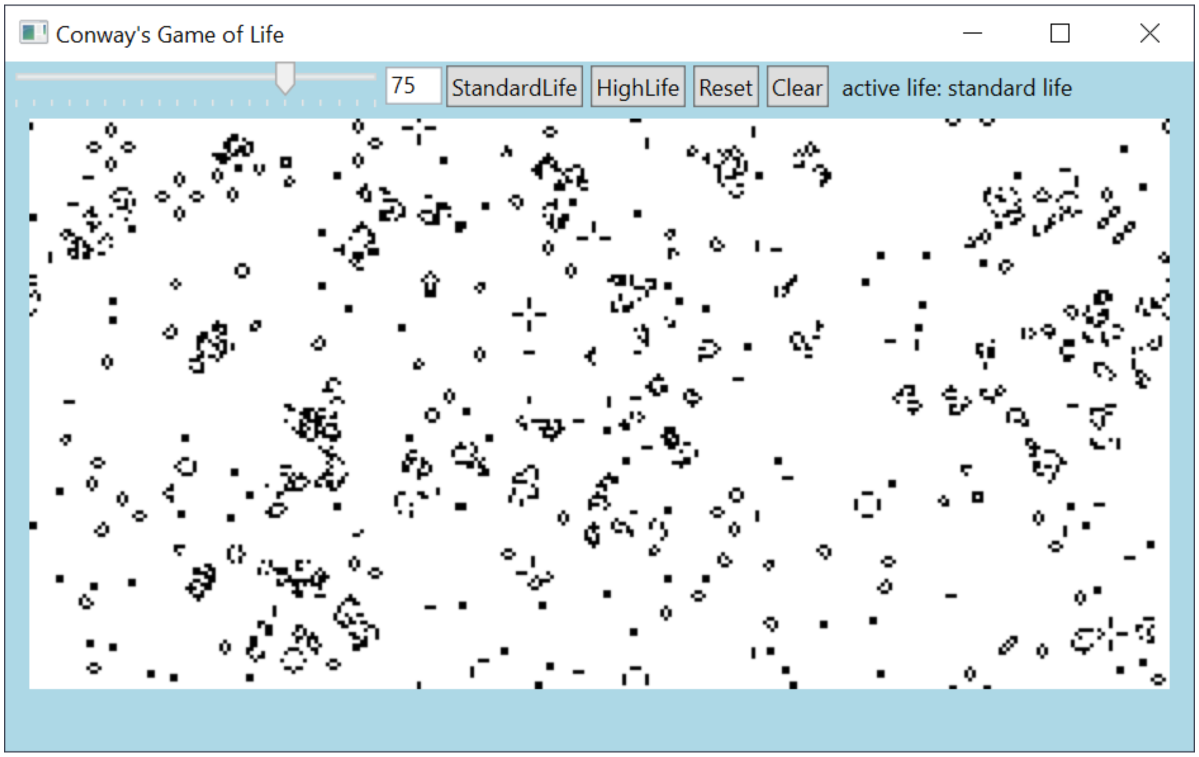


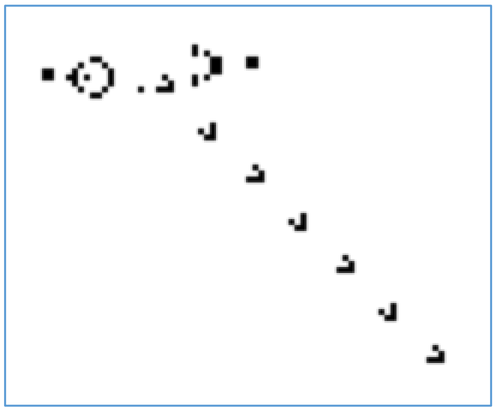
```
Cutting with a knife
Chopping with an axe
Shooting an arrow with a bow
Swinging a sword

Cutting with a knife
Chopping with an axe
Shooting an arrow with a bow
Chopping with an axe
```

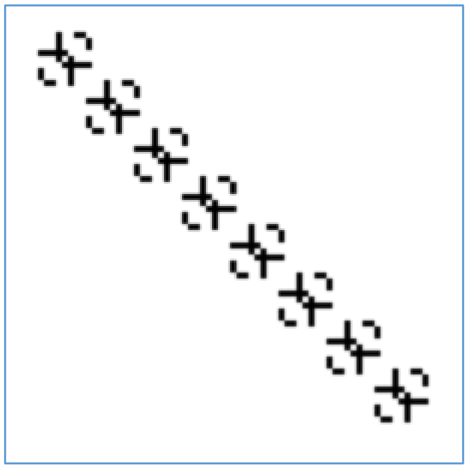
Assignment 2 ('Strategy Pattern')

On Moodle ('Week 4 assignments') you can find application 'Game of Life' (almost the same application as in week 2). The application uses the default rules of live according to John Conway (B3/S23), see screenshot below. Change this application in order to have 2 different variants: a 'standard life' and a 'high life' variant (B36/S23, see http://www.conwaylife.com/wiki/HighLife). So, the same assignment as in week 2, but now implement the 2 variants by using the Strategy pattern on class `ConwayGameOfLife`, so the behaviour can be changed dynamically (without creating a new game object). Method `CellShouldLive`, called by method `Evolve`, must be implemented by 2 separate classes: `StandardLifeBehaviour` and `HighLifeBehaviour`, both implementing interface `ILifeBehaviour`.

For more information about Conway's Game of Life, see https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life.

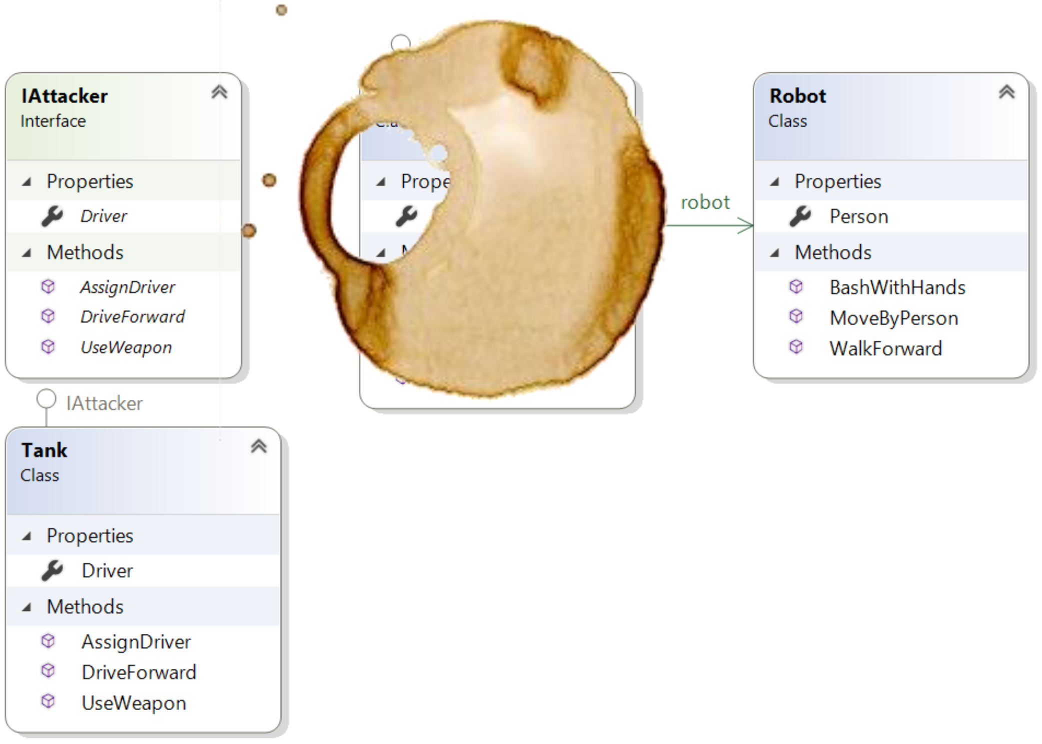



Mouse-click in Standard life will produce this pattern


Mouse-click in High life will produce this pattern

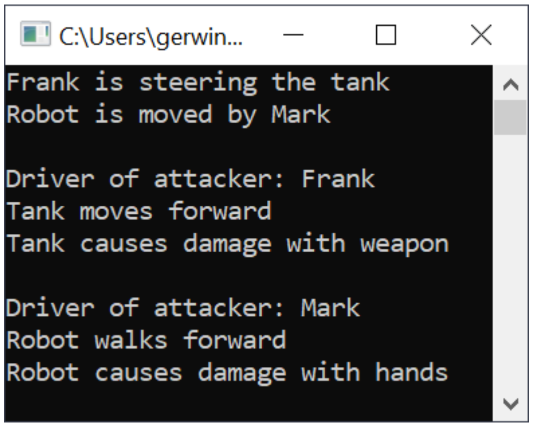
Assignment 3 ('Adapter Pattern')

In a very violent game several 'attackers' are used, to defeat enemies. An example of an attacker is a Tank, implementing interface IAttacker. In the game also a Robot must be added, although it's not really an attacker. The solution for this is to create an adapter for this robot, as shown in the class diagram below (unfortunately someone spilled some coffee on this diagram…).



Create an application that implements the above classes/interfaces.
Use the main program below, that produces the given output.

```
void Start()
{
    // create a tank (and assign a driver)
    // ...

    // create a robot
    // ...

    // create attackers list, and add tank and robot
    List<IAttacker> attackers = new List<IAttacker>();
    // ...

    // process all attackers
    foreach (IAttacker attacker in attackers)
    {
        Console.WriteLine($"Driver of attacker: {attacker.Driver}");
        attacker.DriveForward();
        attacker.UseWeapon();
    }
}
```



```
Frank is steering the tank
Robot is moved by Mark

Driver of attacker: Frank
Tank moves forward
Tank causes damage with weapon

Driver of attacker: Mark
Robot walks forward
Robot causes damage with hands
```