Assignment 1-Group 30

1.1

1. We read our requirements and searched them for nouns.
2. We determined which of the nouns can be used as candidate classes and wrote them down.
3. Filtered the list of possible classes for similar classes and classes we won't need.
4. We made CRC cards from the new classes. See below.
5. We compared the new classes with our current classes. We'll describe the differences below the classes.

Class	**Superclasses**	**Subclasses**
Board		
Purpose	**Collaborators**	
Draw board		
Initialise board	Gem, Combination	
Swap gems	Gem, Combination	
Delete combination	Combination, Gem	
Fill empty cells	Gem	

Currently our board class has a lot of responsibilities, where in this new design it's a lot less. We might wanna try to split off some responsibilities of the board class to other classes.

Class	**Superclasses**	**Subclasses**
Game		
Purpose	**Collaborators**	
Initialise game	Board	
Handles events	Board, Sound	

In our current code this looks a lot like our gamelogic class, although the gamelogic class also keeps the time and draws the time and score. It could be an improvement to remove some of those responsibilities to other classes.

Class	**Superclasses**	**Subclasses**
Combination		
Purpose	**Collaborators**	
Checks combinations	Gem	
Give points for combinations	Score	

In our current design a lot of these responsibilities get handled by the board class. We can lower the board classes responsibility by creating this class.

Class	**Superclasses**	**Subclasses**
Gem		Special gem
Purpose	**Collaborators**	
Initialise gem		

Alter gem		

We already have a Gem class that does exactly this.

Class	**Superclasses**	**Subclasses**
Special gem	Gem	
Purpose	**Collaborators**	
Initialise special gems	Gem	
Alter special gems	Gem	

We havn't implemented special gems yet, when we do, this class would be extremely usefull.

Class	**Superclasses**	**Subclasses**
Time		
Purpose	**Collaborators**	
Initialise time		
Update time	Game	

Time is one of the classes we can create to lower the responsibility of gamelogic.

Class	**Superclasses**	**Subclasses**
Sound		
Purpose	**Collaborators**	
Sound effect	Game	
Background music		

We currently have a soundclass but it isn't used. The sound file gets loaded every time we play a sound, we should improve this by making use of a seperate sound class.

Class	**Superclasses**	**Subclasses**
Score		
Purpose	**Collaborators**	
Initiate score		
Update score	Combination	
Update highscore	Game	

We're not using a score class at the moment, we do have a highscore class for keeping track of the highscores. The board class currently keeps the score, we could split it off to a seperate score class, but it isn't that much code.

1.2

Class	**Superclasses**	**Subclasses**
Board		
Purpose	**Collaborators**	
Draws the Board	Gem	
Fills the Board	Gem	

Removes gems	Gem	
Replaces removed gems	Gem	
Swaps gems	Gem	
Finding combinations	Gem	
Update score		

Class	**Superclasses**	**Subclasses**
GameLogic		
Purpose	**Collaborators**	
Initiate Board	Board	
Initiate Scores	Board	
Update input	Board	
Update time		
Draw score		
Draw time		
Draw highscore		

Class	**Superclasses**	**Subclasses**
Gem		
Purpose	**Collaborators**	
Making 6 different gems		
Change gems		

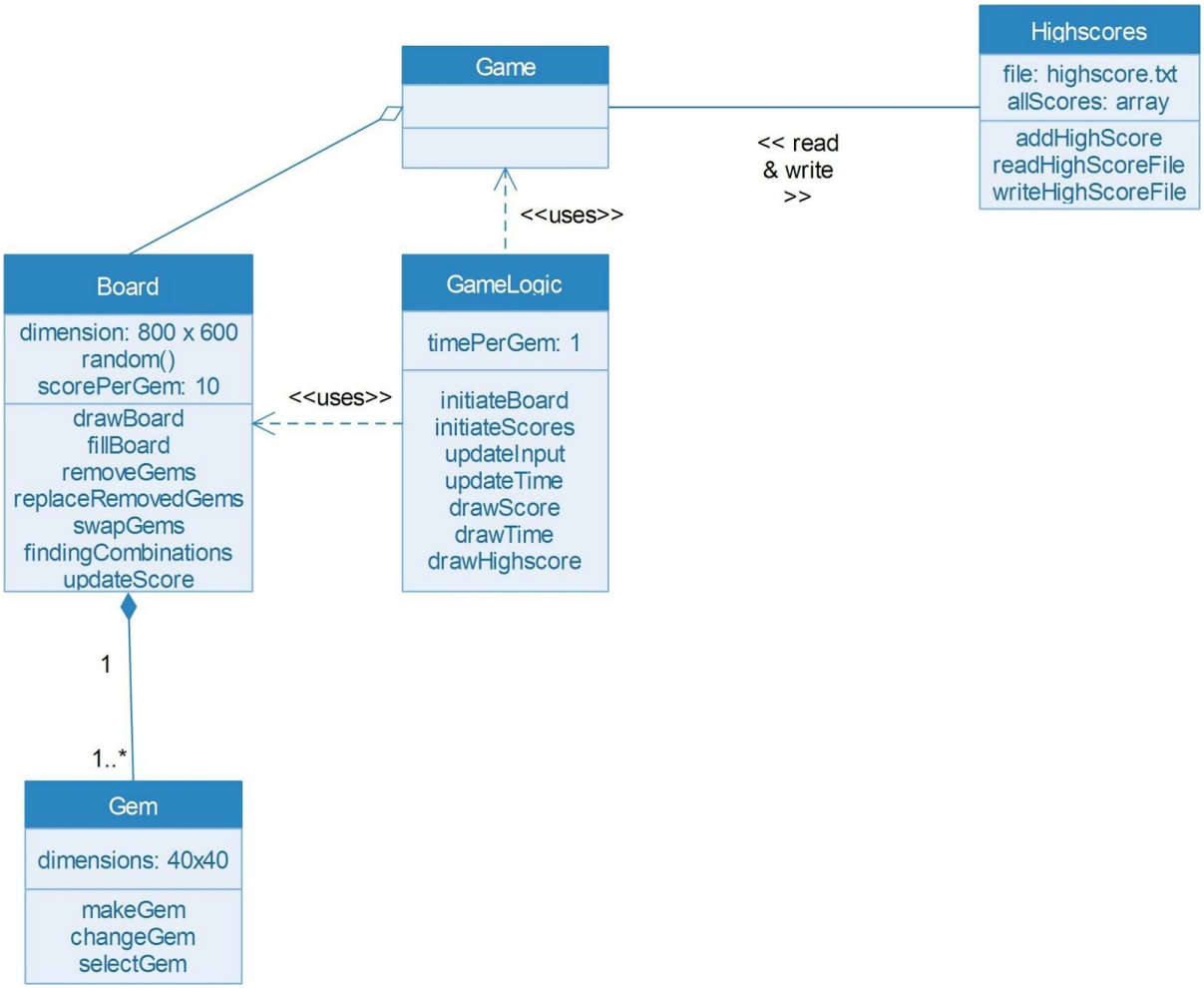
Class	**Superclasses**	**Subclasses**
HighScores		
Purpose	**Collaborators**	
Add score to highscores file		
Writes the highscore file		

1.3

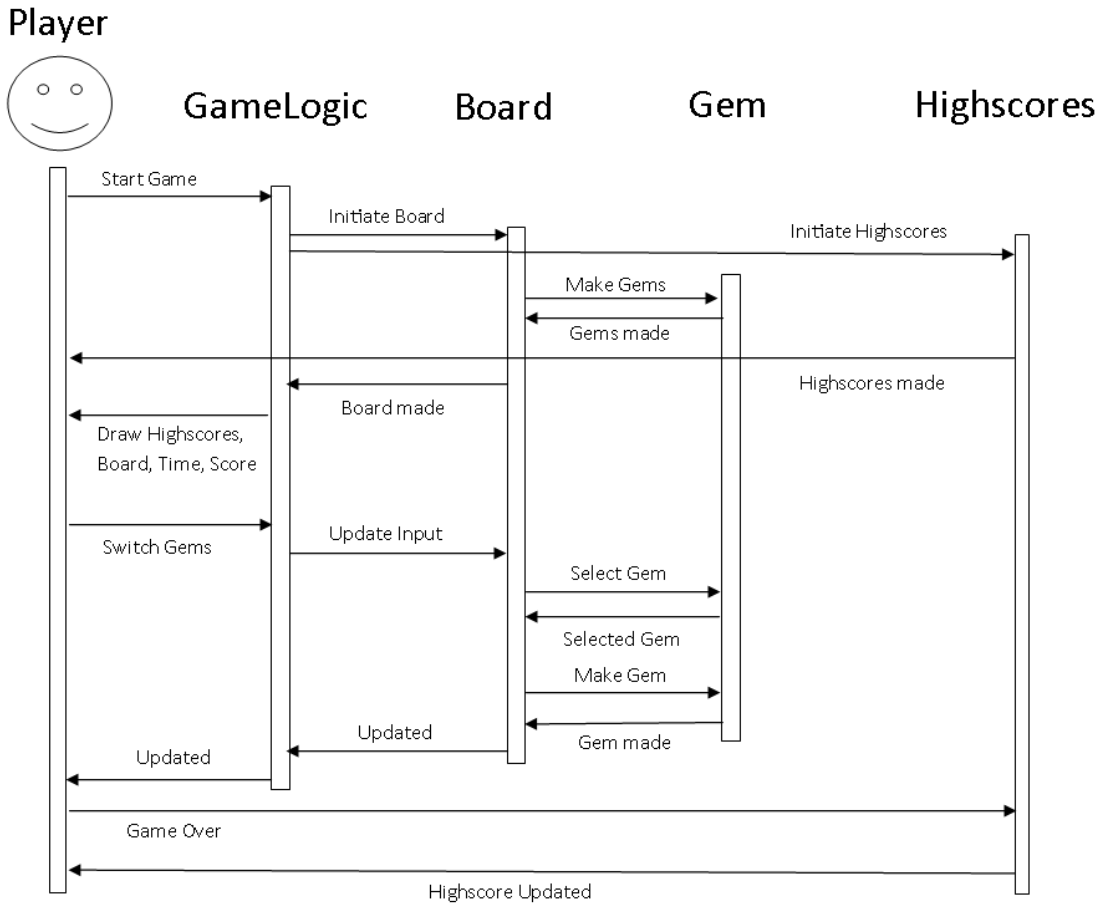
One of the less importent classes is the sound class, it currently has no purpose. We did figure out that we probably do need a sound class, so that is one of the changes we're gonna make. Other less important classes have gui responsibilities, like switching screens and drawing. We do think that we miss some less important classes, like a time and sound class. These could help to lower some of the responsibilities of the board and gamelogic class. We also thought about making a Combination class, but this didn't turn out to be benificial at all so we deleted it.

So for the code changes we added a sound and time class. We chose not to implement a score class.

1.4



1.5



2.1

What is the difference between aggregation and composition?
Both aggregation and composition are relationships between two classes. If two classes have an aggregation relationship they depend on each other, however they can exist without the other. In a composition relationship one class cannot exist without the other.

Where are composition and aggregation used in your project? Describe the classes and explain how these associations work
An example of an aggregation relationship is Sounds. Sounds is called by GameLogic and Main, so has an aggregation relation to both. If for example GameLogic dissappears Sounds will still be working for Main. An example of a composition relationship is Gem. Gem is only called by Board, so if there is no board Gem will do nothing.

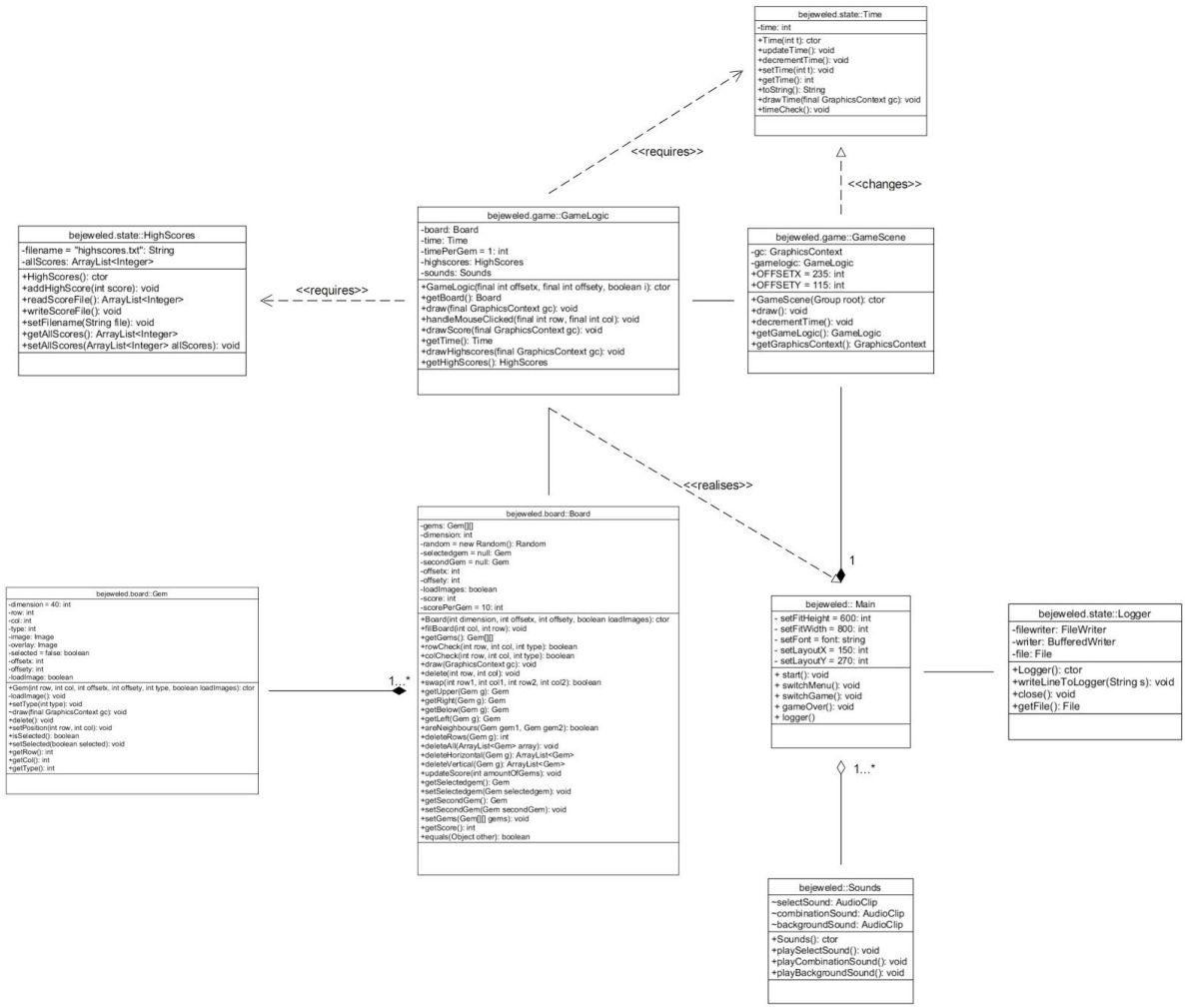
2.2

Is there any parametrized class in your source code? If so, describe which classes, why they are parametrized, and the benefits of the parametrization.
We do not have parametrized classes in our code.

If not, describe when and why you should use parametrized classes in your UML diagrams.
An example of a situation in which a parametrized class would be a good fit is when the

game has accounts. So each player has to make an account. It could be that the it is preffered the players only use letters in theire account name. Then it is smart to confine the input array to letters, which can be done by using a parametrized array.

2.3



3.1

Requirements logging

- When the user starts a game, a logging file will be created.
- When a logging file is created, the name of the file will be the current date and time in the format “YY-MM-DD-hhmmss.txt”
- When the user clicks, the position of the click will be logged. The position will be the indexes within the 8x8 grid.
- When 2 gems are switched, log the indexes of the 2 gems and if the switch succeeded.
- When a combination is formed, log the type, the amount, and the score granted of the combination.
- Log when the game is started.

- When the game is over, log the final score.
- Every log will start with a time-stamp.

Non-functional requirements

- We'll create a seperate logging class, the main class will hold an instance of this class for all other classes to use.
- We'll use a bufferedwriter so we don't need a system call for every line.

3.2
design:

Class	**Superclasses**	**Subclasses**
Logger		
Purpose	**Collaborators**	
Create a new log file		
Log all events that happen in the Game	GameLogic, Main