Opdracht week 2 **2016/11/01**

In deze tweede week wordt je gevraagd een aantal design patterns toe te passen in ons al bestaande xonix-spel. Het gaat daarbij om de patterns: Singleton, Observer en Command. Deze zijn samen met Factory besproken in het afgelopen college.

Je gaat verder werken aan het programma waarin je al een klassenstructuur voor de gameobjecten hebt gemaakt.
Door het Command-pattern toe te passen krijgt de MVC-structuur meer vorm.

Opdracht 1.
Zoek uit van welke klassen in het programma maximaal één object mag bestaan.
Implementeer voor die klassen het Singleton-pattern.

Opdracht 2.
Pas het Command-pattern toe in GameController. Maak een nieuwe abstracte klasse *Command*:

```
abstract class Command extends javax.swing.AbstractAction
                    implements java.awt.event.ActionListener
```

Maak voor elk "commando" een subklasse en definieer daarin steeds de methode

```
public void actionPerformed (java.awt.event.ActionEvent evt)
```

met de implementatie van het desbetreffende "commando".
Door de klassen te laten erven van *javax.swing.AbstractAction* en *java.awt.event.ActionListener* kunnen we hem straks gebruiken voor mogelijk vijf implementatiemethoden:

1. via keybindings;
2. via menu-items;
3. met een timer;
4. vanuit model naar view;
5. met buttons.

Met behulp van *AbstractAction* en *ActionListener* kunnen wij nu het Observer-pattern implementeren. Laat bovendien de klasse *GameWorld* erven van *java.util.Observable*, zodat veranderingen in *GameWorld* kunnen worden overgenomen door *GameView*, waarvan de declaratie dan als volgt wordt:

```
public class GameView extends javax.swing.JFrame implements
java.util.Observer
```

Koppel een "commando" van *GameController* aan de timer. Die voert voortaan dus het timetick-commando uit van de controller.
Implementeer de volgende "commando's" en sluit ze op de goede manier aan:

command	**key**	**menu**	**other**
gonorth	up		
goeast	right		
gosouth	down		
gowest	left		
increase carspeed	i		
decrease carspeed	l		
add monsterball	b	add bomb	
new timeticket	k	add timeticket	
car got new square			gameworld
collision monsterball			gameworld

collision timeticket			gameworld
new group of squares			gameworld
timetick update			timer
about game		about	
quit game	q	quit	

Een voorbeeld van een commando-implementatie is:

```
public class GoNorth extends Command
{

    @Override
    public void actionPerformed (java.awt.event.ActionEvent e)
    {
        gw.car.setHeading (90);
    }
}
```

Die kun je als volgt binden aan een toets van het toetsenbord:

```
gv.map.getInputMap (javax.swing.JComponent.WHEN_IN_FOCUSED_WINDOW).
put (javax.swing.KeyStroke.getKeyStroke (java.awt.event.KeyEvent.VK_UP, 0),
"goNorth");
gv.map.getActionMap ().put ("goNorth", new GoNorth ());
```

Zoek zelf in de documentatie van Java op wat dit betekent. (Hint: https://docs.oracle.com/javase/tutorial/uiswing/misc/keybinding.html)

Opdracht 3.

Ga verder met het verbeteren van het programma in het algemeen. Probeer niet alleen de programma-source te verbeteren maar besteed ook aandacht aan de goede werking userinterface.

Tot slot

Na uitwerking van deze opdrachten heb je weer meer geleerd over objectgeoriënteerd programmeren in Java, begrijp je het gebruik van het framework Swing beter en kun je de patterns Singleton, Command en Observer toepassen.
De deadline voor de opdrachten van deze tweede week is vrijdag 25 november 2016. Je kunt je oplossing laten goedkeuren door de practicumdocent en uploaden via BlackBoard.